CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	godash 2.0 - the next evolution of HAS evaluation
Authors	O'Sullivan, John;Raca, Darijo;Quinlan, Jason J.
Publication date	2020-08-31
Original Citation	O'Sullivan, J., Raca, D. and Quinlan, J. J. (2020) 'Godash 2.0 - The Next Evolution of HAS Evaluation', IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), Cork, Ireland, 31 Aug.-3 Sept., pp. 185-187. doi: 10.1109/WoWMoM49955.2020.00043
Type of publication	Conference item
Link to publisher's version	http://www.cs.ucc.ie/wowmom2020/ - 10.1109/WoWMoM49955.2020.00043https://ieeexplore.ieee.org/document/9217700
Rights	© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Download date	2026-09-14 21:08:31
Item downloaded from	https://hdl.handle.net/10468/10145

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

godash 2.0 - The Next Evolution of HAS Evaluation

John O'Sullivan†, Darijo Raca* and Jason J. Quinlan†
*Faculty of Electrical Engineering, University of Sarajevo, Sarajevo, BiH. Email: draca@etf.unsa.ba
†Computer Science & Information Technology, University College Cork, Ireland. Email: j.quinlan@cs.ucc.ie

***Abstract*—In this short demo paper, we introduce *godash* 2.0 godash is a headless HTTP adaptive streaming (HAS) video streaming platform written in the Google programming language GO. godash has been extensively rewritten for this release so as to provide ease of use, and a host of new features. godash includes options for eight different state of the art adaptive algorithms, five HAS profiles, four video codecs, the ability to stream audio and video segments, two transport protocols, real-time output from five Quality of Experience (QoE) models, as well as a collaborative framework for the evaluation of cooperative HAS streaming. godash also comes complete with its own testbed framework known as *godashbed*. godashbed uses a virtual environment to serve video content locally (which allows setting security certificates) through the Mininet virtual emulation tool. godashbed has options for large scale evaluation of HAS streaming using 4G/5G bandwidth traces, various modes of background traffic, and a choice of web server, namely: Web Server Gateway Interface (WSGI) and Asynchronous Server Gateway Interface (ASGI). In this manner, godash provides a framework for rapid deployment and testing of new HAS algorithms, QoE models and transport protocols.**

I. Introduction

On-demand video streaming is growing faster than content or network providers could imagine. With the global release of Disney+ in 2020, and with over 50 million Disney+ subscribers within weeks, the demand for high quality video content seems unabated. Couple this demand across all service providers and there are over 700 million unique subscriptions typically streaming content using a HAS model for adaptive delivery. This level of demand mandates considerable research into efficient and scalable delivery of HAS content. To this end, we developed *godash* [1], [2]. godash is a headless (non decoding) HAS platform for the development, evaluation and implementation of novel adaptive models and techniques for large-scale delivery of HAS content.

II. godash Design and Configuration

Due to page limitation, in this section we introduce the options and configuration setup available in godash, that we will present in the demo session. A detailed overview of godash is presented in [1]. Installation scripts for godash, godashbed and associated dependencies for Ubuntu 20.04 are freely available to download[1].

godash utilises a command-line terminal interface to build and deploy the player using the Google programming language GO (golang). The easiest option available to run the player as shown in listing 1, is to use the pre-defined configuration file (example shown in listing 2).

[1]http://bit.ly/2WZRTZ8

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

Listing 1: Command to run a single godash client using a configuration file

```
{
    "adapt" : "conventional",
    "codec" : "h264",
    "debug" : "on",
    "initBuffer" : 2,
    "maxBuffer" : 60,
    "maxHeight" : 1080,
    "streamDuration" : 20,
    "storeDash" : "on",
    "outputFolder" : "123456",
    "logFile" : "log_file_2",
    "getHeaders" : "off",
    "terminalPrint" : "on",
    "printHeader" :
        "{\"Algorithm\":\"on\",\"Seg_Dur\":\"on\",
        \"Codec\":\"on\",\"Width\":\"on\",
        \"Height\":\"on\",\"FPS\":\"on\",
        \"Play_Pos\":\"on\",\"RTT\":\"on\",
        \"Seg_Repl\":\"off\",\"Protocol\":\"on\",
        \"P.1203\":\"on\",\"Clae\":\"on\",
        \"Duanmu\":\"on\",\"Yin\":\"on\",
        \"Yu\":\"on\"}",
    "expRatio" : 0.2,
    "quic" : "off",
    "useTestbed" : "on",
    "url" :
        "[http://godashbed.org/4K_dataset/4_sec/x264/bbb/
         DASH_Files/full/bbb_enc_x264_dash_video.mpd]",
    "QoE" : "on",
    "serveraddr" : "on"
}
```

Listing 2: Sample godash configuration file

Options are also available to call godash using each individually parameter available in the configuration file (sample shown in listing 3).

```
# ./godash -url "[<mpd_url>]" -adapt conventional \
  -codec h265 -debug on -initBuffer 2 \
  -maxBuffer 20 -maxHeight 1080 \
  -streamDuration 10 -storeDASH on -debug on \
  -terminalPrint on -outputFolder 123456 \
  -logFile log_file_2
```

Listing 3: Command to run a single godash client using individual parameters (only some parameters shown)

Based on the configuration file, godash provides options for:

- -adapt : adaptation algorithms such as - Hybrid: Arbiter+ [3] and Elastic [4], Buffer Based: Logistic [5] and BBA [6], and Rate Based: Conventional [7], Progressive, Average, Geometric and Exponential.[2]

[2]The last four are averaging methods for measured throughput and not full fledged implementation, but serves as good baseline in experiments

- -codec : video codec: h264, h265, VP9 and AV1
- -debug : video stream debug option for printing information
- -initBuffer : defining the initial number of segments to download before stream starts (start up phase)
- -maxBuffer : defining the maximum stream buffer in seconds
- -maxHeight : defining a maximum height resolution to stream
- -streamDuration : duration in seconds of content to download
- -storeDash : ability to store the downloaded audio/video segments
- -outputFolder : defining a folder location to store the streamed DASH files
- -logFile : optional name to give the output log file
- -getHeaders : MPD url header information extracting for all segments
- -terminalPrint : printing log output to file/terminal columns based on selected print headers
- -printHeader : print the optional output logs
- -expRatio : used by some of the algorithms
- -quic : video streaming using the TCP/QUIC transport protocol ("quic" : "off", means use TCP)
- -useTestbed : using godashbed
- -url : url to the HAS MPD file to use. The AVC and HEVC multi-profile UHD dataset [8] is used in this instance
- -QoE : five well-known QoE metrics from the literature. Our choice of models are: the standardised ITU-T Rec. P.1203 QoE model [9] (where we implement mode 0), Claey [10], Dunamu [11], Yin [12], and Yu [13]
- -serveraddr : collaborative framework for sharing DASH content between multiple clients using consul [14] and gRPC [15]

While not specifically shown in the configuration file, godash also provides options for streaming audio and video HAS segmented content, and supports five HAS profiles: full, main, live, full_byte_range and main_byte_range.

While godash is typically utilised using a single client, we also provide an additional option for testing of godash using a small-scale number of clients, using the 'evaluate' folder. Listing 4 offers a sample call to the './test_goDASH.py' script in the 'evaluate' folder. As can be seen, the script also provides the option of deploying godash clients using the collaborative framework. We define collaborative as a means of sharing HAS content locally between clients using consul [14] (automate network configurations and service discover framework) and gRPC [15] (pen-source high-performance remote procedure call frame-work), thus reducing demand on the backhaul network(s).

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

Listing 4: Template to multiple godash clients

III. GODASHBED DESIGN AND CONFIGURATION

As stated, godash also comes complete with its own testbed framework, known as *godashbed* [16]. godashbed uses a virtual environment to serve video content locally (which allows setting security certificates) through the Mininet virtual emulation tool and provides a greater range of environments in which to evaluate large scale HAS delivery.

godashbed provides options for:

- –bw-net, -b - Bandwidth of bottleneck link
- –delay - Delay in milliseconds of bottleneck link
- –numruns - Number of times experiment will be repeated (default 1). This number is based on number of files in the 'traces' folder. We include one trace from two existing cellular trace datasets (4G [17] and 5G [18]) in godashbed, for ease of use.
- –voipclients - Number of voip clients (default 0). VOIP background traffic is provided through the Distributed Internet Traffic Generator (*D-ITG*) [19]
- –videoclients - Number of video clients (default 0)
- –tm - Transport mode (TCP - HTTP/HTTPS or QUIC - HTTPS)
- –duration - Duration of experiment in seconds
- –bwKPI - Name of the column indicating throughput (default="DL_bitrate")
- –debug - Print output of godash to a log file
- –terminalPrint - output godash logs to the commandline
- –server - Choice of Web server hosting HAS content - Caddy 2 [20] - Web Server Gateway Interface (WSGI - TCP/QUIC), and Hypercorn/Quart [21] - Asynchronous Server Gateway Interface (ASGI - TCP/QUIC)
- –collaborative - Run the evaluation in collaborative mode, and share content between the clients (based only on client requests) - currently TCP only

Through these settings, godashbed accommodates large-scale evaluation of numerous HAS environments. Thus offering a dynamic, adaptive and scalable framework for HAS evaluation.

IV. EXPERIMENTAL RESULTS

In this section, we present sample evaluation results for a single client streaming in a three client collaborative video only streaming session. Table III, presents the notion used in the godash output logs. The output logs generated by godash can be broken into three distinct outputs, namely Default (an example of which is shown in Table I), Optional and QoE (both shown in Table II) (These logs were generated by the configuration settings shown in Listing 2)

As can be seen, the ouput logs are sufficiently detailed to offer numerous features upon which evaluate can be determined. These include QoE values and their variance, transmission protocols, and delivery time. If we focus on 'Del_Time' from Table I, we can see that segment 5 was requested locally, as the delivery time is very low (and Del_Rate very high) in comparison to other similarly sized segments. We can also see that the RTT, from Table II, for this segment is also low in comparison to previous segment requests.

TABLE I: 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	490	256	0	237	1534	98	49093	4000
2	3313	611	0	1085	14616	2232	1116360	8000
3	6121	1362	0	3832	16025	5456	2728417	9192
4	9426	1681	0	4282	9502	3993	1996631	9887
5	11514	464	0	4282	34173	3964	1982069	11800

TABLE II: Sample Optional — QoE output from godash - conventional algorithm and 4-second segment duration

Seg_#	Algorithm	Seg_Dur	Codec	Width	Height	FPS	Play_Pos	RTT	Protocol	P.1203	Claey	Duanmu	Yin	Yu
1	conventional	4000	h264	320	180	60	0	74.922	HTTP/1.1	1.871	0.000	46.465	-11762.149	0.238
2	conventional	4000	h264	640	360	60	4000	188.692	HTTP/1.1	2.543	0.163	40.823	-23524.298	0.661
3	conventional	4000	h264	1920	1080	60	8000	193.372	HTTP/1.1	3.288	0.097	48.888	1560.751	1.719
4	conventional	4000	h264	1920	1080	60	12000	322.236	HTTP/1.1	3.558	0.169	53.060	5393.492	2.359
5	conventional	4000	h264	1920	1080	60	16000	96.492	HTTP/1.1	3.701	0.235	55.752	9675.827	2.744

TABLE III: Notation used in the godash Trace Output logs

Type	**Description**
Default Output:	
Seg_#	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	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:	determined by 'printHeader' config option
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)
Protocol	HTTP protocol
QeM Output:	determined by 'QoE' config option
P.1203	P.1203 standard - scale $[0, 5]$
Clae	Clae model - scale $[0, 5]$
Duanmu	Duanmu model - scale $[0, 100]$
Yin	Yin model - scale dependent on HAS bitrates
Yu	Yu model - scale $[0, 5]$

V. Conclusion

In this short demo paper we presented an overview, configuration steps and results for *godash* - a headless HAS video streaming platform and *godashbed* - a testbed framework for large-scale HAS evaluation. This demo illustrates the ease of use of the godash platform and its various components, while also offering a framework for rapid deployment and testing of new HAS algorithms, QoE models and transport protocols.

*Acknowledgement:*The authors acknowledge the support of Science Foundation Ireland (SFI) under Research Grant 13/IA/1892 and European Regional Development Fund under SFI Grant 13/RC/2077.

References

[1] D. Raca, M. Manifacier, and J. J. Quinlan, "goDASH - GO accelerated HAS framework for rapid prototyping," in *Proceedings of the 12th International Conference on Quality of Multimedia Experience*, 2020.

[2] "godash - GO accelerated HAS framework for rapid prototyping," https://github.com/uccmisl/godash, accessed: 2020-05-25.

[3] A. H. Zahran *et al.*, "ARBITER+: Adaptive Rate-Based InTElligent HTTP StReaming Algorithm for Mobile Networks," *IEEE Transactions on Mobile Computing*.

[4] L. D. Cicco *et al.*, "ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH)," in *2013 20th International Packet Video Workshop*.

[5] Y. Sani *et al.*, "Modelling Video Rate Evolution in Adaptive Bitrate Selection," in *2015 IEEE International Symposium on Multimedia (ISM)*, Dec 2015, pp. 89–94.

[6] T. Huang *et al.*, "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.

[7] Z. Li *et al.*, "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.

[8] J. J. Quinlan *et al.*, "Multi-profile Ultra High Definition (UHD) AVC and HEVC 4K DASH Datasets," in *9th ACM MMSys Conference*.

[9] W. Robitza *et al.*, "HTTP Adaptive Streaming QoE Estimation with ITU-T Rec. P. 1203: Open Databases and Software," in *9th ACM Multimedia Systems Conference*, ser. MMSys '18, 2018, pp. 466–471.

[10] S. Petrangeli *et al.*, "QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming," *ACM Trans. Multimedia Comput. Commun.*

[11] Z. Duanmu *et al.*, "A Quality-of-Experience Database for Adaptive Video Streaming," *IEEE Transactions on Broadcasting*, vol. 64, no. 2, pp. 474–487, June 2018.

[12] X. Yin *et al.*, "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," in *2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15, 2015.

[13] L. Yu *et al.*, "QoE-Driven Dynamic Adaptive Video Streaming Strategy With Future Information," *IEEE Transactions on Broadcasting*, Sep.

[14] "Consul - Automate network configurations and service discover framework," https://consul.io, accessed: 2020-05-25.

[15] "GRPC - open-source high-performance remote procedure call framework," https://godoc.org/google.golang.org/grpc, accessed: 2020-05-25.

[16] "godashbed - Testbed framework for HAS streaming," https://github.com/uccmisl/godashbed, accessed: 2020-05-25.

[17] D. Raca *et al.*, "Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics," in *9th ACM MMSys Conference*.

[18] ——, "Beyond Throughput, The Next Generation: a 5G Dataset with Channel and Context Metrics," in *11th ACM Multimedia Systems Conference*, ser. MMSys '20, 2020.

[19] A. Botta *et al.*, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, 2012.

[20] "Caddy2 is a powerful, extensible platform to serve sites, services, and apps, written in Go," https://caddyserver.com/v2, accessed: 2020-05-25.

[21] "Hypercorn - an Asynchronous Server Gateway Interface (ASGI) web server ," https://pgjones.gitlab.io/hypercorn/, accessed: 2020-05-25.