SMASH: a Supervised Machine Learning Approach to Adaptive Video Streaming over HTTP

Yusuf Sani, Darijo Raca, Jason J. Quinlan, Cormac J. Sreenan
School of Computer Science and Information Technology
University College Cork, Cork, Ireland
Email: {ys8, d.raca, j.quinlan, cjs}@cs.ucc.ie

***Abstract*—The growth of online video-on-demand consumption continues unabated. Existing heuristic-based adaptive bitrate *(ABR)* selection algorithms are typically designed to optimise video quality within a very narrow context. This may lead to video streaming providers implementing different ABR algorithms/players, based on a network connection, device capabilities, video content, etc., in order to serve the multitude of their users' streaming requirements.**

In this paper, we present *SMASH*: a Supervised Machine learning approach to Adaptive Streaming over HTTP, which takes a tentative step towards the goal of a one-size-fits-all approach to ABR. We utilise the streaming output from the adaptation logic of nine ABR algorithms across a variety of streaming scenarios (generating nearly one million records) and design a machine learning model, using systematically selected features, to predict the optimal choice of the bitrate of the next video segment to download. Our evaluation results show that not only does SMASH guarantee a high QoE but its performance is consistent across a variety of streaming contexts.

***Index Terms*—SMASH, HAS, HTTP Adaptive Streaming, DASH, Machine Learning**

I. Introduction

Video-on-demand providers continue to grow their dominance in the area of subscription-based online video streaming services. These providers target a variety of customers distributed around the world. This diversity of customers poses a huge technical challenge in providing a high level of Quality of Experience (QoE) globally. As the underlying transmission medium characteristics can vary quite dramatically, these providers encode their video content into a discrete number of quality levels and implement an adaptive video content delivery mechanism to maximise QoE. Currently, the most prominent adaptive video streaming technology is HTTP Adaptive Streaming *(HAS)*.

Traditionally, the algorithms that adapt the quality of video content to the available network capacity, are categorised into buffer-based, throughput-based or a hybrid of both. Each of these Adaptive Bitrate *(ABR)* selection algorithms is optimised, through significant tuning, to improve QoE for a given streaming context. Several studies [1], [2] have found these schemes do not perform uniformly across a multitude of network conditions and QoE requirements, because the tuned parameters are sensitive to a change in network channel conditions. This lack of generalisation is found to increase re-buffering ratio, video quality fluctuation and start-up delay whenever the statistical distribution of the underlying transmission medium changes [1]. Recently, several machine learning *(ML)* based ABRs have been proposed. These models promise to enhance the ability of an ABR to generalise its performance across different network channels. These ML-based ABRs are designed to either dynamically reconfigure the parameters of an existing ABR [1] or rely on a predictive agent trained to optimise a particular QoE model [2]. However, the former typically consider a restricted set of network conditions and/or rely on a particular ABR algorithm, while the later are either computationally intensive, or rely on a small dataset, hence their performance do not generalise.

An intuitive solution is to determine how the current state-of-the-art algorithms maximise QoE and then aggregate their functionality into a single implementation. This can be a quite daunting task as the dynamics of the underlying QoE metrics optimised by each ABR may be opposite in nature. For example, it is well-known that there is a trade-off between maximising video bitrate and rebuffering events, and between responsiveness and video quality fluctuation. Hence, we contend that an easier and more effective approach can be realised when an *ABR predictive model is trained to learn the art of streaming from a large collection of state-of-the-art ABRs*, where each of the ABRs is optimised to serve a specific streaming context.

In this paper, we present *SMASH*: a **S**upervised **M**achine learning approach to **A**daptive Video **S**treaming over **H**TTP. SMASH is a simple and lightweight client-side learning-based video quality adaptation predictive model. To build SMASH, we generated data from streaming real video using nine well-known ABRs across a variety of scenarios. The dataset is composed of a systematically selected set of features that relate to various aspects of the underlying bitrate selection decision processes, such as throughput, buffer-level and encoding rate. For evaluation, we deployed SMASH into a production grade player, our results show SMASH consistently outperforming the state-of-the-art players.

The following outlines the remainder of the paper. Section II, introduces the background and related work and is followed by Section III, where we discuss data generation, feature selection and the model training. Experimental results are then presented in Section IV, while Section V concludes the paper.

II. Background and Motivation

HTTP-based adaptive streaming *(HAS)* divides a video file into a number of segments of video content (also known as segments). Each segment is encoded into multiple bitrates to facilitate access for clients with differing stream requirements. A client sequentially request a video segment with the highest bitrate that the estimated network capacity should be able to sustain. ABR is the process through which a client decides the optimal bitrate of segment to download.

The traditional heuristic-based approach to the design and implementation of ABR is based on a fixed set of rules that do not capture the nuances of the variety of streaming scenarios that can occur in the production environment, and typically performs sub-optimally when the intended operational context changes [1], [2]. Recent attempts to help ABRs improve their ability to generalise their performance relies heavily on machine learning techniques. At a high level, ML-based ABR schemes can be classified into two groups. The first group dynamically reconfigure the parameters of an existing ABR. In [1], a scheme that pre-computes the parameters of an ABR is proposed. The scheme helps an ABR to dynamically adapt its parameters based on a pattern of changes in the network condition. In [3], a deep learning based scheme for tuning the parameters of ABRs, when streaming context changes, is presented.

The second set of schemes, share similarity with SMASH in that they rely on trained predictive ABR models for bitrate selection decisions. In [4], a composite classification scheme is used to map network-related features to bitrate. This scheme marked an important milestone for the implementation of supervised ML-based ABR. However, it has some drawbacks. First, both the features engineering and the ML algorithm selections were not implemented in a systematic way. Secondly, the trained model is designed to help, rather than replace, the existing fixed-rule based adaptation algorithms. A recently proposed model-free scheme [2], called Pensieve, utilises a reinforcement learning technique to train a neural network based ABR. The scheme makes no explicit assumption about the operational context. Several papers have reported issues with Pensieve , hence we conducted a comprehensive experimental evaluation of Pensive, using different sets of video content and network traces. For lack of space, we only present the summary of our finding here, for that were not reported in the literature (for more details on other issues please refer to [5]).:

1) When the network trace used, during training, is highly variable, we found the Pensieve policy gradient suffers from high variance and failure to converge.
2) When training with a video data-set that contains a wide range of bitrates, for example, when using the UHD, $4K$, data-set presented in [6], we found the selection of the appropriate value of the rebuffing penalty significantly affecting the learning ability of Pensieve. In our evaluation, when we use 1 second of a rebuffing event to be equivalent to the lost of 1 second of the highest bitrate, as presented in the original paper, we found the RL agent learned to stream only at the lowest bitrate. And when the penalty is equal to the loss of 50% percentile bitrate, the trained model only learned to stream at the highest bitrate resulting in unacceptable level of video freezes.

TABLE I: Adaptation Algorithms used for data generation

Rate Base	Buffer Base	Hybrid
Minimum rate	BBA-2 [7]	Elastic [8]
Median rate	Logistic [9]	Arbiter+ [10]
Conventional [11]	BOLA [12]	Exoplayer Default [13]

The heterogeneity of wireless networks means that high variability will be common, and that increasing video resolutions are destined to become more popular. Hence motivating for this work.

III. Data Processing and Model Training

In this section, we discuss how the data used in training SMASH is generated, aggregated and pre-processed. To generate video adaptation logic for SMASH, we stream each of the ABR algorithms, shown in Table I, across the most widely used network technologies (3G, 4G and WiFi) and create a dataset of almost one million records from approximately 1,050 hours of real-time Dynamic Adaptive Streaming over HTTP *(DASH)*.

Fig. 1: The data generation testbed

A. Data Generation

Fig. 1 illustrates the testbed architecture used for all experiments. The testbed consists of one or more mobile devices, a wireless access point (WAP), and a server machine. The mobile devices used are Android-based smartphones. Phones are connected via WiFi to the WAP and use ExoPlayer [1] to stream the video content. ExoPlayer is a production-grade media platform for Android developed by Google. It supports both the DASH standard and HTTP Live Streaming *(HLS)*. The server runs an Apache web server on top of Ubuntu 16.04, equipped with 16GB of RAM and an Intel i7 CPU.

The link between the server and the WAP acts as the bottleneck. To emulate various wireless network characteristics, we use a Linux traffic control (tc) based traffic shaper for bandwidth emulation. Different wireless channels (3G, 4G and WiFi) are represented through collected network traces. The traces collected by Riiser et.al [14] and Raca et al. [15] contain bandwidth logs for 3G (mean=$1.26Mbps$, std=$0.97Mbps$) and 4G (mean=$11.32Mbps$, std=$13.17Mbps$) network technologies, respectively. For the WiFi (mean=$18.71Mbps$, std=$17.73Mbps$), we collected the WiFi traces using an

[1] https://github.com/google/ExoPlayer

TABLE II: Average Encoding Rate versus per Clip Resolution for the 4K clip Sintel [6]

	40Mbps	25Mbps	15Mbps	4.3Mbps	3.85Mbps	3Mbps	2.35Mbps	1.75Mbps	1.05Mbps	750kbps	560kbps	375kbps	235kbps
Sintel	3840x1744	3840x1744	3840x1744	1920x872	1920x872	1280x582	1280x582	720x328	640x292	512x234	512x234	384x174	320x146

TABLE III: Networks utilised during data generation

Technology	Number of Traces	Mode
3G [14]	30	Static, Pedestrian, Bus, Car, Tram, Train
4G [15]	25	Static, Pedestrian, Bus, Car, Train
WiFi	5	Pedestrian, Static

TABLE IV: List of features logged in the collected dataset

Features	**Description**
Chunk_Index	Streamed segment number
Arr_Time	Arrival time in milliseconds (ms)
Del_Time	Time taken to receive the segment (ms)
Stall_Dur	Stall duration (ms)
Rep_Level_i	The bitrate of the ith previously downloaded segments (kbps) ($i \in [1, ..., 5]$)
Avg_Rep_Level_5	Average bitrate of the 5 previous chunks (kbps)
Avg_Rep_Level_2	Average bitrate of the 2 previous chunks (kbps)
Del_Rate	Delivery rate (kbps) $\frac{Byte\_Size\ *\ 8\ bits}{Del\_Time}$
Act_Rate_i	The actual rate (kbps) of ith previous chunk ($i \in [1, ..., 4]$), $\frac{Byte\_Size\ *\ 8\ bits}{Seg\_Dur\ in\ seconds}$
Avg_Act_Rate_5	Average of 5 previous Actual rates (kbps)
Avg_Act_Rate_2	Average of 2 previous Actual rates (kbps)
Byte_Size	Byte size of this segment
Buffer	Buffer level (ms)
Codec	Video encoder
Height	Representation height in Pixels
Width	Representation width in Pixels
FPS	Frame rate of the streamed video
Chunk_Dur	Segment duration (ms).
ChunkStartTime	Video start time of the Segment (ms)
Avg_Thr	Average Throughput (kbps)
Play_Pos	current Playback position (ms)
Target_Rate	The bitrate of next chunk (kbps)

Android mobile network monitoring application called G-NetTrack Pro [2]. It is worth noting that all traces used were gathered on operational networks and thus the traces represent the bandwidth available to a client while, possibly, contending with other devices. From the pool of available network traces, we selected 60 randomised traces, see TABLE III for more detail. For each video streaming session, one or more mobile devices, a wireless access point (WAP), and a server machine are used.

Since it is not possible to include every ABR algorithm available, we carefully select and implement eight video quality adaptation algorithms in addition to the ExoPlayer's default adaptation algorithm (shown in Table I). The selected ABR algorithms cover a wide range of state-of-the-art approaches, and include *Rate based*, *buffer based* and *hybrid* algorithms. Except for ExoPlayer's default adaptation algorithm, we validated our implementations of these algorithms by comparing the results of the experiments we conducted, with the results from the published papers.

To stress test both the algorithms and the network traces, it was decided to stream content at a maximum resolution of 4K, (Ultra High Definition - UHD), which is currently gaining popularity. We utilised Sintel [16], which is a short animation clip of 14 minutes and 48 seconds duration, from the UHD dataset [6]. The clip is encoded using H.264/AVC, and contains eight resolutions across thirteen representation rates. Table II presents the detailed mapping of the video resolution to bitrate. Furthermore, the three most commonly used DASH segment sizes: 2, 4, 8 (seconds) were utilised from [6]. Each streaming session ran for five minutes and was repeated five times. In total, the video was streamed for a total of 63,750 minutes. Table IV presents the twenty-two features logged during each streaming session. All features are captured at the application layer. The logging module, we added to ExoPlayer, runs on a lightweight thread. Whenever new data is generated, the main thread raises an event that notifies the logging thread, which writes to a file in a non-blocking mode. In the current implementation, the main thread raises a notification after an ABR has decided on the bitrate of the video chunk to request next. Thus permitting us to capture all features, including the Target_Rate, and those features that a particular ABR might not support, e.g., throughput estimate and buffer level in the case of buffer-based and rate-based ABRs respectively. For both rate-based and hybrid ABRs, we use the same throughput estimation techniques as per the original cited paper for each algorithm. However, for buffer-based approaches, the averages of the previous two and five segments are recorded.

B. Data Preprocessing

Our generated data needs further processing to be in a form that is acceptable by most machine learning algorithms. In this section, we present the steps taken to pre-process the logged dataset. It should be noted that we did not encounter any missing value issues, because all experiments with missing values were repeated. First, we encoded the *Stall_Dur* with categorical values such that:

$$Stall\_Dur = \begin{cases} 1 & Stall\_Dur > 0 \\ 0 & Stall\_Dur = 0. \end{cases}$$

Furthermore, wherever the value of Stall_Dur is 1, the value of Target_Rate was replaced with the minimum video representation. With this, our aim is to help SMASH to learn to predict the lowest bitrate whenever a video stall event occurs, regardless of the length of the stall duration. Next, we encoded the following features: Rep_Level_i where $i > 1$, Chunk_Dur and Target_Rate into ordinal values. This ensures that during training, the ML algorithm is aware of the natural order that exists in their values. Finally, we normalised and scaled all the numerical features.

[2]http://www.gyokovsolutions.com/

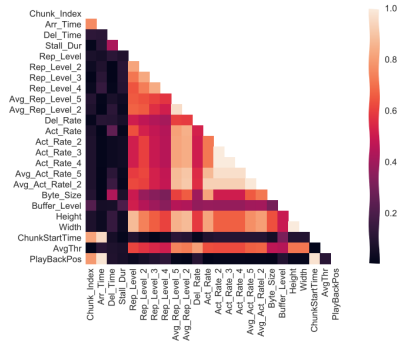


Fig. 2: Features correlation matrix

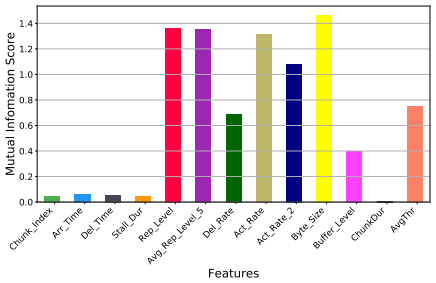


Fig. 3: Mutual Information score

C. Feature Selection

Existing ML-based video rate adaptation approaches rely on the same set of features as the fixed-rule based ABRs, such as throughput and buffer occupancy [2], [17]. To investigate the contribution of non-traditional parameters, as seen in the previous section, we extend the set of features we collect. We adopt this comprehensive approach to ensure that we reduce the chances of overlooking any potentially beneficial features. Next, we investigate how each of the selected features contribute to the quality of our model. Throughout the experimentation presented in this section, we use scikit-learn, a Python module for machine learning algorithms [18].

We start by investigating the relationship between the features logged during each streaming session, as shown in Table IV. It is known that features that are highly correlated contain similar information. In other words, it is redundant to keep all correlated features [19]. Figure 2 depicts the correlation matrix of the collected dataset. We use (correlation > 0.8) as a threshold value for high correlation. As can be seen, Rep_Level_2 , Rep_Level_3 and Rep_Level_4 are highly correlated with Rep_Level, hence all columns of Rep_Level_i, where $i > 1$ are dropped. Also we see, Avg_Rep_Level_2 is highly correlated with Avg_Rep_Level_5, so we drop the former. Other features that are dropped at this stage are: Avg_Act_Rate_2, Act_Rate_3, Act_Rate_4, Height, Width, ChunkStartTime, and PlayBackPos.

Next, we check how the remaining features help us improve the prediction for the next bitrate quality (representation level) to request from the video server (Target_Rate). We employed two widely used techniques to measure the dependency between the Target_Rate and each of the remaining features, namely: Chi-square (χ^2) statistic and Mutual information (MI). While the ranking for the features slightly varies across the two techniques, overall the result is similar. Therefore, we only present the result of the MI test (for more detail on Chi-square (χ^2) statistic refer to [20]). The MI score is a non-negative value that measures the dependency between two variables (X, Y), which quantifies the amount of information that can be obtained about Y, from variable X. MI is computed using the following formula:

$$MI(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \qquad (1)$$

where $p(x,y)$ is the joint probability density function of x and y, and $p(x)$ and $p(y)$ are the probabilities of x and y, respectively. For feature selection, the goal is to select features $K \subset X$, where $|K| = k$ such that

$$K = \underset{k}{\operatorname{argmax}}\, MI(X,Y).$$

We ran this ML test between each of the remaining features x and the target variable (Target_Rate). The result of the test is shown in Figure 3. As can be seen, the MI values of the Chunk_Index, Arr_Time and Chuck_Dur are very close to zero, hence are dropped as input to the training of SMASH. What we found counter-intuitive is that the MI value of the Stall_Dur is extremely low. We are surprised because Stall_Dur is the only feature that encodes stall events, so we expect it to be significant in predicting the Target_Rate. Therefore, we conducted further analysis of our training data-set. From the investigation, we found that less than 3% of our entire data-set records contains stall events. This happened because once the stall starts no content follows hence no data is logged, until it finishes. As such, at the data capturing stage, each stall duration is logged once regardless of the length of stall duration. And since the average stall duration is longer than the average chuck duration, the number of stall events, we recorded, is significantly reduced. As we have emphasised earlier in the previous section, when stall event occurs, the values of all features are irrelevant except that of Stall_Dur and Target_Rate. In light of this fact, we created a range of systemic records with values of Stall_Dur set to 1 and and Target_Rate set to the lowest representation. For the remaining features, we generated values from a normal distribution with the mean and standard deviation of the original columns, The augmented data is then added to the dataset, to double the level of stall instances in our training dataset to 6%.

D. Model Training

In this section, we evaluate eight well-known machine learning *(ML)* models and select the model with highest cross-validation accuracy. To select the best ML model to be used in training our prediction engine, we split our data-set into training and test sets at the ratio of $90\% : 10\%$ which is typical in ML modeling.

We use a *k-fold* cross-validation (CV) method to compare the accuracy of eight ML algorithms. Table V presents each of the ML algorithms and the corresponding hyper-parameters

TABLE V: ML models evaluated

ML Model	Scikitlearn Parameters
Logistic Regression (LR)	penalty=L2, solver=liblinear,
Quadratic Discriminant Analysis (QDA)	priors=None, reg_param=0.0, store_covariance=False, tol=0.0001
K-Nearest Neighbors (KNN)	n_neighbors=20
Decision Tree Classifier (DTC)	max_depth=None, criterion=gini
Gaussian Naive Bayes (NB)	priors=None, var_smoothing=1e-09
Ada Boost Classifier (ADO)	algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=100, random_state=0
Random Forest Classifier (RFC)	n_estimators=100
Multi-layered perceptron (NN)	solver=lbfgs, alpha=1e-5, hidden_layer= (10, 20, 10), random _state = 1

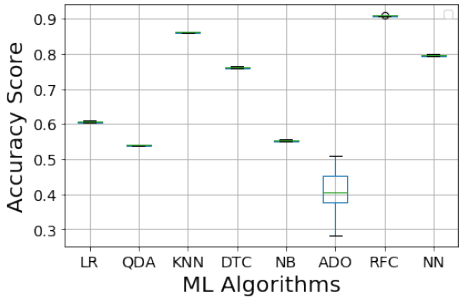


Fig. 4: Result of 10-fold cross-validation.

used. Each ML model is cross-validated using the training portion of the data-set. In a k-fold CV, the training set is divided into k equal-sized parts. First, a portion is used as a validation set and the remainder of the data is used as a training set. The accuracy a_i, which represents the correctly classified instances, is then recorded and the process continues for k times. The average accuracy (CV_a) is computed as

$$CV_a = \sum_{k=1}^{k} a_k. \tag{2}$$

Figure 4 illustrates the result of the our cross-validation, when $k = 10$ is used. As can be seen, ADO exhibited the highest variability in its accuracy score and QDA is the least accurate, with an accuracy score of 54%. RFC with a 91% accuracy has the highest score, hence was selected as the chosen ML model. Next, when we evaluated the performance of the selected model, *RFC*, on the test dataset, it recorded a test accuracy is 87.0%. From this, we conclude our trained model does not suffer from either underfitting or overfitting.

IV. Experimental Results

To evaluate the performance of SMASH, we deploy, the trained RFC model, to the ExoPlayer framework. For model training, we use a Java-based Weka framework [3]. The same parameter(s) as shown in Table V are used. For the performance evaluation of *SMASH*, the same testbed shown in Figure 1, is used. For comparison, we use: BBA2 [7] a buffer-based player, ARBITER+ [10] a hybrid player and Pensieve [2] an ML-based player. For Pensieve, we implemented its client interface in ExoPlayer, while we use the authors' original implementation of the server-side logic [4]. We retrained the Pensive model using a linear reward function, and using the same video content and network traces used in generating the SMASH training dataset.

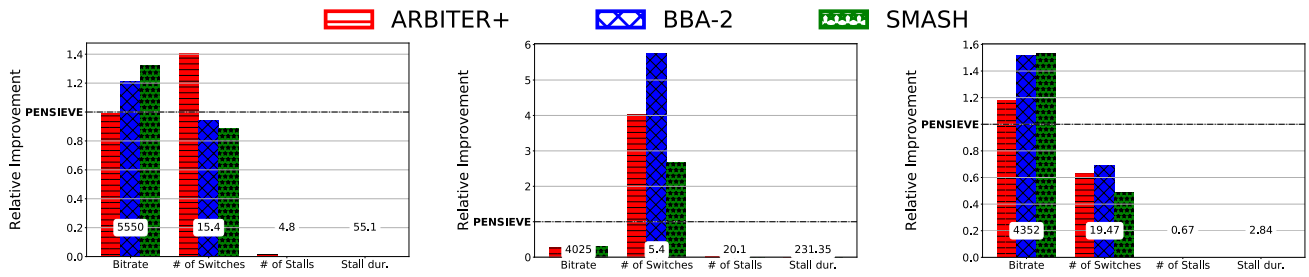
During the experimentation 3G (mean=$0.83Mbps$, std=$0.86Mbps$, 4G (mean=$18.15Mbps$, std=$20.97Mbps$ and WiFi (mean=$12.078Mbps$,std=$6.82Mbps$ networks were emulated. For each category of network channel, we use four traces, and each experiment is run five times, with the results showing the average over the runs. We used different network traces during our evaluation. Throughout the experimentation a DASH video chunk size of $4s$ is used. Figure 5 presents the result of our evaluation. All metrics are normalised to the Pensieve score.

Figure 5a shows the result of all players streaming over a 4G network. SMASH achieves the highest average bitrate and the lowest number of switches compared to other algorithms. In particular, SMASH improves bitrate quality by 38% compared to Pensieve while having 15% fewer bitrate switches. Furthermore, Pensieve suffers the highest number of video stall events, while remaining algorithms complete stall-free sessions. However, stall-free sessions come at the expense of the lowest average bitrate and the highest number of switches for the ARBITER+. BBA-2 falls into the middle between ARBITER+ and SMASH regarding bitrate and switching performances. Overall, SMASH achieves the best performance in a 4G environment.

When we stream video over a 3G network, a link that is known for low throughput and high capacity fluctuation, Pensieve is found to be very aggressive (see Figure 5b) and achieved an average bitrate of 3490kbps. Furthermore, Pensieve is very consistent in its choice of bitrate and is the least responsive to change in network capacity. Even when it switches its bitrate, we found it suffers from high amplitude variation, a technique known to have a detrimental impact on QoE. Consequently, it exhibited the lowest number of bitrate switches. However, this comes at the cost of a significant increase in both number of video stall events (average of 18 stalls per streaming session) and stall duration, staying on average 238 seconds in re-buffering state. This corresponds to approximately half the duration of the streaming session. The two baseline players (BBA2 and ARBITER+) record comparable bitrate but with a significant increase in bitrate switches. BBA2 recorded 252% more switches, while Arbiter suffered 153% more bitrate quality switches compared to to SMASH. Overall SMASH achieves the lowest number of switches, video stalls, and shortest stall duration while having comparable average bitrate. As a result, SMASH strikes the best balance across the QoE metrics and achieves the best performance in a 3G environment.

[3]https://www.cs.waikato.ac.nz/ml/weka/

[4]https://github.com/hongzimao/Pensieve



(a) 4G evaluation results (b) 3G evaluation results (c) WiFi evaluation results

Fig. 5: Relative improvement of different QoE metrics across different HAS algorithms (*The metrics are normalised to the results of the Pensieve algorithm with numbers in white boxes representing the metric value for the Pensieve algorithm*)

Figure 5c depicts the result of our evaluation of streaming video over a WiFi channel. Similar to the 4G case, Pensieve achieved the worst performance (lowest average bitrate, the highest number of stalls and longest stall duration), followed by ARBITER+. Though BBA-2 and SMASH enjoy a similar performance in terms of average video bitrate, BBA-2 recorded 40% more video switches, making SMASH the best-performing algorithm in a WiFi environment.

Furthermore, it is worth noting the performance of the baseline algorithms change as the underlying network condition varies, for example, BBA-2 performs better when network capacity is both abundant and stable, while ARBITER+ performed better in a resource-constrained environment. However, by learning from a variety of players, SMASH learns to customise its behaviour as streaming context changes, and provides the best results across all evaluated networks.

V. Conclusion

In this paper, we propose a **S**upervised **M**achine learning approach to **A**daptive **S**treaming over **HTTP** *(SMASH)*. SMASH is a supervised adaptive bitrate *(ABR)* predictive machine learning *(ML)* model which mimics the best behaviour of a variety of ABR algorithms to predict the optimal choice for the next video segment regardless of the complexity of the streaming context. The results of evaluating SMASH validate our hypothesis, that is, the best way to generalise the performance of an ABR algorithm is for a predictive model to learn the art of streaming from many high performing ABR schemes, i.e., our ML adaptation model makes the correct choice of segment quality irrespective of the transmission context.

Acknowledgements

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant 13/IA/1892, and also acknowledges the support of SFI Grant 13/RC/2077.

References

[1] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang, "Oboe: auto-tuning video ABR algorithms to network conditions," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*.

[2] H. Mao, R. Netravali, and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*.

[3] L. De Cicco, G. Cilli, and S. Mascolo, "Erudite: a deep neural network for optimal tuning of adaptive video streaming controllers," in *Proceedings of the 10th ACM Multimedia Systems Conference*.

[4] Y.-L. Chien, K. C.-J. Lin, and M.-S. Chen, "Machine learning based rate adaptation with elastic feature selection for HTTP-based streaming," in *2015 IEEE International Conference on Multimedia and Expo (ICME)*.

[5] P. G. Pereira, A. Schmidt, and T. Herfet, "Cross-Layer Effects on Training Neural Algorithms for Video Streaming," in *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2018, pp. 43–48.

[6] J. J. Quinlan and C. J. Sreenan, "Multi-profile Ultra High Definition (UHD) AVC and HEVC 4K DASH Datasets," in *Proceedings of the 9th ACM Multimedia Systems Conference*, ser. MMSys '18.

[7] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," in *2014 ACM Conference on SIGCOMM*.

[8] L. D. Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo, "ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH)," in *2013 20th International Packet Video Workshop*.

[9] Y. Sani, A. Mauthe, and C. Edwards, "Modelling Video Rate Evolution in Adaptive Bitrate Selection," in *2015 IEEE International Symposium on Multimedia (ISM)*, Dec 2015, pp. 89–94.

[10] A. H. Zahran, D. Raca, and C. J. Sreenan, "ARBITER+: Adaptive Rate-Based InTElligent HTTP StReaming Algorithm for Mobile Networks," *IEEE Transactions on Mobile Computing*.

[11] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale," *IEEE Journal on Selected Areas in Communications*.

[12] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptationf for online videos," in *IEEE INFOCOM 2016 - 35th Annual IEEE International Conference on Computer Communications*.

[13] Google. (2018, mar) An extensible media player for android. [Online]. Available: https://github.com/google/ExoPlayer

[14] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*.

[15] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics," in *Proceedings of the 9th ACM Multimedia Systems Conference*.

[16] B. Institute. (2010) Sintel - the durian open movie project. [Online]. Available: https://durian.blender.org/about/

[17] C. Sieber, K. Hagn, C. Moldovan, T. Hoßfeld, and W. Kellerer, "Towards Machine Learning-Based Optimal HAS," *arXiv preprint arXiv:1808.08065*, 2018.

[18] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python ," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[19] C. Albon, "Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning," 2018.

[20] A. Satorra and P. M. Bentler, "A scaled difference chi-square test statistic for moment structure analysis," *Psychometrika*.