goDASH - GO accelerated HAS framework for rapid prototyping

Darijo Raca
Faculty of Electrical Engineering
University of Sarajevo, Sarajevo, BiH
draca@etf.unsa.ba

Maëlle Manifacier
Université Clermont Auvergne
Aubière Cedex, France
maelle.manifacier@etu.uca.fr

Jason J. Quinlan
Department of Computer Science
University College Cork, Ireland
j.quinlan@cs.ucc.ie

***Abstract*—In this short paper, we present *goDASH*, an infrastructure for headless streaming of HTTP adaptive streaming (HAS) video content, implemented in the language golang, an open-source programming language supported by Google. goDASH's main functionality is the ability to stream HAS content without decoding actual video (headless player). This results in low memory requirements and the ability to run multiple players in a large-scale-based evaluation setup. goDASH comes complete with numerous state-of-the-art HAS algorithms, and is fully written in the Google golang language, which simplifies the implementation of new adaptation algorithms and functions. goDASH supports two transportation protocols Transmission Control Protocol (TCP) and Quick UDP Internet Connections (QUIC). The QUIC protocol is a relatively new protocol with the promise of performance improvement over the widely used TCP.**

We believe that goDASH is the first emulation-based HAS player that supports QUIC. The main limitation in using QUIC protocol is the need for a security certificate setup on both ends (client and server) as QUIC demands an encrypted connection. This limitation is eased by providing our own testbed framework, known as *goDASHbed*. This framework uses a virtual environment to serve video content locally (which allows setting security certificates) through the Mininet virtual emulation tool. As part of Mininet, goDASH can be used in conjunction with other traffic generators.

***Index Terms*—Golang, HAS, HTTP Adaptive Streaming, QUIC**

I. Introduction

Globally, video traffic dominates today's Internet. Over 60% of traffic carried over the Internet belongs to video streaming (Sandvine 2019). Service providers like Netflix, Hulu, and Amazon offer subscription-based Video on Demand (VoD) services. Adaptive video streaming is the streaming technique at the centre of video content delivery. This method permits streaming video content on the end-devices while allowing seamlessly content quality adaptation to the network conditions and device capabilities. This method complies with the Internet's best effort policy.

HAS splits video content into multiple fixed-duration chunks (typically 2-10 seconds). Each chunk is encoded into multiple qualities (e.g., 200, 500, and 1000kbps). At the client side, the player maintains a playback buffer for storing downloaded chunks for decoding. All the intelligence is typically at the client side. The player estimates the available network resources and adapts by requesting the maximum chunk quality that minimises stalling events. The player can also take device capabilities in to consideration when selecting chunk quality. For example, if the device's screen supports only High Definition (HD) resolution, then the player can omit all the qualities encoded in higher resolutions.

In the literature, many HAS adaptation algorithms have been developed over the years [1]. Typically, these algorithms can be grouped into three categories: rate-, buffer- and hybrid-based. The rate-based algorithms solely rely on the measurement of the available throughput for chunk quality decision [2]. In a similar vein, buffer-based algorithms monitor playback buffer levels and maps them to chunk quality [3]. Finally, most state-of-the-art algorithms use a hybrid approach, taking both rate and buffer levels into the decision [4]–[7]. Further classification can be made based on approaches employed by the algorithm's adaption logic, ranging from optimisation [8], [9], game theory [1], control theory [6], prediction [10], machine learning [11], improving bandwidth estimation and other heuristics.

Several HAS players exist in the literature. The most well-known players include dash.js [12], GPAC [13] and ExoPlayer [14]. These players are most commonly used for experimentation of new HAS algorithms. However, the main drawback of these players is the inability to stream video content without decoding, making them unsuitable for large-scale evaluation. Alternatively, emulation-based players such as TAPAS [15], AStream [16], and dashc [17] allow streaming video content without decoding. However, compared to goDASH, these players have limited functionalities in terms of the number of implemented HAS algorithms, support for different manifest files, support for the QUIC protocol, and ease of implementing new functionalities.

In this short paper, we present *goDASH*, a video player for headless streaming of HAS content, implemented in the language golang [18], an open-source programming language supported by Google. goDASH provides options for:

- adaptation algorithms such as - Hybrid: Arbiter+ [7] and Elastic [6], Buffer Based: Logistic [3] and BBA [4], and Rate Based: Conventional [19], Progressive, Average, Geometric and Exponential.
- video codec: h264, h265, VP9 and AV1
- DASH profiles: full, main, live, full_byte_range and main_byte_range
- config file input
- video stream debug option for printing information

- MPD url header information extracting for all segments
- defining the maximum stream buffer in seconds
- defining the initial number of segments to download before stream starts (start up phase)
- defining a maximum height resolution to stream
- printing log output to file/terminal columns based on selected print headers
- video streaming using the TCP/QUIC transport protocol
- defining a folder location to store the streamed DASH files
- comes complete with its own testbed framework, known as *goDASHbed*. This framework uses a virtual environment to serve video content locally (which allows setting security certificates) through the Mininet virtual emulation tool, including set up of https certs

The remainder of the paper consists of: Section II, where we introduce goDASH in more detail. Section III provides an overview of the configuration setup for goDASH and provides an example goDASH stream output log. Section IV presents our future work in this area, and concludes the paper.

II. goDASH Overview

In this section, we introduce goDASH, a modular, dynamic and easily configurable headless DASH player. goDASH is written in golang, an open-source programming language supported by Google. goDASH enables numerous researcher options beyond the traditional comparison of multiple HAS algorithms. goDASH supports two transportation protocols TCP and QUIC. QUIC protocol is a relatively new protocol with a promise of performance improvement over widely used TCP. goDASH is the first tool that enables an exhaustive evaluation of HAS with the QUIC protocol. For performance comparison, goDASH calculates objective quality metrics (e.g., average quality, stall and switching performance). Many algorithms rely on information about future segment sizes. Typically, in the research environment, this information is stored in a separate file. However, in practice, this information may not be readily available. goDASH overcomes this problem by supporting two modes offline and online. Offline mode reads segment sizes from the file, while online mode requests segment sizes directly from the server (through sending HTTP HEAD requests). This functionality makes goDASH unique compared to other HAS players and moves it closer to a realistic environment.

goDASH also comes equipped with it's own testbed framework, known as *goDASHbed* (https://github.com/uccmisl/goDASHbed.git). This framework utilises Mininet for network emulation and permits streaming of multiple clients per session, each with their own debug/log output files. VOIP background traffic is provided through the Distributed Internet Traffic Generator (*D-ITG*) [20]. Variance in network throughput is provided through trace based datasets containing cellular key performance indicators (KPIs) - options for 3G, 4G and 5G exist.

We include one trace from two existing cellular trace datasets (4G [21] and 5G [22]) in goDASHbed, for ease of use. goDASHbed also includes options for DASH streaming using the TCP (HTTP/HTTPS) and QUIC (HTTPS) transport protocols. Integration of goDASH with a Mininet emulation environment opens new research areas and impact analysis of HAS traffic on other traffic types (i.e., web and VoIP) and vice versa. Little work has been done in this area [23]–[25].

TABLE I: Notation used in the goDASH Trace Output logs

Type	Description
Default Output:	
Seg_#	Streamed segment number (full, main, live) & segment range (full_byte_range, main_byte_range)
Arr_Time	Arrival time in milliseconds (ms)
Del_Time	Time taken to receive the segment (ms)
Stall_Dur	Stall duration (ms)
Rep_Level	Representation Quality (kbps)
Del_Rate	Delivery rate (kbps) $\frac{Byte\_Size * 8\ bits}{Del\_Time}$
Act_Rate	Actual rate (kbps) $\frac{Byte\_Size * 8\ bits}{Seg\_Dur\ in\ seconds}$
Byte_Size	Byte size of this segment
Buffer_Level	Buffer level (ms)
Optional Output:	
Algorithm	Adaptive Algorithm
Seg_Dur	Segment duration (ms)
Codec	Video encoder
Width	Representation width in pixels
Height	Representation height in pixels
FPS	Frame rate of the streamed video
Play_Pos	Current Playback position (ms)
RTT	Packet level (ms) - determined using HTTP head request
Protocol	HTTP protocol

goDASH has an easily modifiable configuration file (shown in Listing 2). Table I illustrates the options for per segment output to the client logs. The default headers are always output to terminal, while the remainder of the "optional options" can be displayed either "on" or "off" through settings in the configuration file. Note: segmented DASH video content is required for goDASHbed, and a script is provided to download the content of your choice. We propose that you use either of the following recently released DASH datasets (work with goDASH and goDASHbed): the AVC and HEVC multi-profile UHD dataset [26] which provides multiple profiles per UHD clip, while the Multi-Codec Dataset [27] consists of content in a range of video encoders (AVC, HEVC, VP9, and AV1) and resolutions. Installation scripts for goDASH/goDASHbed can be downloaded from: http://cs1dev.ucc.ie/misl/goDASH/

III. Synopsis of goDASH

Listing 1: Template to run a single goDASH client

```
1 # ./godash --config ./config/configure.json
```

goDASH provides options for both parameter passing to the executable and an easily modifiable json configuration file. An example of the call to the configuration file is shown in Listing 1, while an example of the parameter settings in the configuration file is shown in Listing 2. As can be seen in Listing 2, there are numerous options from which to configure goDASH, such as adaptation algorithm, codec to use, initial number of segments to buffer the player, maximum buffer in seconds, max resolution height to restrict the player to, stream

TABLE II: Sample Default trace output from goDASH - conventional algorithm and 4-second segment duration

Seg_#	Arr_time	Del_Time	Stall_Dur	Rep_Level	Del_Rate	Act_Rate	Byte_Size	Buff_Level
1	133	133	0	237	4955	164	82382	4000
2	2353	1371	0	4334	25074	8594	4297176	8000
3	4043	869	0	4334	23466	5098	2549074	10310
4	5581	617	0	4334	22227	3428	1714298	12773
5	7174	651	0	4334	22222	3616	1808357	15180

TABLE III: Sample Optional output from goDASH - conventional algorithm and 4-second segment duration

Seg_#	Algorithm	Seg_Dur	Codec	Width	Height	FPS	Play_Pos	RTT	Protocol
1	conventional	4000	h264	320	180	24	0	42.893	HTTP/1.1
2	conventional	4000	h264	1920	1080	24	4000	43.482	HTTP/1.1
3	conventional	4000	h264	1920	1080	24	8000	43.123	HTTP/1.1
4	conventional	4000	h264	1920	1080	24	12000	43.969	HTTP/1.1
5	conventional	4000	h264	1920	1080	24	16000	44.560	HTTP/1.1

duration in seconds, as well as options for debug and logging output, and the MPD url to stream from.

```
{
    "adapt" : "conventional",
    "codec" : "h264",
    "debug" : "on",
    "initBuffer" : 2,
    "maxBuffer" : 60,
    "maxHeight" : 3000,
    "streamDuration" : 4,
    "storeDash" : "347985",
    "logFile" : "log_file_2",
    "getHeaders" : "off",
    "terminalPrint" : "on",
    "printHeader" :
        "{\"Algorithm\":\"off\",\"Seg_Dur\":\"off\",
        \"Codec\":\"off\",\"Width\":\"on\",
        \"Height\":\"on\",\"FPS\":\"off\",
        \"Play_Pos\":\"off\",\"RTT\":\"off\",
        \"Seg_Repl\":\"off\",\"Protocol\":\"on\"}",
    "hls" : "off",
    "expRatio": 0.2,
    "quic" : "off",
    "url" : "[http://cs1dev.ucc.ie/misl/
        4K_non_copyright_dataset/2_sec/
        x264/bbb/DASH_Files/full/bbb_enc_x264_dash.mpd]",
    "useTestbed" : "off"
}
```

Listing 2: Sample goDASH configuration file

To supplement the original configuration setup, we also add an "evaluate" framework to goDASH, which offers a means of running multiple goDASH clients during one streaming session, using an easily configurable Python script. Through this framework, each client utilises its own configure.json file, and generating its own log and output files. An example of the call to the Python script is shown in Listing 3.

Listing 3: Template to multiple goDASH clients

```
# python3 ./test_goDASH.py --numClients=10 \
    --terminalPrint="off" --debug="off"
```

Currently, the Python script utilises a library file containing a list of the possible MPD urls to choose from the five profiles of the AVC and HEVC UHD DASH datasets [26]. Each client randomly chooses a different MPD, and downloads the relevant number of segments depending on the stream duration, and segment duration. –numClients - defines the number of goDASH clients to stream, –terminalPrint - determines if the clients should output their logs to the terminal screen and –debug - defines if the debug logs should be created - note: even if "debug" is set to "off", a log file, "logDownload.txt", containing the output features of each downloaded segment will be created per client in the output log folders. "test_goDASH.py" has been tested with up to 100 goDASH clients with no loss to the content of the output logs. Each client consumes up to 0.7% of CPU on average.

A. goDASH streaming log output

As presented in Table I, the output logs generated by goDASH can be broken into two distinct outputs, namely Default (an example of which is shown in Table II), and Optional (shown in Table III) (These logs were generated by the configuration settings shown in Listing 2) To view any of the Optional outputs, the respective key in the "printHeaders" dictionary of the configuration.json file, must also be set to "on". The steps to set the various output parameters in the configuration setup are clearly defined in the README file in goDASH (https://github.com/uccmisl/godash.git).

IV. Conclusions and Future Work

In this paper, we present *goDASH*, a novel headless DASH player, written in Google Go (golang). goDASH is a scalable emulation video streaming player that supports TCP and QUIC transport protocols and enables large-scale experiments and background traffic from different application types. Future work will consider adding the real-time generation of qualitative output for well-known Quality of Experience (QoE) models, such as, but not limited to, P.1203 [28] and Claye [29]. Through continued modified to the player, we hope to offer a validation framework which can be used in the creation of new HAS algorithms and QoE models.

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

REFERENCES

[1] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 562–585, Firstquarter 2019.

[2] J. Jiang, V. Sekar, and H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive," *IEEE/ACM Transactions on Networking*, vol. 22, no. 1, pp. 326–340, Feb 2014.

[3] 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.

[4] T.-Y. 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 *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 187–198.

[5] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[6] 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*. IEEE, Dec 2013, pp. 1–8.

[7] A. H. Zahran, D. Raca, and C. Sreenan, "ARBITER+: Adaptive Rate-Based InTElligent HTTP StReaming Algorithm for Mobile Networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2018.

[8] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "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*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 325–338. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787486

[9] A. H. Zahran, J. Quinlan, D. Raca, C. J. Sreenan, E. Halepovic, R. K. Sinha, R. Jana, and V. Gopalakrishnan, "OSCAR: An Optimized Stall-cautious Adaptive Bitrate Streaming Algorithm for Mobile Networks," in *Proceedings of the 8th International Workshop on Mobile Video*, ser. MoVid '16. New York, NY, USA: ACM, 2016, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/2910018.2910655

[10] D. Raca, A. H. Zahran, C. J. Sreenan, R. K. Sinha, E. Halepovic, R. Jana, and V. Gopalakrishnan, "On Leveraging Machine and Deep Learning for Throughput Prediction in Cellular Networks: Design, Performance, and Challenges," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 11–17, March 2020.

[11] 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*, ser. SIGCOMM '17.

[12] "dash.js," https://github.com/Dash-Industry-Forum/dash.js, accessed: 2020-03-29.

[13] "Gpac," https://gpac.wp.imt.fr/, accessed: 2020-03-29.

[14] "Exoplayer," https://github.com/google/ExoPlayer, accessed: 2020-03-29.

[15] 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: ACM, 2014, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2676652.2676654

[16] 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)*, June 2015, pp. 1765–1770.

[17] 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*.

[18] "Goolge go - golang," https://golang.org, accessed: 2020-03-29.

[19] 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, April 2014.

[20] A. Botta, A. Dainotti, and A. Pescapè, "A Tool for the Generation of Realistic Network Workload for Emerging Networking Scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[21] Raca, Darijo and Quinlan, Jason J. and Zahran, Ahmed H. and Sreenan, Cormac J., "Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics," in *Proceedings of the 9th ACM Multimedia Systems Conference*. Association for Computing Machinery, 2018.

[22] D. Raca, D. Leahy, C. J. Sreenan, and J. J. Quinlan, "Beyond Throughput, The Next Generation: a 5G Dataset with Channel and Context Metrics," in *11th ACM Multimedia Systems Conference*, ser. MMSys '20, 2020.

[23] G. Hong, J. Martin, and J. M. Westall, "On fairness and application performance of active queue management in broadband cable networks," *Computer Networks*, vol. 91, pp. 390 – 406, 2015.

[24] A. Mansy, B. Ver Steeg, and M. Ammar, "SABRE: A Client Based Technique for Mitigating the Buffer Bloat Effect of Adaptive Video Flows," in *Proceedings of the 4th ACM Multimedia Systems Conference*, ser. MMSys '13. ACM, 2013, pp. 214–225.

[25] D. Raca, A. H. Zahran, and C. J. Sreenan, "Sizing network buffers: An http adaptive streaming perspective," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, Aug 2016, pp. 369–376.

[26] 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*.

[27] Zabrovskiy, Anatoliy and Feldmann, Christian and Timmerer, Christian, "Multi-codec DASH Dataset," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018.

[28] A. Raake, M.-N. Garcia, W. Robitza, P. List, S. Göring, and B. Feiten, "A bitstream-based, scalable video-quality model for HTTP adaptive streaming: ITU-T P.1203.1," in *Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, May 2017.

[29] Petrangeli, S. and Famaey, J. and Claeys, M. and Latré, S. and De Turck, F., "QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming," *ACM Trans. Multimedia Comput. Commun. Appl.*, Oct. 2015.