Open-Source Software Tools for Measuring Resources Consumption and DASH Metrics

Mario Montagud1,2, Juan Antonio De Rus1, Rafael Fayos-Jordan1, Miguel Garcia-Pineda1, Jaume Segura-Garcia1

1Universitat de València (UV), Departament d’Informàtica, València (Spain)
2i2CAT Foundation, Barcelona (Spain)
{monclima; derusa; rafael.fayos, Miguel.Garcia-Pineda, jaume.segura}@uv.es

ABSTRACT
When designing and deploying multimedia systems, it is essential to accurately know about the necessary requirements and the Quality of Service (QoS) offered to the customers. This paper presents two open-source software tools that contribute to these key needs. The first tool is able to measure and register resources consumption metrics for any Windows program (i.e. process id), like the CPU, GPU and RAM usage. Unlike the Task Manager, which requires manual visual inspection for just a subset of these metrics, the developed tool runs on top of the Powershell to periodically measure these metrics, calculate statistics, and register them in log files. The second tool is able to measure QoS metrics from DASH streaming sessions by running on top of TShark, if a non-secure HTTP connection is used. For each DASH chunk, the tool registers: the round-trip time from request to download, the number of TCP segments and bytes, the effective bandwidth, the selected DASH representation, and the associated parameters in the MPD (e.g., resolution, bitrate). It also registers the MPD and the total amount of downloaded frames and bytes. The advantage of this second tool is that these metrics can be registered regardless of the player used, even from a device connected to the same network than the DASH player.

CCS CONCEPTS
• Information systems ➔ Information systems applications
• Software and its engineering ➔ Software creation and its management

KEYWORDS
DASH, Open-Source Software, Performance Metrics, Quality of Service (QoS), Traffic Analysis

1 Introduction
In multimedia systems, it is essential to know about the required resources for and effective deployment and usage, as well as about the Quality of Service (QoS) that is, or can be, offered, to the customers.

On the one hand, many multimedia applications are resources intensive, so it becomes very valuable to accurately know about the resources that are actually used to take the corrective actions and/or to choose the adequate processing devices for seamlessly running these applications. A typical example is the execution of media players for traditional and/or immersive formats. In particular, in Virtual Reality (VR) scenarios, in which many heterogeneous media elements and advanced interaction features can be integrated (e.g. [1] [2]), the hardware and software requirements of the processing devices become key aspects to take into account. Therefore, appropriate tools are needed to accurately measure resources consumption metrics (e.g., CPU, GPU, RAM, data exchange…). These tools can be used, for instance, to benchmark specific applications, to determine the impact of the inclusion of different media elements and features, and to select and/or adapt the processing devices with the required hardware and software resources to guarantee satisfactory QoS and Quality of Experience (QoE) levels.

On the other hand, multimedia streaming services, like those making use of Dynamic Adaptive Streaming over HTTP (DASH) [3] [4], raise key challenges to provide satisfactory QoS and QoE levels to the consumers. That is especially true when distributing high-quality media over non-controlled network environments, and when using equipment with limited resources. DASH provides key advantages to contribute to this, as the client devices can dynamically select the available media quality that best fits the current network and consumption device status. In the last years, the research community and industry are devoting significant efforts to overcome remaining challenges in DASH.
streaming scenarios. Relevant examples are: bandwidth prediction (e.g. [5]); latency minimization (e.g. [6]); design of dynamic media quality switching strategies (e.g. [7]); scalability (e.g. [8]); and QoE modeling (e.g. [9]). All these research objectives and initiatives share a common denominator: it is essential to measure and register relevant QoS and QoE related metrics to better understand what are the limitations, propose appropriate solutions and corroborate the obtained performance and benefits, probably in comparison with benchmarking or alternative solutions.

The availability of media players with appropriate probes and Application Program Interfaces (API) for measuring valuable metrics is of great value to succeed in all these goals. The most relevant example is the DASH Reference Player (dash.js)\(^1\), which comes with a quite complete and handy API for measuring valuable metrics, and even with tools for visualizing them. However, unfortunately these tools cannot be also always used and/or may not completely meet the targeted requirements. In some situations, other players are being used and cannot be replaced. These other players may not provide tools for measurement of metrics, and may not be open-source. Therefore, although they may be able to play out the desired DASH streams, they may be considered as black boxes that prevent from actually measuring key QoS and QoE related metrics. In some other situations, like when using Unity based scenarios, the dash.js player cannot be integrated, and not many compatible DASH players exist. Likewise, it can be beneficial to obtain further relevant metrics and information about the DASH streaming session beyond the ones provided by dash.js. Examples are: the round-trip times since request to full download for each DASH chunk; the number of TCP segments and bytes for each chunk, and information available in the Media Presentation Description (MPD) metadata file describing the DASH assets (e.g. resolutions, rates….). In addition, the availability of an external tool to measure and register the targeted metrics can be very beneficial to compare the performance and behavior of different players for specific DASH streams.

This paper presents two open-source software tools that have been developed to provide effective solutions to the discussed needs and limitations in terms of metrics measurement. First, a tool for the measurement of Resources Consumption Metrics (called RCM-UV tool, where UV stands for the University of Valencia, entity having developed it) has been developed to evaluate the performance and requirements for different Windows applications. The RCM-UV tool is able to measure the usage of CPU, GPU, RAM for any Windows program (i.e. process id) by just providing its name. Unlike the Task Manager, which requires manual visual inspection for just a subset of these metrics, the developed tool runs on top of the Windows Powershell\(^2\) to periodically measure these metrics, calculate statistics, and register them in log files. The rationale behind developing the RCM-UV tool was to analyze how many resources are used/necessary to simultaneously play out different combinations of media elements in ad-hoc Unity-based and web-based players. However, their applicability spans to other media scenarios and applications.

Second, a tool for the measurement of QoS and QoE related metrics from DASH streaming sessions (called DQM-UV tool, where again UV stands for the University of Valencia, entity having developed it) has been developed to evaluate the performance and behavior of media applications making use of DASH streaming over non-secure HTTP connections. For each DASH chunk, the tool registers: the round-trip time from request to download; the number of TCP segments and bytes; the effective bandwidth; the selected DASH representation; and the associated parameters in the MPD (e.g. resolution, frames per second, bitrate…). The MPD and the total amount of downloaded frames and bytes are also registered. The rationale behind developing the RQM-UV tool was to analyze the obtained QoS, and infer the perceived QoE, in different DASH streaming scenarios, considering both traditional and immersive media formats, via different network connections, under different conditions, and when using different media players and Adaptive Bitrate (ABR) strategies. The advantage of this tool is that these metrics can be registered regardless of the player being used, even from a device connected to the same network than the player.

Both software tools are completely operational and have been released on Github (https://github.com/ETSE-UV/), such that the community can take advantage of their existence, and customize them to the particular needs, if necessary. A detailed documentation about how to run and use these tools is also provided in the Github repository.

The remainder of this paper is organized as follows. Section 2 provides a brief review of related tools and works. Then, Section 3 provides more details about the developed tools. Finally, Section 4 presents some conclusions and plans for future work.

2 Related Work

The measurement of performance and QoS metrics has been traditionally a fundamental aspect when designing and deploying multimedia systems. This section briefly indicates existing practices, and reviews tools and research works, in this domain.

2.1 Resources Consumption Metrics Measurement

In Windows applications, the usage of resources consumption metrics is typically determined by relying on the Task Manager, and by taking snapshots or manual annotations at certain times. Although this process could be somehow automated by developing routines and making use of Optical Character Recognition (OCR) tools, it is still not very flexible and convenient. In addition, the Task Manager provides valuable

\(^1\) Open-Source DASH Reference Player (dash.js): https://reference.dashif.org/dash.js/ Last Access in February 2020

\(^2\) Windows Powershell: https://docs.microsoft.com/en-us/powershell Last Access in February 2020
information about some metrics, but may not inform about all the metrics of interest, and/or about their associated parameters or information (e.g., used GPU engines).

If the specific application at hand is open-source or developed from scratch, then specific Operating System (OS) tools and libraries could be used to measure the desired metrics. However, this situation does not always happen, but the measurement of these metrics when using (closed) third-party applications is often desired. Therefore, the development of an external tool for these measurements allows overcoming the above limitations, while decoupling the execution of tasks from the measurement of metrics.

### 2.2 DASH Metrics Measurement

In DASH streaming scenarios, the measurement and analysis of QoS and QoE related metrics is commonplace. For instance, this is typically used to determine the benefits of novel techniques to improve the performance in terms of the challenges introduced in Section 1. Most of the existing works in these domains (e.g., [5-9]) make use of the APIs provided by existing DASH players, like dash.js or the Bitmovin Analytics API, or simulation tools, like Network Simulator 3 (ns-3). Other recent works have built end-to-end evaluation testbeds, by relying on these players and APIs and extending them with registration and analysis features (e.g., [9], [10]). However, the same limitations discussed for the measurement of resources consumption metrics apply to the measurement of DASH metrics. On the one hand, it may happen that the available measurement tools do not provide all metrics of interest, like the size and round-trip times for each DASH chunk. On the one hand, the available players and APIs cannot be always used, but closed third-party players need to be used or analyzed. Therefore, the availability of an external and flexible tool for such purposes becomes very useful.

### 3 Performance / QoS Metrics Measurement Tools

This section describes the RCM-UV and DQM-UV tools, including their objectives, requirements, key implementation aspects, and the necessary steps to make use of them. Both tools are console based, and have been developed by using Microsoft Studio, in C# language.

#### 3.1 Resources Consumption Metrics (RCM-UV) Measurement Tool

**3.1.1 Objectives.**

The RCM-UV tool has been developed to be able of measuring and registering relevant resources consumption metrics when running either single process or multi-process Windows programs, given their process name. In particular, it registers the percentage of CPU usage, the percentage of GPU usage (clustered by the used GPU engines), and the RAM usage. When the process is finished, the tool calculates statistics (max, min and average values, per interval if desired) and writes a log file (with the execution time as id) for posterior analysis.

**3.1.2 Requirements.**

- **Requirement a)** The software needs to be run in Windows devices, because it makes use of the *Windows PowerShell*. Its previous installation is then required.

- **Requirement b)** The same rationale for measuring this metric than for the CPU usage applies. It is measured in amount of memory (bytes).

- **Requirement c)** GPU usage: The same rationale for measuring this metric than for the CPU usage applies. It is especially relevant when performing video- or graphic-related processing tasks. It can be measured in terms of percentage and bytes, and gives information about the number of engines used.

**3.1.4 Implementation Overview.**

The RCM-UV tool makes use of the *<<System.Management.Automation.dll>>* library of the *Windows PowerShell* to trigger queries for the particular process id (PID) of the indicated process name from which to retrieve the targeted metrics. An example of how to get the PID is shown below:

```csharp
Process[] activeProc = Process.GetProcessesByName("process_name");
int unity_pid = activeProc[0].Id;
```

An example of the queries sent to the *PowerShell* to retrieve the CPU usage is shown below:

```ps
//CPU usage: query
String ps_script_cpu_first= "(@Get-WmiObject -class Win32_PerfFormattedData_PerfProc_Process -filter IDProcess="");
String ps_script_cpu_second= " | select-object PercentProcessorTime";
//CPU usage: parsing the obtained values
cpu_str = PowerShell.Create()
    .AddScript(ps_script_cpu_first + unity_pid + ps_script_cpu_second)
    .AddCommand("(Out-String)")
    .Invoke<Dictionary<string,string>>();
```

---

1. Bitmovin Analytics API Reference: https://bitmovin.com/docs/analytics/api-reference/all Last Access in February 2020
2. Network Simulator 3 (ns-3): https://www.nsnam.org/ Last Access in February 2020
The source code for the complete program lifecycle, for each of the considered metrics, and for their posterior statistical analysis, can be checked at the Github.

### 3.1.5 Usage

**Step 0.** Make sure that the *Windows PowerShell* is successfully installed before running the tool.

**Step 1.** Run the RCM-UV tool (e.g., double click on the .exe file).

**Step 2.** Indicate the name of the program to be analyzed. Then, the RCM-UV tool will check if the specific program is already running to start registering the resources consumption metrics. If not, the tool will ask the user to start running it.

**Step 3.** While the program is running, the tool will periodically measure the metrics – the measurement interval can be configurable - and display the obtained values in the console. When the execution is completed, the tool will calculate global statistics, display them in the console (see Figure 1), and register all compiled data in a log file.

**Step 3.** The user can check the displayed information in the console and the log files (created within the *log* folder).

The necessary steps to use the RCM-UV tool are explained with further details in the *Readme* section at the Github, being accompanied by screen captures and a demo video ([YouTube](https://www.youtube.com/watch?v=vPMkxQbk0lE)).

### 3.2 DASH QoS Metrics (DQM-UV) Measurement Tool

#### 3.2.1 Objectives

The DQM tool has been developed to be able of measuring and registering QoS and QoE related metrics for DASH streaming sessions, when using either native or web-based players, even those lacking of metrics measurement capabilities. The set of metrics that are measured extends those measured by the state-of-the-art APIs and players, like e.g. *dash.js*.

![Example of the resources consumption metrics measured and displayed by the RCM-UV tool](image)

**Figure 1:** Example of the resources consumption metrics measured and displayed by the RCM-UV tool

#### 3.2.2 Requirements

**Requirement a)** The software needs to be run in Windows devices, because it makes use of the *Windows PowerShell*. Its previous installation is then required.

**Requirement b)** The software makes use of *TShark*[^5], the terminal oriented version of *Wireshark*, to monitor and analyze the DASH traffic. Its previous installation is then required, and the path to its installation folder must be provided.

**Requirement c)** The software is valid for non-secure HTTP connections. The DASH traffic over HTTPS (HTTP Secure) is encrypted, and thus cannot be analyzed (in this current version).

**Requirement d)** The software requires the capture of the MPD to run successfully. Therefore, the web browser cache must be cleared before running the software. Otherwise, in case that the MPD has been downloaded before and is stored in the cache, the software will not work successfully.

#### 3.2.3 Metrics

The following metrics are measured and registered by making use of the DQM-UV tool:

- **a) Round Trip Times for DASH chunks**: total time since an HTTP GET is sent to the DASH server (e.g., a conventional HTTP server, like Apache), and all associated TCP segments for the associated DASH chunk have been downloaded.

- **b) TCP Segments Count and Length**: number of TCP segments downloaded for each DASH chunk and their length in bytes.

- **c) URL of the selected DASH chunk**: with this information, the associated parameters from the MPD can be obtained, like the codec, resolution, framerate, bitrate, audio channels, audio frequency, etc. This information is extracted from the MPD, which is also captured and downloaded.

- **d) Effective bandwidth**: it is calculated by dividing the total size of each chunk into the time required to fully download it.

- **e) Total number of frames and bytes downloaded**: This is relevant to know about the total amount of data downloaded, enabling e.g. a comparison between conditions and players.

#### 3.2.4 Implementation Overview

The DQM-UV tool makes use of the `<System.Management.Automation.dll>` library of the *Windows PowerShell* to obtain the PID of the process name under study. If a URL of the MPD is provided, then the tool obtains its IP address as follows:

```csharp
//remove http://
URL = URL.Replace("http://", ";");
//remove everything after the first /
URL = URL.Substring(0, URL.IndexOf("/")));…
IPHostEntry host = Dns.GetHostEntry(URL);
// if error, it is checked if the URL is already a valid IP
```

[^5]: TShark: [https://www.wireshark.org/docs/man-pages/tshark.html](https://www.wireshark.org/docs/man-pages/tshark.html) Last Access in February 2020
Tools for Measuring Resources Consumption and DASH Metrics

If the URL of the MPD is not provided, then the active network connections are checked, by using Netstat\textsuperscript{*} combined with a PID filter:

\begin{verbatim}
Process netstat = new Process();
netstat.StartInfo.FileName = "cmd";
netstat.StartInfo.Arguments = @"/C ""netstat -ano -p TCP | findstr "$" process_pid;"
netstat.StartInfo.CreateNoWindow = false;
etstat.StartInfo.UseShellExecute = false;
netstat.StartInfo.RedirectStandardOutput = true;
do {
    Thread.Sleep(1000);
    netstat.Start();
    response_NetStat
=netstat.StandardOutput.ReadToEnd();
    netstat.WaitForExit();
} while (response_NetStat == "");
\end{verbatim}

When a connection is found, then its IP address can be used as a filter for the traffic capture via TShark:

\begin{verbatim}
tshark.StartInfo.Arguments = "-I + interface_TShark + -f "host "$" -q -a duration:" + duration_TShark + "-w captureFile + "-z io,stat," + timeInterval_TShark + ",FRAMES,BYTES";
tshark.Start();
response_TShark =tshark.StandardOutput.ReadToEnd();
Console.WriteLine(response_TShark);
\end{verbatim}

The pieces of code for the capture of the MPD, and its parsing as an XML file, can be checked at the Github.

3.2.5 Usage.

Step 0. Make sure that the Windows Powershell and TShark are successfully installed, before running the DQM-UV tool.

Step 1. Run the DQM-UV tool (e.g., double click on the .exe file).

Step 2. Confirm or indicate the TShark path (it is read from a text file in the program folder).

Step 3. Select the network interface from which the DASH traffic will be captured. If no interfaces are listed, then execute the <\texttt{<sc start ncpap博文>>} command with admin rights, and try again.

Step 4. Select the time duration for the capture.

Step 5. Select the desired intervals for which the capture needs to be divided. By default, a single interval for the whole period is used.

Step 6. Select whether a single process or multiple process program will be analyzed. The selection of the multiple process option is required for web-based DASH players, as web browsers commonly instantiate new processes and threads during their execution.

\textsuperscript{*} Netstat: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/netstat Last Access in February 2020

4 Conclusions & Future Work

This paper has presented two valuable open-source software tools for the measurement of resources consumption metrics for Windows applications and of QoS metrics for DASH streaming sessions when using non-secure HTTP connections. Both tools are completely operational and have been released in Github (https://github.com/ETSE-UV/), such that the community can take advantage of their existence, and customize them to the particular needs, if necessary.
These tools are going to be used in ongoing research activities. On the one hand, the RCM-UV tool will be used to evaluate the performance and requirements for different Unity-based and web-based players, when playing out different combinations of traditional and immersive media formats using different devices. On the other hand, the DQM-UV tool will be used to analyze the obtained QoS in different DASH streaming scenarios, considering both traditional and immersive media formats, via different network connections, under different conditions, and when using different media players and ABR strategies.

Apart from these planned tasks, future work will be targeted at extending and refining these tools. First, the tools will be adapted to provide universal support, not just limited to Windows devices. It will be achieved by making use of Linux software tools and dockerizing them. Second, graphical user interfaces and (real-time) metrics visualization and analysis tools will be integrated. Third, an additional tool will be developed to be able to accurately reconstruct the received video at the client side.

ACKNOWLEDGMENTS

This work has been funded by Fundación BBVA with a “Beca Leonardo a Investigadores y Creadores Culturales 2018”, and by the Spanish Ministry of Science, Innovation and Universities with a Juan de la Cierva Incorporación grant, with reference IJC1 2017 34611. Additionally, the work has been carried out within a close collaboration with the EU H2020 VR-Together (agreement n° 762111) and ImAc projects (agreement n° 761974).

REFERENCES

[1] M. Montagud, J. A. Núñez, T. Karavellas, I. Jurado, S. Fernández, “Convergence between TV and VR: Enabling Truly Immersive and Social Experiences”, Workshop on Virtual Reality, co-located with ACM TVX 2018, Seoul (South Korea), June 2018.
[2] J. Segura-Garcia, M. Montagud, S. Mirasol-Menacho, J. Oleza-Simó “Theatrical Virtual Acoustic rendering with head movement interaction”, AI EDAM, Volume 33, Issue 3, pp. 359-368, August 2019.
[3] T. Stockhammer, “Dynamic adaptive streaming over HTTP: standards and design principles”, ACM MMVSYS 2011, San Jose, CA (USA), February 2011.
[4] DASH: https://mpeg.chiariglione.org/standards/mpeg-dash. Last Access: February 2020.
[5] A. Bentaleb, C. Timmerer, A. C. Begen, R. Zimmermann, “Bandwidth prediction in low-latency chunked streaming”, 9th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV’19), Amherst, MA (USA), June 2019.
[6] M. B. Yahia, Y. Le Louedec, G. Simon, L. Nuaymi, X. Corbillion, “HTTP/2-based Frame Discarding for Low-Latency Adaptive Video Streaming”, ACM Trans. Multimedia Comput. Commun. Appl. 15, 1, Article 18, February 2019.
[7] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann., A Survey on Bitrack Adaptation Schemes for Streaming Media Over HTTP. IEEE Communications Surveys Tutorials 21, 1, 562–585, Firstquarter 2019.
[8] J. W. Kleinrooeweler, S. Cabrero, P. Cesar, “Delivering Stable High-Quality Video: An SDN Architecture with DASH Assisting Network Elements”, ACM Multimedia Systems 2016 Conference (MMSys 16), Klagenfurt am Wörthersee (Austria), May 2016.
[9] C. Timmerer, A. Zabrowski, “Automating QoS and QoE Evaluation of HTTP Adaptive Streaming Systems”, ZTE COMMUNICATIONS, 17(1), March 2019.
[10] M. Montagud, E. Meyerson, J. Fraile, S. Fernández, “Modular Testbed for KPI Monitoring in Omnidirectional Video Streaming Scenarios”, MMEDIA 2019, València (Spain), March 2019.

Figure 2: Example of DASH metrics measured and displayed by the DQM-UV tool