PyDash – A Framework Based Educational Tool for Adaptive Streaming Video Algorithms Study

Marcelo A. Marotta, Gustavo C. Souza, Maristela Holanda, Marcos F. Caetano
Department of Computer Science
University of Brasília (UnB)
Brasília, Brazil
{marcelo.marotta|mholanda|mfcaetano}@*unb.br*, *costa.gustavo@aluno.unb.br*

***Abstract*—Full Paper in the Innovative Practice track – The pandemics caused by the spreading of the COVID 19 virus cornered the educational system worldwide, changing the classroom into remote class activities. This change in our social behavior has directly impacted the volume and shape of the Internet traffic data. A recent study shows 15% to 30% increases in Internet traffic caused, among other reasons, by educational video streaming traffic during few weeks in the 2020 lockdown period in Europe. To give some perspective, network providers usually work with a 30% data traffic increase per year. In 2021, it is expected that almost 82% of all Internet traffic will be video, according to CISCO annual forecast report. This scenario has a tremendous impact on the Internet bandwidth capacity, demanding optimized video streaming solutions, such as adaptive bitrate algorithms (ABR). On the other hand, considering the educational challenges in computer network courses, the core activities must be executed using specialized infrastructure to develop students' capabilities with networking equipment. As these types of equipment are costly to be obtained and forwarded to in-home students or simply e-students, a remote platform capable of reproducing an environment for networking applications is required. This is the scenario where PyDash was built. PyDash is a framework for the development of adaptive streaming video algorithms. It is a learning tool designed to abstract the networking communication details, allowing e-students to focus exclusively on developing and evaluating ABR protocols. This paper presents our practical experience developing and using PyDash as an educational tool for teaching ABR protocols at Computing Networking courses at the Department of Computer Science at the University of Brasilia, Brazil. Last semester, over 120 students, divided into four different undergraduate courses, had their first contact with PyDash. Even though this was their first contact with ABR concepts and the pyDash tool, they were able to perform the design, implementation, validation, and analysis of some state-of-the-art algorithms used by Netflix and Youtube.**

***Index Terms*—Educational Tool, Adaptive Streaming Video, Adaptive Bitrate Algorithms, Testbed, DASH, ABR**

I. Introduction

The pandemics caused by the spreading of the COVID 19 virus cornered the educational system worldwide, changing the classroom into remote class activities. This change in our social behavior has directly impacted the volume and shape of the Internet traffic data. A recent study shows 15% to 30% increases in Internet traffic caused, among other reasons, by educational video streaming traffic during few weeks in the 2020 lockdown period in Europe. To give some perspective, network providers usually work with a 30% data traffic increase per year. In 2021, it is expected that almost 82% of all Internet traffic will be video, according to CISCO annual forecast report [1]. This scenario has a tremendous impact on the Internet bandwidth capacity, demanding optimized video streaming solutions, such as Dynamic Adaptive Streaming over HTTP (DASH).

DASH is an adaptive video bitrate streaming technique that allows high-quality streaming of media content over the Internet, delivered from conventional HTTP web servers. DASH works by dividing the video content into a sequence of small segments, which are served over Hypertext Transfer Protocol (HTTP) [2]. Each segment contains a short amount of fixed video time, representing a scene from the original video. The union of all these segments can form the content of a film or a live broadcast event. According to the DASH protocol, content is made available at various bit rates and segment sizes. A movie is stored as a fixed time slot segment sequence (segment size) with differently encoded quality versions associated at the server-side.

While a DASH client is playing the content, it uses a bit rate adaptation (ABR) algorithm [3] to automatically select the next segment with the highest possible encoded bit rate that can be downloaded without causing crashes or re-buffering in playback. Therefore, a DASH client adapts itself to face the dynamic network fluctuation and provides high-quality reproduction, with few blocking or re-buffering events, given the reasoning within the ABR algorithm [2].

The ABR algorithm can be implemented considering many network metrics, such as throughput, jitter, network delays, and packet error rate, that, when combined with different techniques, such as machine learning and linear programming, create a rich educational opportunity for network students. However, to give this opportunity to the students during the pandemics is challenging. A DASH developing platform must be deployed over specialized infrastructure to develop students' capabilities with networking equipment. As these types of equipment are costly to be obtained and forwarded to in-home students or simply e-students, a remote platform capable of reproducing an environment for networking applications is required.

This is the scenario where PyDash was built. PyDash is a framework for the easy development of adaptive streaming video algorithms. It is a learning tool designed to abstract the networking communication details, allowing e-students to

focus exclusively on developing and evaluating ABR protocols. This paper presents our practical experience developing and using PyDash as an educational tool for teaching ABR protocols at the Computer Networking course, at the Department of Computer Science, at the University of Brasília (UnB), Brazil. Last semester, over 120 students, divided into four different undergraduate courses (i.e., computer engineering, mechatronics engineering, computer science, and degree in computing), with a different professor per class, had their first contact with PyDash. Even though this was their first contact with ABR concepts and the PyDash tool, they were able to perform the design, implementation, validation, and analysis of some state-of-the-art algorithms used by Netflix and Youtube.

This paper is structured as follows, Section II presents the related works regarding others DASH evaluation platforms. The PyDash architecture is discussed in Section III. Section IV will present how a new ABR algorithm is prototyped in the PyDash platform. The standard output log information provided is presented in Section V. The methodology and empirical results using PyDash as an educational platform are discussed in Section VI. Lastly, Section VII will present the conclusions and the future work for the evolution of the PyDash platform.

II. Related Work

One of the main purposes, of a DASH client evaluation platform is to design and evaluate ABR algorithms able to dynamically select a video bitrate that fits the available bandwidth, avoiding playback pauses and maximizing the users quality of experience metrics (QoE). On that matter, many tools were proposed considering such requirements. Some of them rely on decoding and displaying the video content, like dash.js [4], GPAC [5], and ExoPlayer [6]. Other ones are *emulation-based* players such as TAPAS [7], AStream [8], dashc [9], and goDash [10]. Lastly, there are *simulation-based* players such as Sabre [11] platform. *Simulation-based* player is a solution able to streaming video content without the need to decode it. A player with no need to decoding/display video content not only provides a low CPU and memory occupancy rate as allows a large number of opened parallel video streaming sessions. This has a better evaluation scenario scalability with a higher number of dash clients involved.

In [7] is proposed the TAPAS platform. Its main goal is focusing on ABR protocols development. The tool is written in Python language and composed of three main modules, which are: (1) the MPD parser module; (2) the Controller module; and (3) the Media Engine module. It doesn't implement a network bandwidth shaper module on the client-side. TAPAS is designed for specialist use without considering an educational purpose. It relies on advanced knowledge to install specifics libraries' dependencies that hinder its usage. The AStream tool is presented in [8] as a part of a Segment Aware Rate Adaptation (SARA) algorithm proposal. It has a modular architecture, using external GNU/Linux open software to implement traffic shaper features. It is not an educational tool and requires previous technical knowledge to be installed and operated.

In [9], the authors present DashC evaluation tool. The base idea is to provide a DASH player for extensive experiments running on a single machine. Working as an emulation-based player, DashC uses lower memory and CPU allocations than TAPAS solution. Like the others solutions, DashC also doesn't have an educational purpose. Similarly, goDASH [10] is a modular, dynamic, and lightweight DASH compatible video streaming tool, that comes with an easily modifiable JSON configuration file. The goDASH tool is written in golang, and it is comparable with DashC. Both offer a means of running multiple parallels DASH clients instances. On the other hand, goDASH provides a framework for fast coding, implemented QoE models, and transport protocols. However, goDASH is not an educational tool and requires previous knowledge to install and operates it. Looking for general proposing networking simulation tools, we can mention NS3 [12] and OMNeT++ [13] simulators. Both tools are implemented in C++ programming language and they were designed for networking protocol evaluation and optimization. From the educational point of view, these simulation tools have a very high learning curve, which turns difficult to use in an undergraduate course.

In summary, current DASH clients systems are particularly effective in providing a lightweight, flexible, scalable, and ease of use tool. In contrast, compared to PyDash, these platforms do not have educational purposes, hindering their usage for a networking class environment. They also need previous knowledge to be installed and configured. Some of these requirements are advanced, such as virtualization, packet filtering, and network configuration which are not necessarily common knowledge for e-students.

III. Architecture

The *PyDash* platform was designed to allow the implementation and evaluation of ABR algorithms for educational purposes. Before we approach how the PyDash implementation is done, let us first analyze the main components that are part of its architecture.

A. DASH client

Inheriting the Dash architecture [2], Figure 1 shows the relationship of the class *DashClient* and its main components, as well as its interaction with an HTTP server, which stores and publishes videos following the MPEG-DASH format. The class *DashClient* is formed by three main attributes, which are organized in a stack structure, such as described next:

- ***Player*** – The *Player* class implements the behavior of the video player. It implements the *buffer* structure, which is used to control which segments need to be retrieved from the HTTP server and which have already been stored. Of the stored segments, the *Player* labels which ones have been played by the user from those not yet consumed. In addition, the *Player* monitors several performance metrics, such as the number of video pauses, how long

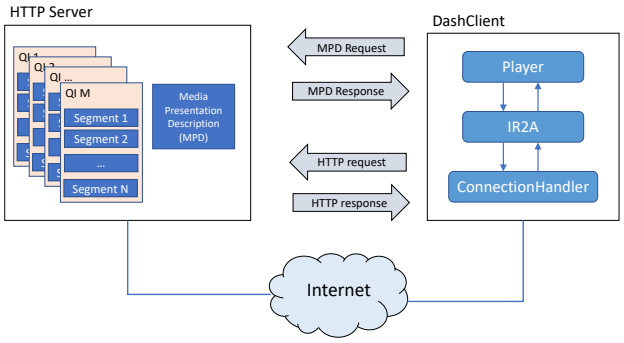

Fig. 1. DashClient Solution Architecture.

were these pauses, the selected qualities, among others. The *Player* also generates the statistics of the monitored metrics, which e-students can use to monitor the behavior of the *buffer* during the video reproduction. The *Player* is responsible for defining which will be the next segment to be retrieved from the HTTP server. It sets up this request in a *base.SSMessage* message. Once assembled, the message of type *base.SSMessage* is forwarded to the layer below (*IR2A*).

- ***IR2A*** – The *Interface of Rate Adaptation Algorithms* (IR2A) is an Abstract Class that defines a common interface to be implemented by the new ABR algorithms. The interfaces define abstract methods, which must be implemented by the ABR algorithm to be validated. A valid ABR algorithm developed by a e-student can be loaded automatically in the *PyDash* evaluation platform. After being loaded, the ABR algorithm will be located between the classes *Player* and *ConnectionHandler*, intermediating the messages that go down (*resquest*) and up (*response*) in the *DashClient* stack. From the class *Player*, the e-students ABR algorithm receives the message of type *base.SSMessage* and, based on its logic and selection policies, defines which quality should be used when requesting a segment by creating a message. The algorithm forwards this message to the bottom layer (*base.SSMessage*). The work to be developed will concentrate exclusively on the ABR algorithm that should implement the *IR2A*.
- ***ConnectionHandler*** – class responsible for the communication between the platform *PyDash* and the HTTP server, as defined in the configuration file (*e.g., dash_client.json*). Converts a request defined by a message of type *base.SSMessage* into a functional HTTP connection and vice versa. Upon receiving a message of type *base.SSMessage*, coming from the upper layer, it makes a connection to the HTTP server and forward its response to the upper layer, reusing the same format in the process. This class also implements the *traffic shaping* algorithm for bandwidth control. This mechanism permits the test of ABR e-student's implementation.

The current PyDash architecture design allows the e-students effort be focused only in *IR2A* layer. It is not necessary for them to have knowledge of *Player* and *ConnectionHandler* layers internal operation. In order to incorporate a new ABR protocol into PyDash, the e-students should inherit *IR2A* class and place their python file in **r2a** directory. This approach permit the e-students being focus in their solutions instead in details of network communication, video reproduction and statistics.

We stimulate e-students to send requests for bug fixing or for new features, on the PyDash platform, throughout a versioning service such as GitHub or GitLab. Using a versioning platform, we allow e-students to get familiarized and understand how software is created and sustained by the computer science community. In this case, e-students are advised and instigated to submit a formal software update request. We also allow their request to become a ticket to be studied that can be later either attended or rejected, emulating how most of the software development process is performed. Finally, attended requests become updates to the original source code that will be made available in the versioning repository.

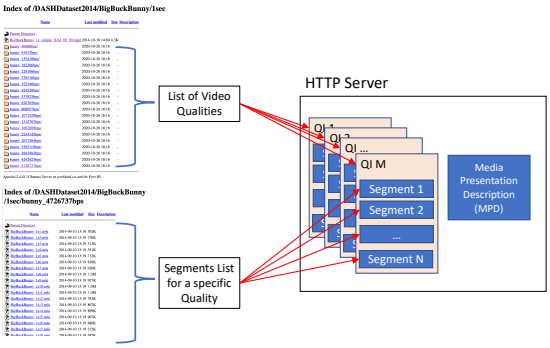

Fig. 2. Architecture Details *DashClient* Server side

In MPEG-DASH format, videos are stored and organized following a specific way. Figure 2 presents one example of a stored video following MPEG-DASH format. The video is segmented into different fixed sizes. Considering 1-second segment size, the video has 20 different qualities in which it was encoded, with the lowest quality being $46,980$ bps and the highest $4,726,737$ bps.

IV. Prototyping a New ABR Algorithm

As described in the previous section, the architecture of the *PyDash* platform is organized in three layers. In the middle layer, there is the abstract class *IR2A*. This class plays an important role for the e-students, since all new ABR algorithms must inherit this class to be incorporated into the platform. It means that the *IR2A* serves as super class for the e-students to develop their own *ABR* algorithms.

Figure 3 shows the relationship of how a e-student's ABR implementation can be dynamically coupled to the *PyDash* platform using the *IR2A*. By implementing all the abstract methods, defined by the class *IR2A*, the platform

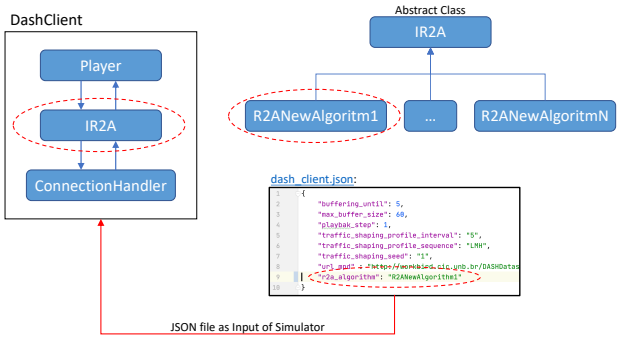

Fig. 3. Architecture Details *PyDash* Client side

dynamically loads the e-student *ABR* protocol, obtaining its name from the **r2a_algorithm** field present in the file *dash_client.json*. In the figure, as an example, the protocol is called *R2ANewAlgorithm1*. Algorithm 1 presents the abstract methods of the class *IR2A*, to be implemented by the e-students' new ABR algorithm. This new policy must be implemented in a class that will inherit the attributes of *IR2A*.

Algorithm 1 Abstract methods of the class *IR2A*.

```
class IR2A (SimpleModule):
  ...

  @abstractmethod
  def handle_xml_request(self, msg):
      pass

  @abstractmethod
  def handle_xml_response(self, msg):
      pass

  @abstractmethod
  def handle_segment_size_request(self,msg):
      pass

  @abstractmethod
  def handle_segment_size_response(self,msg):
      pass

  @abstractmethod
  def initialize(self):
      pass

  @abstractmethod
  def finalization(self):
      pass
```

Lastly, we release in PyDash repository, three ABR protocols to work as coding examples of a fully functional working ABR protocols. The first one implements a fixed approach, always aiming for the same video quality. It doesn't consider the buffer occupancy rate or the bandwidth fluctuation as an input parameter. The second one has a random approach. For every request it will roll a dice to decide which video quality have to choose. The last one records the mesuared throughput history and uses this information to guess its next video quality. Those strategies are very simple ones, however it makes a good example of how the e-students should implement their own ABR protocol in the PyDash platform.

V. The PyDash's Standard Output

The output log information dynamically provided by PyDash is an important tool for the e-students to understand the behavior of their implemented ABR protocol. In its current release, PyDash provides a text-based output as a standard user interface. Figure 4 presents a screenshot of the standard console terminal during PyDash running. Using the terminal, the e-students are able to follow some running statistics such as segments online requests, buffer size evolution, playback pauses, among others. In the end, PyDash will provide more information such as the number of pauses experienced, the average QI, and the average QI distance. For the last two, the values will include also their standard deviation values and the variance.

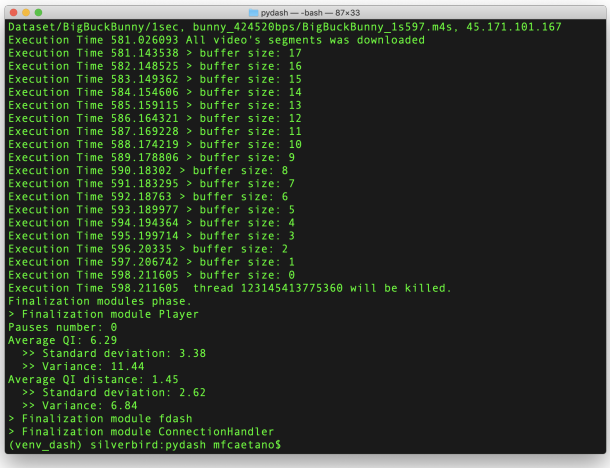

Fig. 4. An example of output from PyDash terminal execution.

PyDash will also generate some standard graphs about its last run. Here, we are going to present the standard graphs generated from an example of the execution of a student's group implementation of the FDASH [14] protocol. Figure 5 shows the playback history during the FDASH protocol running. As can be seen, for the set evaluation scenario the user didn't experience any pauses. The playback status is defined as 1 during all execution time.

The non-pauses experience is explained by the buffer size behavior shown in Figure 6. Since the beginning of the execution time, the buffer size starts to fast increase and stays up to near 15 seconds size. This happened because the FDASH implementation was able to correctly predict which segment quality fitted better for the available networking throughput.

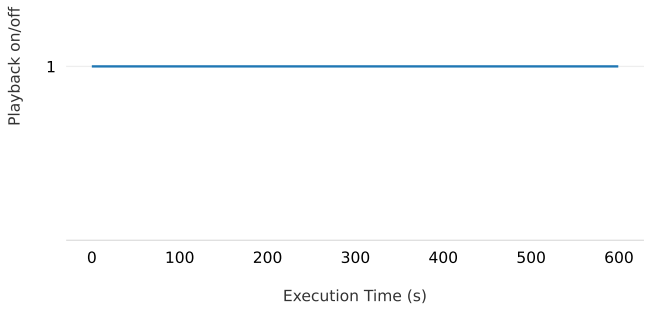

Fig. 5. The Playback history metric running [14] algorithm.

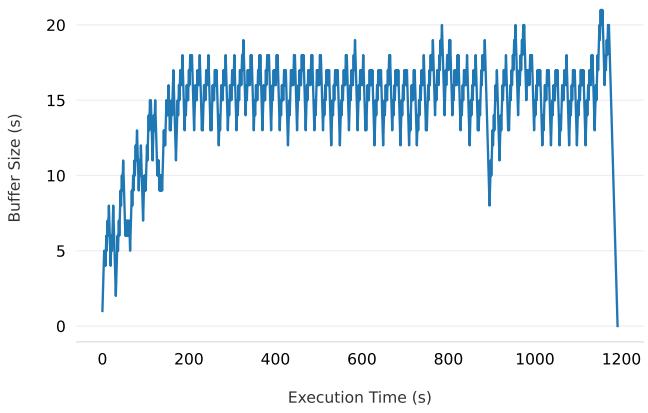

Fig. 6. The Playback Buffer Size metric running [14] algorithm.

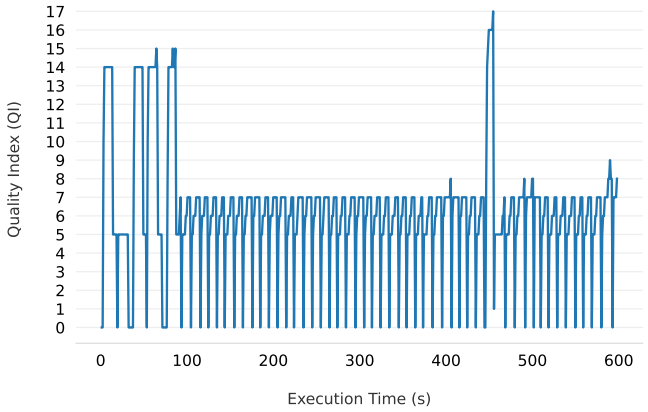

Fig. 7. The Playback QI metric running [14] algorithm.

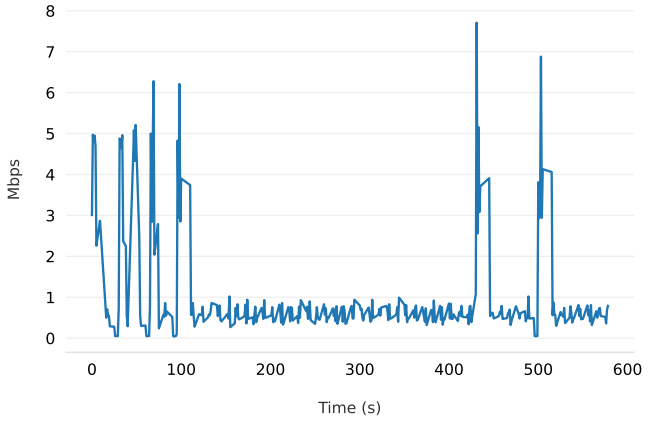

Fig. 8. The Throughput metric running [14] algorithm.

Figure 7 presents the history of the selected quality indexes (QI) for the downloaded video segments. Despite the ups and downs, the average selected quality was near seven, which is a fair value. However, it is possible to see the protocol behavior trying to adjust the best QI for each video segment during execution. Comparing to the results from Figure 8, it is possible to see the correlation between the throughput availability and the selected segment quality. Near to the throughput spikes, Figure 7 shows an increase in the QI values. It demonstrates the adaptive working behavior expected from the implemented ABR protocol. Making it easier for the e-students to see and understand the ABR concepts they are trying to implement in the PyDash tool.

VI. Methodology and Empirical Results

One of the primary purposes of the PyDash framework is to release a flexible educational tool to allow e-students to implement, test, and evaluate state-of-the-art ABR protocols quickly. In order to do an evaluation of the PyDash tool, from the educational point of view, 120 undergraduate e-students were asked to use it, during a final assignment, for a specific computer networking course.

As the proposal was the implementation of state-of-the-art ABR protocols, the e-students were gathered into groups of three to support each other during the process. For each group, a list of ABR papers was shared, which are presented in Table I. The Table I also presents the classification of the protocols in terms of complexity (*very simple*, *simple*, *complex*, and *high complexity*), publication year, chosen approach and implemented technique. We proposed the complexity classification of the papers based on a threshold between the chosen approach and the technique used. The e-students were also stimulated to propose their own ABR protocol solutions after implementing an example from literature.

Beyond the project specification, which contains elaborated documentation of PyDash architecture and the road map for a new-ABR protocol implementation, the e-students had access to recorded explanation videos. The first one presents the concepts of DASH protocols technology [25], based on a computer networking course textbook [26]. The second one has programming content [27], teaching how to code a new ABR protocol using the PyDash platform. They also had access to three straightforward ABR protocols source-code.

TABLE I
ABR PROTOCOLS SUGGESTED AND/OR IMPLEMENTED BY THE E-STUDENTS IN PYDASH FRAMEWORK.

Papers	Year	Acronyms	Approach	Technique	Solution Complexity
[15]	2012	THANG	Bandwidth-based	Normalization and Probability	Simple
[16]	2020	BOLA	Buffer-based	Near-Optimal Mathematical Solution	High Complexity
[17]	2013	QL	Buffer-based	Q-Learning Based	Complex
[18]	2017	DSSS	Bandwidth-based	Probability	Complex
[14]	2016	FDASH	Buffer-based	Fuzzy Logic	Complex
[19]	2020	KNN-Q	Bandwidth and Buffer Based	KNN with Q-Learning Algorithm	High Complexity
[20]	2014	PANDA	Bandwidth-based	TCP's Congestion Control Based	Simple
[21]	2019	RST	Bandwidth and Buffer Based	Normalization Driven	Simple
[22]	2011	LIU	Bandwidth-based	Moving Average	Simple
[23]	2012	QDASH	Bandwidth-based	Probing Approach	Simple
[24]	2015	SARA	Bandwidth and Buffer Based	Weighted Harmonic Mean	Simple
Student's Proposal	2020	FDASH-DF	Buffer-based	FDASH using a Digital Filtering	Complex
Student's Proposal	2020	AVG	Bandwidth-based	Simple Average	Very Simple
Student's Proposal	2020	AVG_BF	Bandwidth and Buffer Based	AVG with Buffer Size Threshold	Very Simple
Student's Proposal	2020	AVG_HM	Bandwidth-based	Harmonic Mean	Very Simple
Student's Proposal	2020	AVG_GM	Bandwidth-based	Geometric Mean	Very Simple

These starting points simple protocols enable the possibility for the e-students to see how an ABR protocol interacts with PyDash, having access to a real working source code implementation at the same time. As a result of their work, the e-students had to submit a final text report about their solution, attaching the developed source code. They also made fifteen minutes presentation, explaining their approach (protocol and source code), results, challenges, and difficulties faced during their working process. From here, we will present and discuss the main results achieved by the e-students using the PyDash educational tool. It is worth mentioning that personal information from e-students was anonymized, meeting security and privacy requirements.

As our first result, Figure 9 presents the final grade range distribution obtained by the e-students for this assignment. In our university, a student will pass if his/her score is greater or equal to 5.0. The majority of the e-students were able to obtain high scores. A high score level is considered for a score greater or equal to 7.0. Approximately 77% of the e-students got high scores. Figure 10 presents the final evaluation outcome of the e-students. For this assignment, 89% of the e-students passed, and only 5% have failed. We had 6% of the e-students dropped the assignment. For this group, an individual interview had shown personal problems related to the pandemic's situation. Since the vast majority of e-students succeed in creating their experiment from their own homes, we consider it as an important indicator to characterize PyDash as an appropriate platform for remote classes.

Figure 11 shows the complexity distribution of the submitted solutions regarding the kind of ABR protocol coded by the e-students. As expected, 71% choose to code ABR protocols classified as *very simple* or *simple*. However, 29% decided to code those classified as *complex* or *high complexity* protocols. This is a very good result, especially considering they are undergraduate e-students in the middle of their course. Also, this networking computer course is their first contact with networking programming concepts.

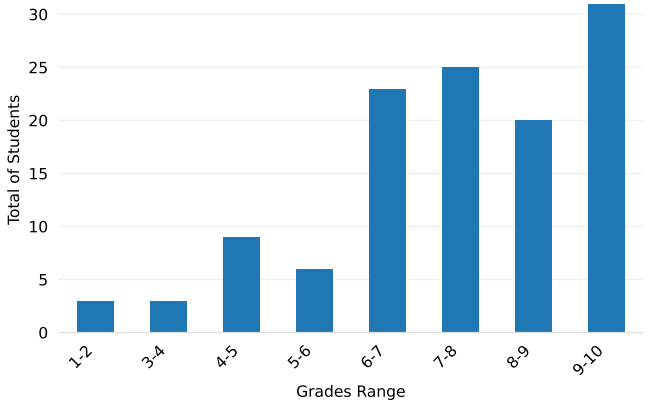

Fig. 9. Grades range distribution of the e-students.

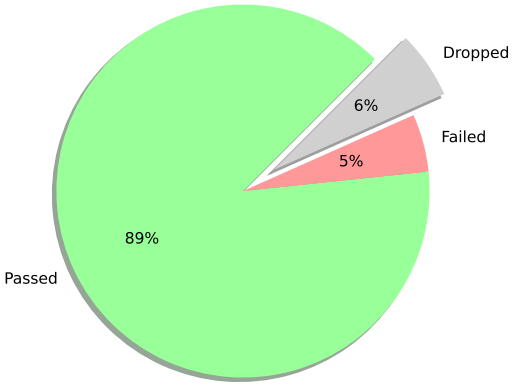

Fig. 10. Final evaluation outcome of the e-students.

To better understand which protocols were implemented, Figure 12 shows the discriminated complexity distribution. Almost 40% of the submitted solutions has coded PANDA [20]

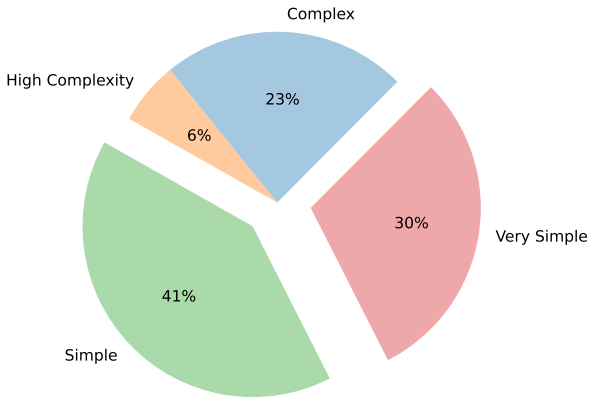

Fig. 11. Protocols complexity distribution implemented by the e-students.

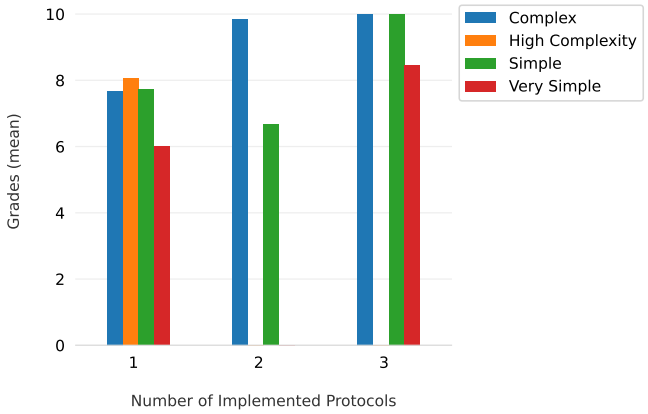

Fig. 13. Average grades computed by the number of protocols implemented by each student group.

and AVG protocols, which have *simple* and *very simple* complexity classification, respectively. On the other hand, over 25% has coded BOLA [16] and FDASH [14] protocols. These are state-of-the-art ABR protocols, requiring solid mathematical background skills combined with networking aspects to understand and implement the solution.

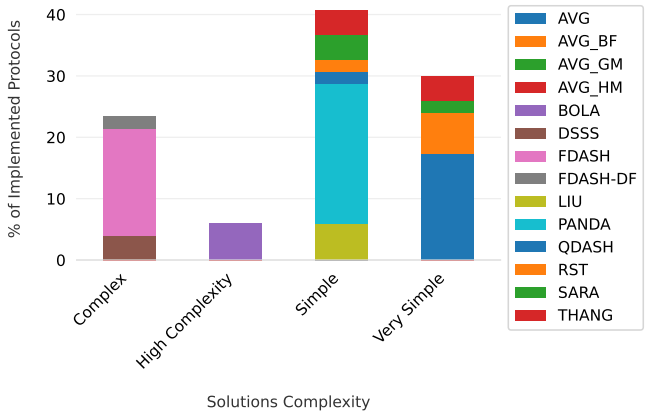

Fig. 12. Discriminated protocols complexity distribution implemented by the e-students.

Lastly, Figure 13 shows the correlation between average e-students' grades by the number of implemented protocols (NIP) and their complexity classification. The e-students were demanded to implement just one ABR protocol. However, some decided to implement and submit more than one solution. It is possible to see that the groups that have implemented two or three solutions had better grades. The only exception was for a group that decided to implement only simple solutions, and the NIP is equal to two. When the number of the implemented protocol is three, the grades are greater than 8.0

VII. Conclusion

This work proposes the PyDash educational framework tool for adaptive streaming video algorithm study. PyDash is a learning tool designed to abstract the networking communication details, allowing e-students to focus exclusively on developing and evaluating ABR protocols. This paper presents our practical experience developing and using PyDash as an educational tool for teaching ABR protocols at Computing Networking courses at the Department of Computer Science at the University of Brasilia, Brazil. Last semester, over 120 students divided into four different undergraduate courses had their first contact with the PyDash platform. The success rate of the students was 89% and 29% of students have coded ABR protocols classified as *complex* or *high complexity* protocols. These are state-of-the-art ABR protocols, requiring solid mathematical background skills to understand and implement the solution. It is a very good result, especially considering they are undergraduate students in the middle of their course. Also, this networking computer course is their first contact with networking programming concepts.

The PyDash is an under developing educational tool, and new features are scheduled to be released in the near future. We plan to incorporate 5G communications throughput tracing into PyDash to have more realistic testing scenarios. Also, we are planning to have a graphic interface, making it possible for the students to reproduce/watching videos, seeing the impact on the quality reproduction during the ABR protocol segment selection's decisions.

Acknowledgements

This work received video hosting support from GigaCandanga (http://gigacandanga.net.br).

References

[1] U. Cisco, "White paper: Cisco annual internet report (20182023)," p. 35, 2020, accessed on 04/23/2021.
[2] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, 2011.
[3] D. I. F. (DASH-IF), "ABR Logic," https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic, 2018, accessed on 04/23/2021.
[4] "dash.js," https://github.com/Dash-Industry-Forum/dash.js, accessed on 04/28/2021.

[5] "Gpac," https://gpac.wp.imt.fr/, accessed on 04/28/2021.

[6] "Exoplayer," https://github.com/google/ExoPlayer, accessed on 04/28/2021.

[7] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo, "Tapas: A tool for rapid prototyping of adaptive streaming algorithms," in *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, ser. VideoNext '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 16. [Online]. Available: https://doi-org.ez54.periodicos.capes.gov.br/10.1145/2676652.2676654

[8] P. Juluri, V. Tamarapalli, and D. Medhi, "Sara: Segment aware rate adaptation algorithm for dynamic adaptive streaming over http," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015, pp. 1765–1770.

[9] A. Reviakin, A. H. Zahran, and C. J. Sreenan, "Dashc: A highly scalable client emulator for dash video," in *Proceedings of the 9th ACM Multimedia Systems Conference*, ser. MMSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 409414. [Online]. Available: https://doi.org/10.1145/3204949.3208135

[10] D. Raca, M. Manifacier, and J. J. Quinlan, "godash go accelerated has framework for rapid prototyping," in *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, 2020, pp. 1–4.

[11] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: Improving bitrate adaptation in the dash reference player," vol. 15, no. 2s, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3336497

[12] N. Team, "Ns-3 network simulator," https://www.nsnam.org/, accessed on 06/28/2021.

[13] O. Team, "Omnet discrete event simulator," https://omnetpp.org/, accessed on 06/28/2021.

[14] D. J. Vergados, A. Michalas, A. Sgora, D. D. Vergados, and P. Chatzimisios, "Fdash: A fuzzy-based mpeg/dash adaptation algorithm," *IEEE Systems Journal*, vol. 10, no. 2, pp. 859–868, 2016.

[15] T. C. Thang, Q.-D. Ho, J. W. Kang, and A. T. Pham, "Adaptive streaming of audiovisual content using mpeg dash," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 78–85, 2012.

[16] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[17] M. Claeys, S. Latr, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck, "Design of a q-learning-based client quality selection algorithm for http adaptive video streaming," in *Adaptive and Learning Agents Workshop, part of AAMAS2013, Proceedings*, 2013, pp. 30–37.

[18] L. Bedogni, M. Di Felice, and L. Bononi, "Dynamic segment size selection in http based adaptive video streaming," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2017, pp. 665–670.

[19] H. Lin, Z. Shen, H. Zhou, X. Liu, L. Zhang, G. Xiao, and Z. Cheng, "Knn-q learning algorithm of bitrate adaptation for video streaming over http," in *2020 Information Communication Technologies Conference (ICTC)*, 2020, pp. 302–306.

[20] 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*, vol. 32, no. 4, pp. 719–733, 2014.

[21] P. C. L. Neto, A. Clio V. N., and D. C. Muchaluat-Saade, "The relative smoothed throughput approach for adaptive http streaming," in *2019 IEEE Symposium on Computers and Communications (ISCC)*, 2019, pp. 1–6.

[22] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive http streaming," ser. MMSys '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 169174. [Online]. Available: https://doi.org/10.1145/1943552.1943575

[23] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang, "Qdash: A qoe-aware dash system," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1122. [Online]. Available: https://doi.org/10.1145/2155555.2155558

[24] P. Juluri, V. Tamarapalli, and D. Medhi, "Sara: Segment aware rate adaptation algorithm for dynamic adaptive streaming over http," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015, pp. 1765–1770.

[25] M. F. Caetano, "Computer networking: Application layer – dash cdn / section 2.6 (portuguese video)," https://youtu.be/Tzm9uDi8ZG0, accessed on 05/15/2021.

[26] J. Kurose and K. Ross, *Computer Networking: A Top-down Approach, 8^{th} Edition*. Pearson, 2021. [Online]. Available: https://gaia.cs.umass.edu/kurose_ross/eighth.htm

[27] M. F. Caetano, "Computer networking: Programming project – pydash library (portuguese video)," https://youtu.be/PEdK-9W2uTI, accessed on 05/15/2021.