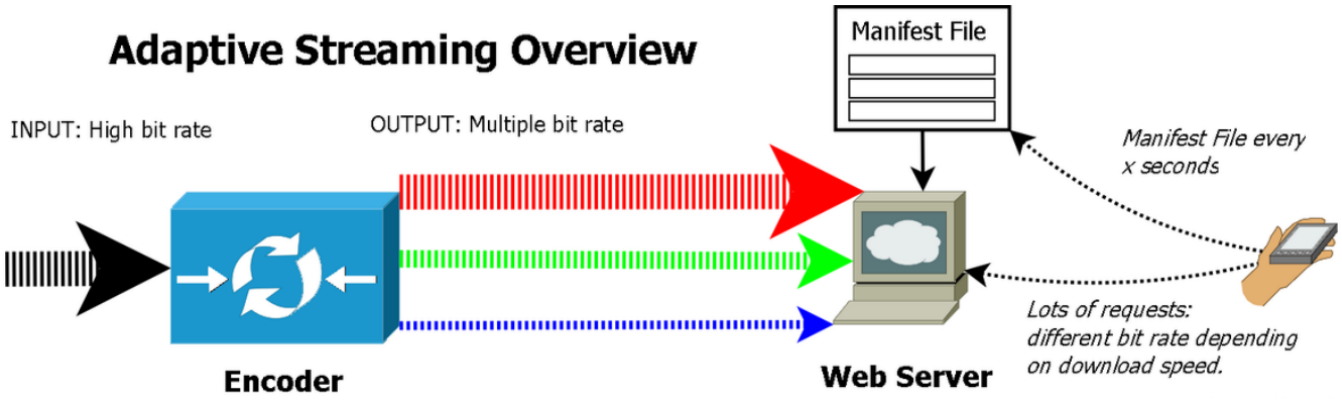
Adaptive Bitrate Streaming

This project aims to learn high-level thought and implement a demo of adaptive bitrate (ABR) algorithms for video streaming. We provide the video simulator, which will simulate video download and playback and continuously execute the user-written algorithm for bitrate decisions. We will first introduce the whole picture of video bitrate adaptation, then illustrate the task you need to finish and show a paper list of ABR algorithms. The algorithm you write will be tested over various simulated network environments and ultimately be given a final quality of experience(QoE) score.

0x01 Overview - What is adaptive bitrate streaming?



Adaptive bitrate streaming is a method for improving streaming over HTTP networks. The term “bitrate” refers to how quickly data travels across a network and is often used to describe an Internet connection’s speed. A high-speed connection is a high-bitrate connection. Streaming — or the process that makes watching videos online possible — consists of transmitting video files hosted on a remote server to a client. In streaming, videos are segmented into smaller clips so viewers do not need to wait for an entire video to load before they can begin watching it.

First, multiple versions of video files are created and encoded to fit a variety of network conditions. Then, based on factors like bandwidth and device type, the video player selects the highest-quality file that the device can play with the smallest amount of buffering possible. This allows playback to be as smooth as possible for end users worldwide, regardless of device or Internet speed.

Adaptive bitrate streaming works in the way of adapting the network. The video player gradually learns what video quality a connection can withstand. If the connection struggles to play a video segment, the player will switch to a smaller file with lower quality for the next segment. A viewer may experience some changes in quality, but the video will continue to play.

0x02 ABR Architecture - How does adaptive bitrate streaming work?

ABR starts with your raw data, and it is prepped before streaming. Video is transcoded and segmented into chunks. After this, the process is up to the playback device, which requests these chunks of data according to what it can handle given the available bandwidth. Let's look at these steps in more detail and some factors that might affect how they work.

Video Encoding and Transcoding

Encoding is the process by which raw video data is compressed and prepared for transport to a playback device. Transcoding is the process by which already compressed data is decompressed and decoded to alter it, often resulting in multiple versions of the original data. These changes are typical as follows:

- Resizing the video frame rate – or resolution – to accommodate different screens.
- Changing the bitrate of the decompressed file to accommodate different connection speeds. This can include changing the frame rate or the resolution.

Transrating is critical to adaptive bitrate streaming. After all, a playback device can't access data at a specific bitrate or resolution if it doesn't exist in that form.

Video Segmenting

You now have a collection of various sizes and resolutions for your video data that a playback device can access. But what if the playback device chooses incorrectly and the bitrate of the selected file is not ideal for the available bandwidth?

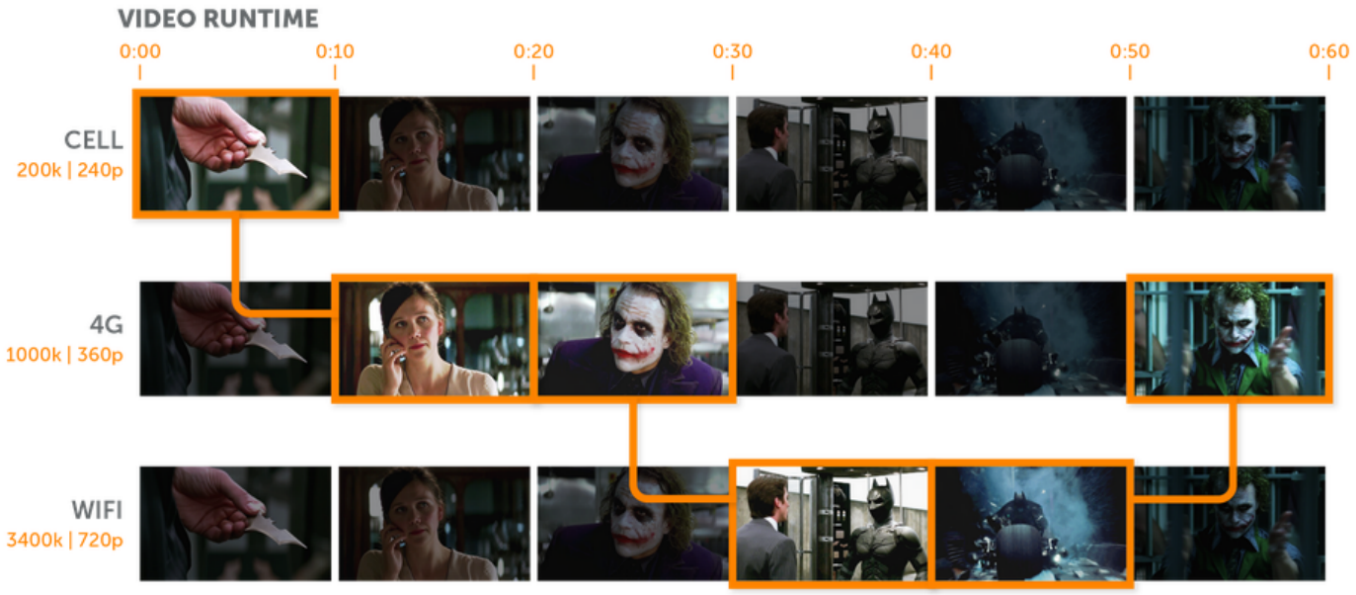
This is where segmenting comes in. Also known as chunked encoding, or chunking, this is the process by which streaming data is separated into a series of non-overlapping segments before being sent to the playback device. Each chunk typically ranges in length from 2 to 10 seconds. By breaking up the data this way, it's possible to adjust the size of data sent to a playback device mid-stream.

Initial Startup

Now that the data is fully prepared, the viewer's playback device takes the wheel. Before streaming begins, the playback device downloads a manifest that describes all the available chunks and bitrates. This is a menu from which the playback device can start streaming. Typically, a playback device will take it slow, selecting a bitrate it knows it can handle before adjusting.

Video Playback

After each segment, the playback device recalibrates and requests the next segment based on the new information. For example, the first segment was likely a much lower bitrate than necessary. The device will then request a higher bitrate for the next segment. If the bandwidth lessens or the device struggles to play a segment, it will adjust downward when requesting the next one.



0x03 Tasks

We provide several different tasks. You need to choose and finish one of them.

Task-1: (Complete it alone)

Read a paper of ABR algorithm, and implement the corresponding ABR algorithm according to the paper. Submit a 2 pages pdf report, including `Algorithm Analysis`, `Implementation`, and `Evaluation.`

Grading:

- Program implementation: 70%.
- Report score: 30%.

Task-2: (A group up to 3 people)

Design an ABR algorithm by yourself, write the code, and submit a 4 pages pdf report, including `Algorithm Design,` `Implementation,` and `Evaluation.` Please describe the `mathematical model / algorithms / neural network structure / dynamic programming transition function` or any other methods you use in your algorithm. You can use pictures, pseudocode, and formulas to illustrate them.

Grading:

- Program implementation: 50%.
- Report score: 30%.
- Algorithm performance: 20%. (The score of the effect is distributed to 12-20 points according to the ranking.)
  However, if your algorithms outperformed all the given examples, full marks will be awarded directly.

- An additional file, "contribution.pdf," should be attached to describe the contribution of each member in the group and arrange the names according to the weight of their contributions. This file will serve as a reference for grading each member.

ABR Paper List

The following is a list of the ABR-related paper, including but not limited to:

> 1. Adaptation algorithm for adaptive streaming over HTTP. (2012)
> 2. Towards agile and smooth video adaptation in dynamic HTTP streaming. (2012)
> 3. Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network. (2012)
> 4. Downton abbey without the hiccups: Buffer-based rate adaptation for HTTP video streaming. (2013)
> 5. Probe and adapt: Rate adaptation for HTTP video streaming at scale. (2014)
> 6. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. (2014)
> 7. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. (2015)
> 8. Machine learning based rate adaptation with elastic feature selection for HTTP-based streaming. (2015)
> 9. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. (2016)
> 10. Neural Adaptive Video Streaming with Pensieve. (2017)
> 11. Oboe: auto-tuning video ABR algorithms to network conditions. (2018)
> 12. Bit Rate Adaptation Using Linear Quadratic Optimization for Mobile Video Streaming. (2020)
>
> ...

0x04 Environment and Programming Entry

After writing an ABR algorithm it can be tested by first running `python studentComm.py` in one terminal window and then `python simulator.py <tracefile.txt> <manifestfile.json>` in another. `studentComm.py` is a simple communication layer between the simulator and the student example code inside of `studentcodeExample.py`. Test conditions can be manipulated by modifying the `tracefile.txt` and `manifestfile.json` files. Their specifications are detailed below.

Run the Example

For example, from the main directory, and run the following two command in two different terminal:

```
python studentComm.py
```

```
python simulator.py inputs/traceHD.txt inputs/manifestHD.json
```

Trace Files

The trace file dictates the bandwidth throughout a test run. On each line the 1st value represents the Video Time threshold where a bandwidth switch occurs and the 2nd value

represent the bandwidth value it will switch to. The first and second value must be separated by a single space.

A tracefile always must have a value for time 0. There can be as many or as few values as needed. The last bandwidth will describe the bandwidth until the test finishes.
For example:

```
0 1000000
10 5000000
20 2000000
30 800000
40 1000000
50 5000000
```

Manifest

The manifest file dictates other parameters such as chunks information, available bitrate, buffer size, and much more. rand_sizes.py can be used to quickly change chunk size information. It uses a normal distribution to randomly generate chunk sizes for a more natural test case.
Here is an example configuration for a Manifest file:

```
{
  "Video_Time": 60,
  "Chunk_Count": 30,
  "Chunk_Time": 2,
  "Buffer_Size": 4000000,
  "Available_Bitrates": [
    500000,
    1000000,
    5000000
  ],
  "Preferred_Bitrate": null,
  "Chunks": {
    "0" : [
62966,
125069,
567114
],

"1" : [
54274,
132844,
578807
],

"2" : [
68388,
116288,
540269
],
.
```

```
.
.
"29" : [
62442,
122537,
563240
]
  }
}
```

Grader

After writing an ABR algorithm in studentcodeEX.py, the grader can be ran using:
`python grader.py`
The grader will look for a directory named "tests" within the current directory. Within tests should be multiple directories, each representing a testcase. The name of a testcase directory is arbitrary and will correspond to the name of the test. Inside each testcase directory must be a manifest and trace file which will be run on the simulator.

The end result will output a grader.txt showing how well the ABR algorithm performed across the various test cases with a score as well as other metrics. More test cases can be added by following the same file structure of <test_name>/manifest.json and <test_name>/trace.txt

Please note that we will add more test cases when testing your final programs.

Files Specification

Below is the file tree of the given environment zip file of this homework:

```
├──Classes //python classes used in the simulator and grader
├──inputs //inputs files used for single use testing
├──papers //some paper references used for the ABR algorithms
├──tests //tests that grader will run
│   ├──testALThard //test that have a unstable bandwidth that confuses ABR algos
│   ├──testALTsoft //test that have a lot of alternating bandwidth
│   ├──testHD //test that have high quality bandwidth and other params
│   ├──testHDmanPQtrace //test that have high quality bandwidth but low params
│   ├──testPQ //test that have low quality bandwidth and param, will rebuffer.
│   └──...
├──grader.py //python file that graded the ABR algorithm via QOE
├──rand_sizes.py //python helper file use to generate chunk sizes
├──simulator.py //the simulator that generate parameters from text and json files
├──studentcodeExample.py //the file where that contains the student entrypoint
└──studentComm.py //the program the student will call to invoke their ABR algorithm
```

Entry

As aforementioned, `studentComm.py` is the test entry to example code inside of `studentcodeExample.py`.**You need to implement your own studentcode_119010001.py(replace with your ID), and change the entry in studentComm.py to your code.**

0x05 Submission Details

- **Due on: 23:59, 10 Nov. 2024**
- Submission gudeline:

  1.Please compress your own code files into **Assignment_{your id}.zip**.
  2. A report in format of PDF, named Report_{your_id}.pdf
- Please note that, teaching assistants may ask you to explain the meaning of your program, to ensure that the codes are indeed written by yourself. Please also note that we would check whether your program is too similar to your fellow students' code using plagiarism detectors for all assignments.
- The report should be submitted in the format of pdf, together with your source code, (and contribution.pdf for Task-2). Format mismatch would cause grade deduction. Violation against each format requirements will lead to 5 demerit points.
- **If you use third-party python library, please install them in conda environment, and remember to export your conda environment by using command `conda list -e > req.txt`. Please attach this environment file in the zip file.**
- We don't recommand to use other languages. However, if you still want to use language rather than python(C/C++/Java/Go, etc.), remember the entry is fixed and call your program from python (such as using ctypes library to call C function from Python). **Please make sure your program can execute directly and got result without any errors.** Also, remember to attach all of the source code, compile files, excutable files, and list all installed-library/installed-command you need.
- By default, your code may be used in scientific experiments or posted on the Internet. If you do not want your code to be used in the future, please illustrate it in your report and highlight it.
- **We take your honesty seriously. Please write your own code.**

0x06 References:

1. Sani Y, Mauthe A, Edwards C. Adaptive bitrate selection: A survey[J]. IEEE Communications Surveys & Tutorials, 2017, 19(4): 2985-3014.
2. Huang T Y, Johari R, McKeown N, et al. A buffer-based approach to rate adaptation: Evidence from a large video streaming service[C]//Proceedings of the 2014 ACM conference on SIGCOMM. 2014: 187-198.
3. Ayad I, Im Y, Keller E, et al. A practical evaluation of rate adaptation algorithms in http-based adaptive streaming[J]. Computer Networks, 2018, 133: 90-103.
4. Akhtar Z, Nam Y S, Govindan R, et al. Oboe: Auto-tuning video ABR algorithms to network conditions[C]//Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. 2018: 44-58.
5. Cloudflare, Inc. What is adaptive bitrate streaming?[2022] (https://www.cloudflare.com/learning/video/what-is-adaptive-bitrate-streaming/).
6. Sydney Whalen. Adaptive Bitrate Streaming: How It Works and Why It Matters (Update). [August 24, 2022] (https://www.wowza.com/blog/adaptive-bitrate-streaming)
7. Wikipedia. Adaptive bitrate streaming. [April 22, 2022] (https://en.wikipedia.org/wiki/Adaptive_bitrate_streaming)

8. Hong Xu. CSCI4430 - Computer Networks(CSE@CUHK). [May 4, 2022] (https://github.com/henryhxu/CSCI4430-ESTR4120)
9. Nathan Ankomah-Mensah, Zpeats. 50863_ABR_Lab. [May 6, 2020] (https://github.com/zpeats/50863_ABR_Lab)
10. Li Z, Bampis C, Novak J, et al. VMAF: The journey continues[J]. Netflix Technology Blog, 2018, 25(https://netflixtechblog.com/vmaf-the-journey-continues-44b51ee9ed12).
11. Tracy F, Peter S, Christopher C, et al. CMU 15-441/641 [Fall 2021] (https://computer-networks.github.io/fa20/)
12. Akhshabi S, Begen A C, Dovrolis C. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP[C]//Proceedings of the second annual ACM conference on Multimedia systems. 2011: 157-168.
13. Miller K, Quacchio E, Gennari G, et al. Adaptation algorithm for adaptive streaming over HTTP[C]//2012 19th international packet video workshop (PV). IEEE, 2012: 173-178.
14. Tian G, Liu Y. Towards agile and smooth video adaptation in dynamic HTTP streaming[C]//Proceedings of the 8th international conference on Emerging networking experiments and technologies. 2012: 109-120.
15. Liu C, Bouazizi I, Hannuksela M M, et al. Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network[J]. Signal Processing: Image Communication, 2012, 27(4): 288-311.
16. Huang T Y, Johari R, McKeown N. Downton abbey without the hiccups: Buffer-based rate adaptation for http video streaming[C]//Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking. 2013: 9-14.
17. Li Z, Zhu X, Gahm J, et al. Probe and adapt: Rate adaptation for HTTP video streaming at scale[J]. IEEE Journal on Selected Areas in Communications, 2014, 32(4): 719-733.
18. Yin X, Jindal A, Sekar V, et al. A control-theoretic approach for dynamic adaptive video streaming over HTTP[C]//Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. 2015: 325-338.
19. Chien Y L, Lin K C J, Chen M S. Machine learning based rate adaptation with elastic feature selection for HTTP-based streaming[C]//2015 IEEE International Conference on Multimedia and Expo (ICME). IEEE, 2015: 1-6.
20. Spiteri K, Urgaonkar R, Sitaraman R K. BOLA: Near-optimal bitrate adaptation for online videos[J]. IEEE/ACM Transactions on Networking, 2020, 28(4): 1698-1711.
21. Mao H, Netravali R, Alizadeh M. Neural adaptive video streaming with pensieve[C]//Proceedings of the conference of the ACM special interest group on data communication. 2017: 197-210.
22. Kang Y, Lim Y. Bit Rate Adaptation Using Linear Quadratic Optimization for Mobile Video Streaming[J]. Applied Sciences, 2020, 11(1): 99.
23. Huang T, Zhang R X, Zhou C, et al. QARC: Video quality aware rate control for real-time video streaming based on deep reinforcement learning[C]//Proceedings of the 26th ACM international conference on Multimedia. 2018: 1208-1216.