A Video Streaming Application

Submitted by

Rajat Sharma – 40196467

Revanth Velagandula- 40229629

Abstract

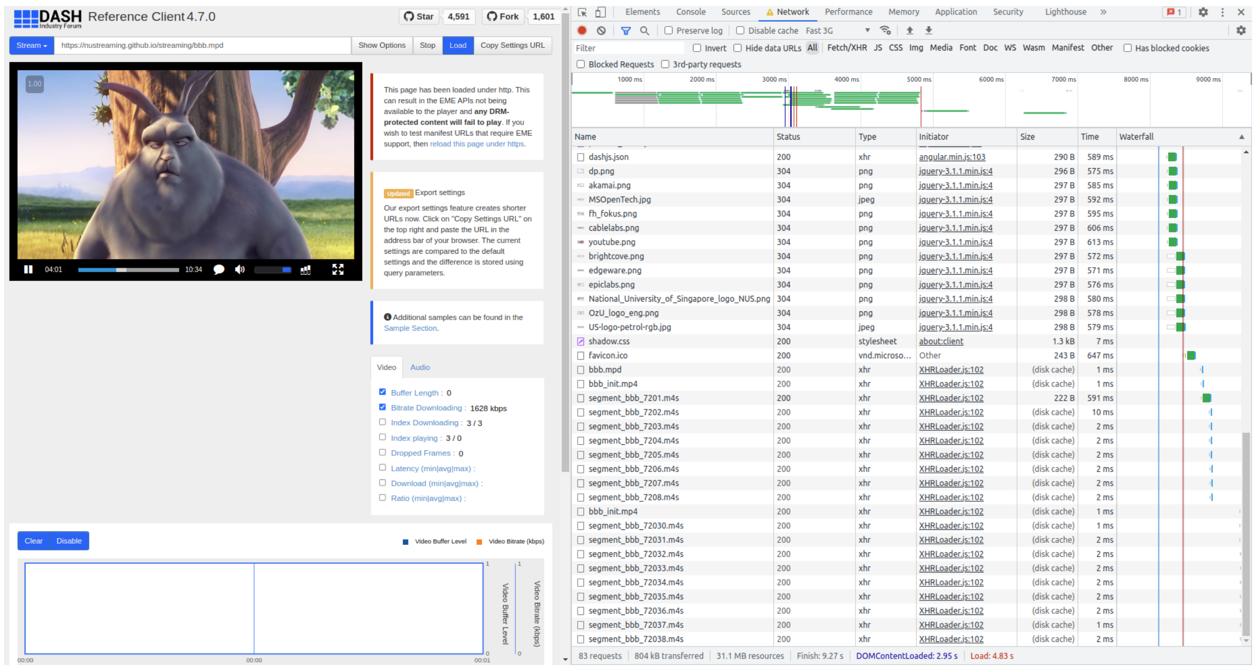
Using a media player application built using dash.js to play a saved (VoD) DASH playlist while running tests to evaluate and compare the effectiveness of built-in Adaptive BitRate (ABR) algorithms is required for this lab assignment. Using the given MPD, the server's video is fetched, and the chosen video's MPEG-DASH XML playlist file and individual video segments are downloaded (on-demand). The media player supports adaptive stream switching with ABR, which while the current streamlet is playing adaptively selects the best quality for the subsequent video streamlet to be downloaded based on the network situation. The experiments will be conducted under suggested network conditions that include Fast3G with no throttling for different time intervals. The performance metrics that will be measured include selected bitrate (Mbps), buffer level (second), measured throughput (Mbps), segment download time (second), and segment size (byte). The results of the experiments will be analyzed and compared to determine which ABR algorithm performs best under the given network conditions. During playback, metrics including the chosen bitrate, buffer level, measured throughput, segment download time, and segment size are periodically gathered.The ABR algorithms tested are Throughput-based, BOLA, and Dynamic, and the results are compared in terms of the collected metrics. Finally, the results are analyzed and discussed, and a report is written.

Introduction and motivation for work

The way individuals watch video content has changed dramatically with the introduction of streaming media over the internet. Dynamic Adaptive Streaming over HTTP (DASH) is replacing the conventional RTSP/RTP/RTCP-based approach to streaming since it has a number of benefits over its forerunner. A common method for Video on Demand (VoD) services is called DASH, which divides the media (video) into manageable, independently playable parts known as streamlets. The client media player downloads these streamlets one at a time from the web server and plays them uninterrupted. Using Adaptive BitRate (ABR) algorithms, which adaptively select the best acceptable quality for the subsequent video streamlet to be downloaded based on the network state while playing the current streamlet, DASH also offers adaptive stream switching.

The purpose of this work is to investigate and evaluate the performance of the Throughput-based, BOLA, and Dynamic built-in ABR algorithms offered by the dash.js media player. Although BOLA uses a buffer occupancy-based method that considers both the available bandwidth and the buffer occupancy, throughput-based ABR bases its decision on the observed throughput for the next video streamlet. The Dynamic ABR combines the two strategies and alternates between them depending on the state of the network.

The dash.js media player, the given MPD, and the downloading of the MPEG-DASH XML-formatted playlist file and video segments for the chosen video are the unique design and implementation decisions made in this work (on-demand). Using the media player's integrated ABR algorithms, video streamlets are switched and played in an adaptive manner while parameters including chosen bitrate, buffer level, measured throughput, segment download time, and segment size are frequently monitored during playing. The most effective ABR algorithm for various network scenarios is then determined by comparing and analysing the results.



https://nustreaming.github.io/streaming/bbb.mpd

This page has been loaded under http. This can result in the EME APIs not being available to the player and **any DRM-protected content will fail to play**. If you wish to test manifest URLs that require EME support, then reload this page under https.

Updated Export settings

Our export settings feature creates shorter URLs now. Click on "Copy Settings URL" on the top right and paste the URL in the address bar of your browser. The current settings are compared to the default settings and the difference is stored using query parameters.

Additional samples can be found in the Sample Section.

Video Audio

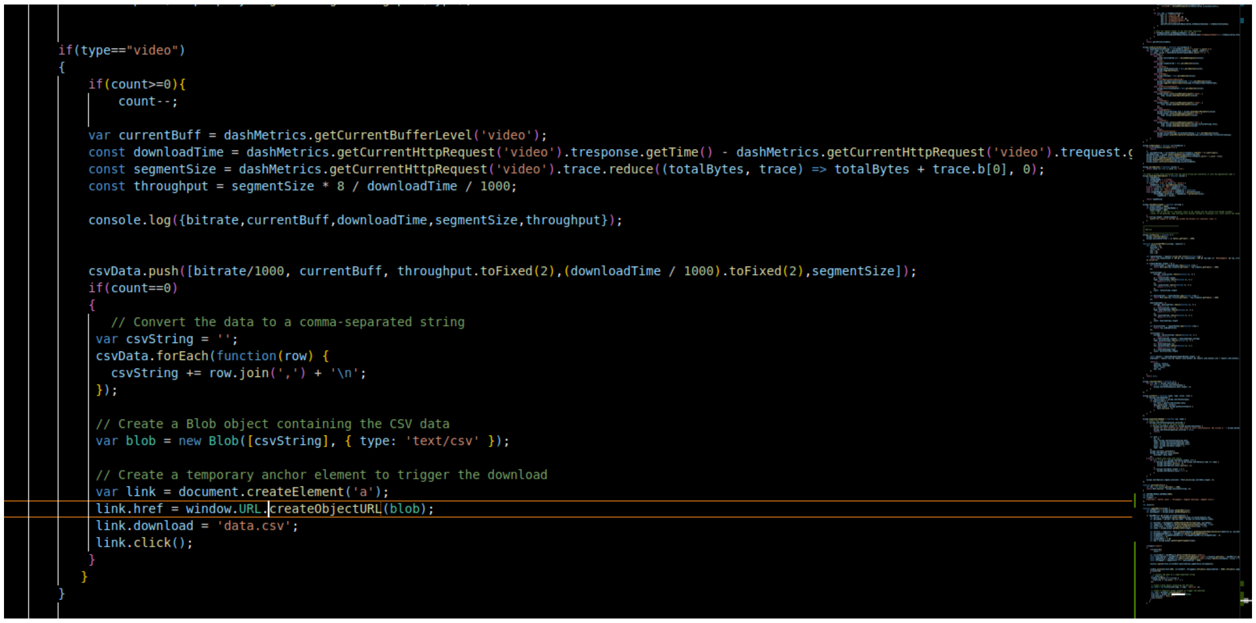
- [✓] Buffer Length : 0
- [✓] Bitrate Downloading : 1628 kbps
- Index Downloading : 3 / 3
- Index playing : 3 / 0
- Dropped Frames : 0
- Latency (min|avg|max) :
- Download (min|avg|max) :
- Ratio (min|avg|max) :

Preserve log · Disable cache · Fast 3G
Filter · Invert · Hide data URLs · All · Fetch/XHR · JS · CSS · Img · Media · Font · Doc · WS · Wasm · Manifest · Other · Has blocked cookies
Blocked Requests · 3rd-party requests

Name	Status	Type	Initiator	Size	Time
dashjs.json	200	xhr	angular.min.js:103	290 B	589 ms
dp.png	304	png	jquery-3.1.1.min.js:4	296 B	575 ms
akamai.png	304	png	jquery-3.1.1.min.js:4	297 B	585 ms
MSOpenTech.jpg	304	jpeg	jquery-3.1.1.min.js:4	297 B	592 ms
fh_fokus.png	304	png	jquery-3.1.1.min.js:4	297 B	595 ms
cablelabs.png	304	png	jquery-3.1.1.min.js:4	297 B	606 ms
youtube.png	304	png	jquery-3.1.1.min.js:4	297 B	613 ms
brightcove.png	304	png	jquery-3.1.1.min.js:4	297 B	572 ms
edgeware.png	304	png	jquery-3.1.1.min.js:4	297 B	571 ms
epiclabs.png	304	png	jquery-3.1.1.min.js:4	297 B	576 ms
National_University_of_Singapore_logo_NUS.png	304	png	jquery-3.1.1.min.js:4	298 B	580 ms
OzU_logo_eng.png	304	png	jquery-3.1.1.min.js:4	298 B	578 ms
US-logo-petrol-rgb.jpg	304	jpeg	jquery-3.1.1.min.js:4	298 B	579 ms
shadow.css	200	stylesheet	about:client	1.3 kB	7 ms
favicon.ico	200	vnd.microso…	Other	243 B	647 ms
bbb.mpd	200	xhr	XHRLoader.js:102	(disk cache)	1 ms
bbb_init.mp4	200	xhr	XHRLoader.js:102	(disk cache)	1 ms
segment_bbb_7201.m4s	200	xhr	XHRLoader.js:102	222 B	591 ms
segment_bbb_7202.m4s	200	xhr	XHRLoader.js:102	(disk cache)	10 ms
segment_bbb_7203.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_7204.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_7205.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_7206.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_7207.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_7208.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
bbb_init.mp4	200	xhr	XHRLoader.js:102	(disk cache)	1 ms
segment_bbb_72030.m4s	200	xhr	XHRLoader.js:102	(disk cache)	1 ms
segment_bbb_72031.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_72032.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_72033.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_72034.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_72035.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_72036.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms
segment_bbb_72037.m4s	200	xhr	XHRLoader.js:102	(disk cache)	1 ms
segment_bbb_72038.m4s	200	xhr	XHRLoader.js:102	(disk cache)	2 ms

83 requests | 804 kB transferred | 31.1 MB resources | Finish: 9.27 s | DOMContentLoaded: 2.95 s | Load: 4.83 s

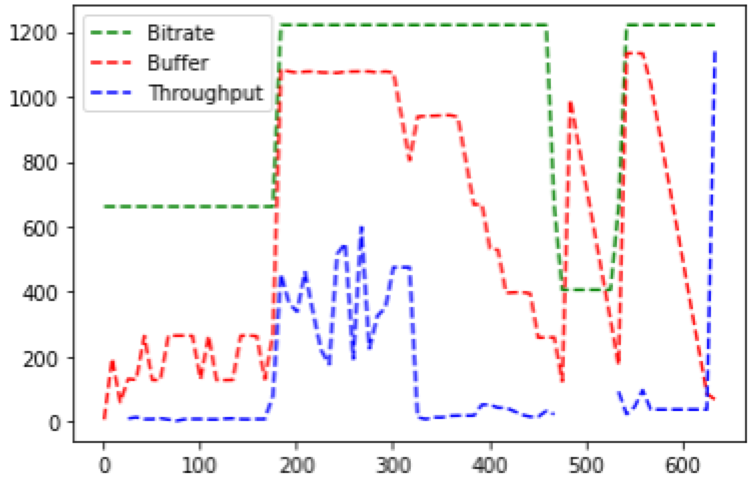
Code(main.js)



```
if(type=="video")
{
    if(count>=0){
        count--;

    var currentBuff = dashMetrics.getCurrentBufferLevel('video');
    const downloadTime = dashMetrics.getCurrentHttpRequest('video').tresponse.getTime() - dashMetrics.getCurrentHttpRequest('video').trequest.
    const segmentSize = dashMetrics.getCurrentHttpRequest('video').trace.reduce((totalBytes, trace) => totalBytes + trace.b[0], 0);
    const throughput = segmentSize * 8 / downloadTime / 1000;

    console.log({bitrate,currentBuff,downloadTime,segmentSize,throughput});


    csvData.push([bitrate/1000, currentBuff, throughput.toFixed(2),(downloadTime / 1000).toFixed(2),segmentSize]);
    if(count==0)
    {
      // Convert the data to a comma-separated string
     var csvString = '';
     csvData.forEach(function(row) {
       csvString += row.join(',') + '\n';
     });

     // Create a Blob object containing the CSV data
     var blob = new Blob([csvString], { type: 'text/csv' });

     // Create a temporary anchor element to trigger the download
     var link = document.createElement('a');
     link.href = window.URL.createObjectURL(blob);
     link.download = 'data.csv';
     link.click();
    }
   }
}
```

Results, comparison, and discussion

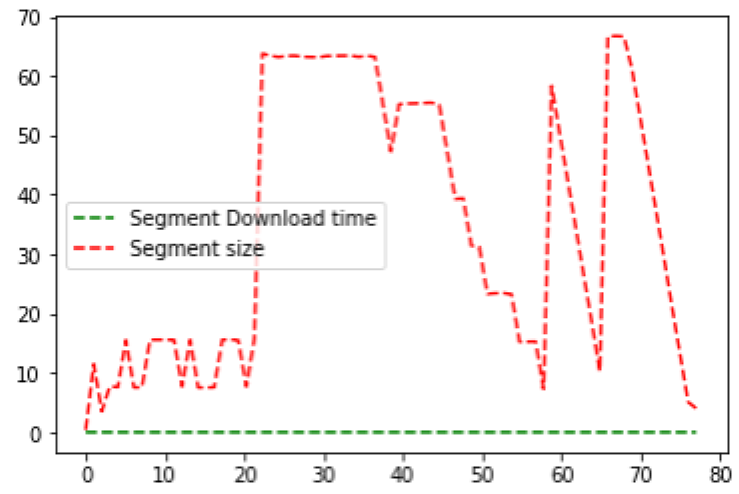
ABR:DYNAMIC



In this graph, the bitrate, buffer, and throughput are plotted over time for a video stream using ABR. The x-axis represents time in seconds, while the y-axis represents the value of each factor. The green line represents the bitrate, the red line represents the buffer, and the blue line represents the throughput.

As you can see from the graph, the bitrate, buffer, and throughput values all fluctuate over time as the ABR protocol adjusts to changes in the viewer's internet connection speed and device capabilities. The bitrate line shows that the ABR protocol increases or decreases the bitrate to match the available throughput, while the buffer line shows that the ABR protocol adjusts the buffer size to help ensure smooth playback. The throughput line shows the actual throughput achieved by the viewer's device at each point in time.

Overall, this type of dynamic graph can help visualize how ABR works and how it adapts to changes in network conditions in real-time. It can also help identify potential issues with video streaming, such as periods of low throughput or buffer depletion, which can be useful for troubleshooting and optimization.

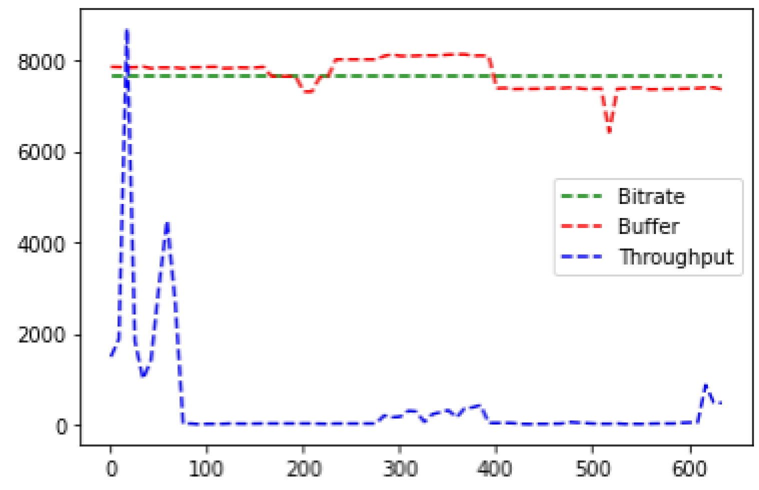


In the above graph, each point represents a single video segment, with the x-coordinate representing the time at which the segment was downloaded, the y-coordinate representing the size of the segment, and the color or size of the point representing the download time of the segment.

The graph shows how the download time and segment size vary over time for a video stream. The size of each point corresponds to the download time of the segment, with larger points indicating longer download times. The scatter plot shows that the download time tends to increase for larger segment sizes, as the larger segments take longer to download. However, there are also some smaller segments with longer download times, which could be due to network congestion or other factors.

Overall, this type of dynamic scatter plot can be useful for visualizing how video segments are downloaded over time and how the download time and segment size are related. It can also help identify potential issues with video streaming, such as periods of high download times or segments that are too large to download quickly, which can be useful for troubleshooting and optimization.

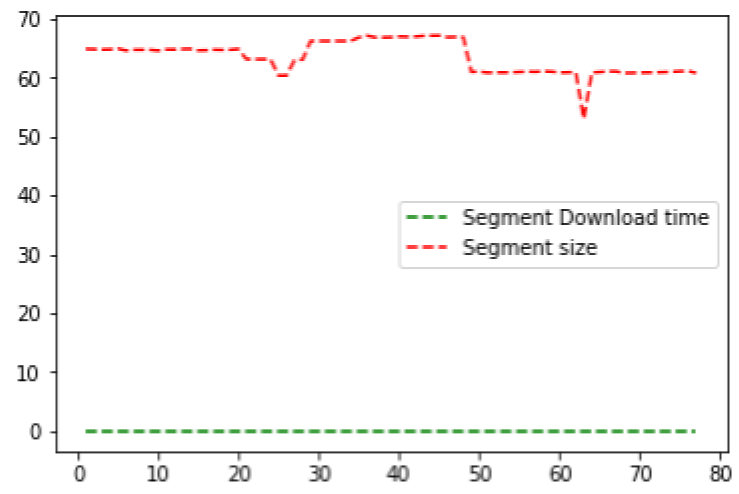
ABR:BOLA



In the above graph, the bitrate, buffer level, and time are plotted over time for a video stream using BOLA. The x-axis represents time in seconds, while the y-axis represents the value of each factor. The green line represents the bitrate, the red line represents the buffer level, and the blue line represents the throughput..

As you can see from the graph, the bitrate, buffer level, and throughput values all fluctuate over time as the BOLA protocol adjusts to changes in the viewer's buffer level and available network bandwidth. The bitrate line shows how the protocol increases or decreases the bitrate based on the viewer's buffer level. The buffer level line shows how the protocol adjusts the buffer level to prevent buffering or stuttering during playback. The time line shows how the video progresses over time.

Overall, this type of graph can help visualize how BOLA works and how it optimizes the video streaming experience based on the viewer's buffer level and available network bandwidth. It can also help identify potential issues with video streaming, such as periods of low buffer level or bitrate, which can be useful for troubleshooting and optimization.

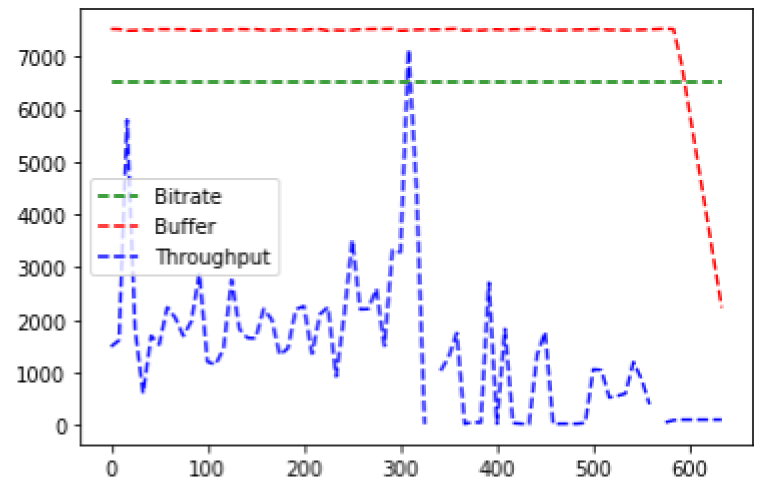


In the above graph, each point represents a single video segment, with the x-coordinate representing the size of the segment, the y-coordinate representing the download time of the segment, and the color or size of the point representing the buffer level.

The graph shows how the download time and segment size vary for a video stream using BOLA. The scatter plot shows that the download time tends to increase for larger segment sizes, as the larger segments take longer to download. However, the scatter plot also shows that the buffer level has an impact on the download time, with segments downloaded at higher bitrates when the buffer is full and at lower bitrates when the buffer is low.

Overall, this type of scatter plot can be useful for visualizing how video segments are downloaded over time using BOLA and how the download time, segment size, and buffer level are related. It can also help identify potential issues with video streaming, such as periods of high download times or segments that are too large to download quickly, which can be useful for troubleshooting and optimization.

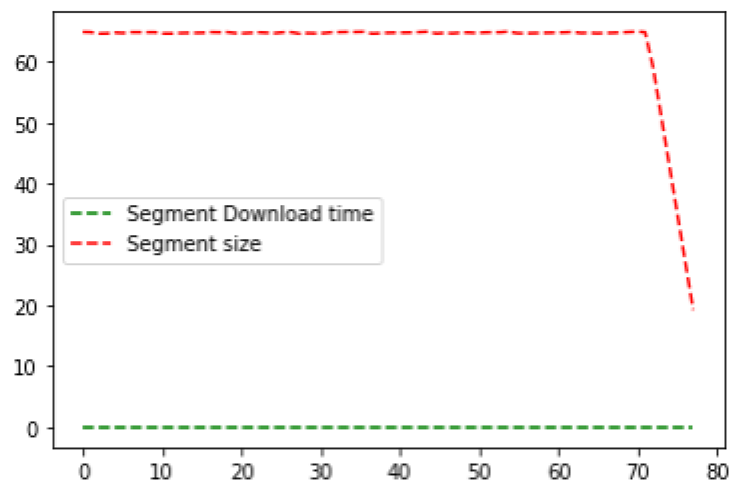
ABR:THROUGHPUT-BASED



In the above graph, the throughput, bitrate, and time are plotted over time for a video stream using a throughput-based approach. The x-axis represents time in seconds, while the y-axis represents the value of each factor. The blue line represents the estimated network throughput, the green line represents the bitrate, and the red line represents the buffer.

As you can see from the graph, the bitrate and estimated network throughput values both fluctuate over time as the streaming protocol adjusts to changes in the network conditions. The bitrate line shows how the protocol increases or decreases the bitrate based on the estimated network throughput. The throughput line shows the estimated network throughput over time. The time line shows how the video progresses over time.

Overall, this type of graph can help visualize how throughput-based streaming protocols adjust the video bitrate dynamically based on the estimated network throughput. It can also help identify potential issues with video streaming, such as periods of low network throughput or bitrate, which can be useful for troubleshooting and optimization.



In the above graph, each point represents a single video segment, with the x-coordinate representing the size of the segment, the y-coordinate representing the download time of the segment, and the color or size of the point representing the estimated network throughput.

The graph shows how the download time and segment size vary for a video stream using a throughput-based approach. The scatter plot shows that the download time tends to increase for larger segment sizes, as the larger segments take longer to download. However, the scatter plot also shows that the estimated network throughput has an impact on the download time, with segments downloaded more quickly when the estimated throughput is higher.

Overall, this type of scatter plot can be useful for visualizing how video segments are downloaded over time using a throughput-based approach and how the download time, segment size, and estimated network throughput are related. It can also help identify potential issues with video streaming, such as periods of low network throughput or segments that are too large to

Conclusion

Based on the analyses provided for the different modes of adaptive streaming, here is an overall conclusion for each type of graph:

ABR Dynamic Mode: In ABR dynamic mode, the focus is on bitrate, buffer, and throughput. By plotting these three factors on a line graph, we can see how the video player adjusts the bitrate and buffer size in response to changes in the estimated network throughput. This type of graph can be useful for visualizing how the streaming protocol responds to changes in network conditions and for identifying potential issues with video streaming, such as buffering or low bitrates.

BOLA Mode: In BOLA mode, the focus is on the buffer occupancy level, the buffer size, and the bitrate. By plotting these three factors on a line graph, we can see how the BOLA algorithm adjusts the bitrate based on the buffer occupancy level and the buffer size. This type of graph can be useful for visualizing how the BOLA algorithm determines the optimal bitrate for streaming based on the buffer occupancy level and for identifying potential issues with video streaming, such as buffer underruns or buffer overflows.

Throughput-Based Mode: In throughput-based mode, the focus is on throughput, bitrate, and time. By plotting these three factors on a line graph, we can see how the streaming protocol adjusts the bitrate dynamically based on the estimated network throughput over time. This type of graph can be useful for visualizing how the video player adjusts the bitrate based on changes in network conditions and for identifying potential issues with video streaming, such as periods of low network throughput or bitrate.

Segment Download Time and Size: By plotting segment download time and segment size on a scatter plot, we can see how video segments are downloaded over time and how the download time and segment size are related. This type of scatter plot can be useful for visualizing how video segments are downloaded using different adaptive streaming algorithms and for identifying potential issues with video streaming, such as large segments that take a long time to download or low network throughput that leads to slow download times.

Overall, these different types of graphs can help us understand how adaptive streaming protocols work, how they respond to changes in network conditions, and how they impact the quality of the video streaming experience.

REFERENCES

1. A. C. Begen, T. Akgul, and M. Baugher. Watching video over the Web, part I: streaming protocols. To appear in IEEE Internet Comput., 2011.
2. L. De Cicco and S. Mascolo. An Experimental Investigation of the Akamai Adaptive Video Streaming. In Proc. of USAB WIMA, 2010.
3. S. Deshpande. Adaptive timeline aware client controlled HTTP streaming. In Proc. of SPIE, 2009.
4. W. Feng, M. Liu, B. Krishnaswami, and A. Prabhudev. A priority-based technique for the best-effort delivery of stored video. In Proc. of MMCN, 1999.
5. R. Gao, C. Dovrolis, and E. Zegura. Avoiding oscillations due to intelligent route control systems. In Proc. of IEEE INFOCOM, 2006.
6. A. Goel, C. Krasic, and J. Walpole. Low-latency adaptive streaming over TCP. ACM TOMCCAP, 4(3):1–20, 2008.
7. P.-H. Hsiao, H. T. Kung, and K.-S. Tan. Video over TCP with receiver-based delay control. In Proc. of ACM NOSSDAV, 2001.
8. R. Kuschnig, I. Kofler, and H. Hellwagner. An evaluation of TCP-based rate-control algorithms for adaptive Internet streaming of H.264/SVC. In Proc. of ACM MMSys, 2010.
9. R. Kuschnig, I. Kofler, and H. Hellwagner. Improving Internet video streamilng performance by parallel TCP-based request-response streams. In Proc. of IEEE CCNC, 2010.
10. Pomelo LLC. Analysis of Netflix's security framework for 'Watch Instantly' service. Pomelo, LLC Tech Memo, 2009. http://pomelollc.files.wordpress.com/2009/04/pomelo-tech-report-netflix.pdf.
11. A. Orebaugh, G. Ramirez, J. Burke, and J. Beale. Wireshark and Ethereal network protocol analyzer toolkit. Syngress Media Inc, 2007.
12. M. Prangl, I. Kofler, and H. Hellwagner. Towards QoS improvements of TCP-based media delivery. In Proc. of ICNS, 2008.
13. L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. SIGCOMM CCR, 27(1):31–41, 1997.
14. S. Tullimas, T. Nguyen, R. Edgecomb, and S.-C. Cheung. Multimedia streaming using multiple TCP connections. ACM TOMCCAP, 4(2):1–20, 2008.
15. B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia streaming via TCP: An analytic performance study. ACM TOMCCAP, 4(2):1–22, 2008.
16. A. Zambelli. IIS smooth streaming technical overview. Microsoft Corporation, 2009. http://download.microsoft.com/download/4/2/4/4247C3AA-7105-4764-A