User's Manual

Dynamic Adaptive Streaming Educational Tool

Jacob Feldman, Lumbardh Halitjaha, Michael Buhrer and Rohan Chaudhari

Software Requirements

The simulator is tested and working with the latest version of dependencies on Linux only. Other operating systems have had compatibility concerns regarding VLC's integration with python.

2.0 Getting Started:

1) To get started you will first need to have Python 3 installed. You can download the latest Python 3 installer at https://www.python.org/download/releases/3.0/
2) The tkinter module is necessary for this software. Ensure it is properly installed for Python 3. You can find instructions to do this at https://tkdocs.com/tutorial/install.html#installmac.
3) The latest release of the vlc-python module is necessary. This can be found at https://pypi.org/project/python-vlc/#history. This module also requires the installation of the VLC media player software. Download and install the latest version at https://www.videolan.org/vlc/

2.1: Downloading and installing the Software:

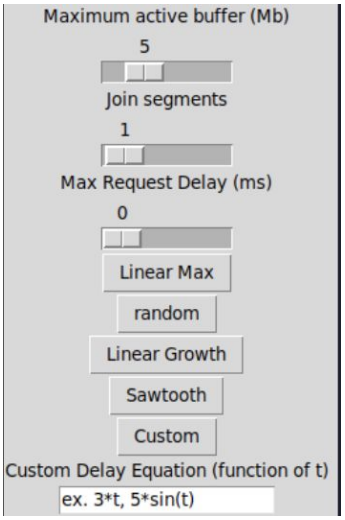
The software is available for download at https://github.com/LumiH/Adaptive-Informative-Real-Time-Streaming. To get this repo, open terminal and use the command "cd /[path to your repository]" to navigate to the directory where you would like to install the software. Once there, run "git clone https://github.com/LumiH/Adaptive-Informative-Real-Time-Streaming.git" to download the software.

2.2: Overview of the User Interface:

The user interface is broken down into four main sections.

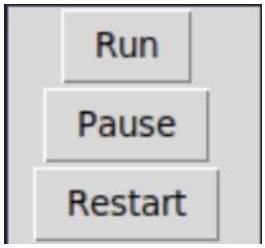
1) Parameter control— This section has four main components:
    a) Maximum active buffer: This is the size of your buffering window. The streaming player will not hold more than this amount of data at any one time.
    b) Join segments: This number corresponds to the number segments that must be buffered for playback to start.
    c) Max request delay: In order to simulate different bandwidth constraints the simulator uses a delay in milliseconds which times when incoming packets arrive. This delay is in addition to the round trip delay which already exists across your

connection. For this reason, if your connection is limited, it is recommended that this value remain low.

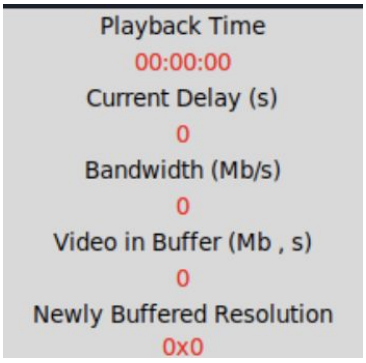


d) Delay types: clicking on one of these buttons changes the behavior of the delay over time.

2) Playback controls
   a) Run: This button starts playback
   b) Pause: This button freezes the current buffering state and stops playback.
   c) Reset: returns playback to the beginning and empties buffer



3) Live data— this section displays the current state of the stream quantitatively



a) Playback time: This shows the position of the current playback head in minutes:seconds:milliseconds. This value is calculated passively to avoid interrupting key functions.
b) Current delay: this is the delay which occurred upon requesting the last segment.
c) Bandwidth: The current connection bandwidth as calculated from the draw time of the most recently pulled segment.
d) Video in buffer: The size of the segments and length of video in the buffer.
e) Newly buffered resolution: The resolution of most recently buffered segment

4) Playback window— this part will display the video at normal playback speed.

3.0 Using the Simulator:

1) Starting the Simulator
   a) To start the simulator open a terminal and use the command "cd /[path to your repository]" to navigate to the directory containing the github repository.
   b) Once you are in the repository simply call "python3 stream.py". That's it!

2) Adjusting Parameters

   You can adjust the sliders and delay type described in the overview of the interface at any time. These changes will however not take effect until you press the "run" button. As a result, if you wish to alter parameters in the middle of a simulation you must first pause it and click "run" again.

3) Simulating Bandwidth Conditions

   a) Overview

   In order to simulate bandwidth conditions we choose to simplify the implementation by providing a simple delay mechanism which simply complimented the round trip delay time. This effectively simulates packets which are queued for long periods of time or drop packets with timeouts. There are five different delay functions which can be taken advantage of :

   1) Linear Max
   2) Random
   3) Linear Growth
   4) Sawtooth
   5) Custom

To select any one of these types you simply click on the type you would like to use. Your change is automatically saved so that the next time you click "run" the simulator will use the delay type you selected.

b) Delay type specifications

i) Linear Max

The linear max delay type incurs a constant delay equal to the value set using the "Max Request Delay" slider. *This is the default setting when starting the application.*

ii) Random

The random delay type incurs a random amount of delay between 0ms and the value set using the "Max Request Delay" slider.

iii) Linear Growth

The linear growth delay type incurs a delay which increases linearly as a function of playback time from 0ms to the value set using the "Max Request Delay" slider.

iv) Sawtooth

The sawtooth delay type incurs a delay which is either 0ms or the value set using the "Max Request Delay" slider at random.

v) Custom

The custom delay type allows the user to input their own delay function.

Important:

- Equations input into the input box must use t as the *only* variable
- All numericals must be entered as float values. Integers will cause the system to incur no delay
- All equations must use standard python syntax for multiplication, division, ect.
- The input box will accept sin, cos, and tan but not other trigonometric functions

Extra Important:

- t is a function of seconds. Meaning if you input a function like 3.0*t, by the end of a 60 second video you will be requesting a segment which will halt buffering for 180 seconds. This will make the window appear to be stalling

since tkinter cannot update during this time. As a result be sure to divide t by some reasonable factor, eg. (t/100) or (t/1000).

Good Examples:

(t/1000.0)**2.0 , sin(t/100.0)*3.0+1

Bad Examples:

3*t, 4.0xt ,arctan(t), 3.0*t+5.0*x

4.0 Implementing Different Streaming Algorithms:

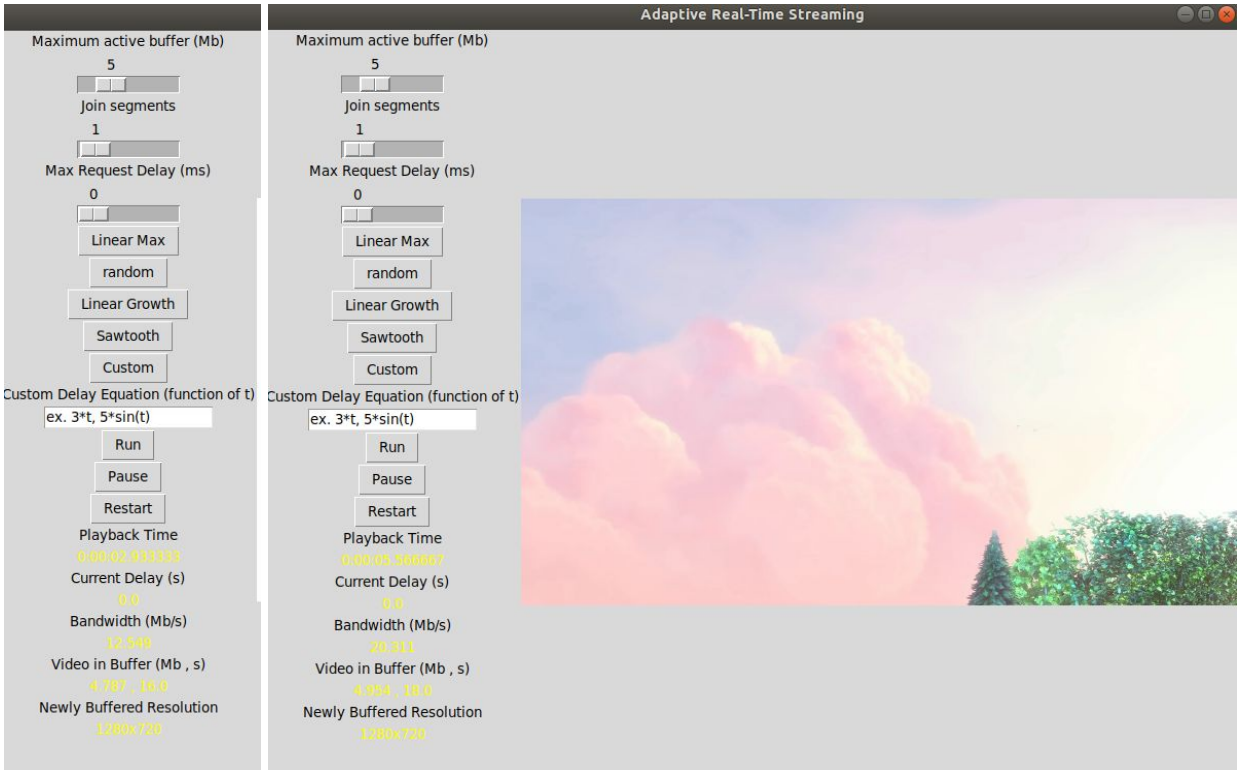
Part of the code can be modified in order to implement a different streaming algorithm into our software, allowing the user to observe the effects of their own streaming algorithm within our framework. Within the file “stream.py”, one can modify lines 200-229 to altogether change our default streaming algorithm implementation. The current solution relies most heavily on lines 221-222 where we predict what the bandwidth will be based on the size of the previous segment in bytes and the time it took to pull this segment. We are in essence measuring the users network bandwidth and assuming it will remain close to this in pulling the next segment. Therefore, to change prediction method simply remove these two lines and supplement another method for predicting bandwidth.

The lines past 222 make some basic assumptions about interfacing with the server such as assuming that the user prefers the highest resolution possible. By altering the for loop you can change prioritizing parameters to include more complex metrics such as type of connection. This could allow you to limit the bandwidth spent on a mobile stream for instance by suggesting a moving target for resolution.

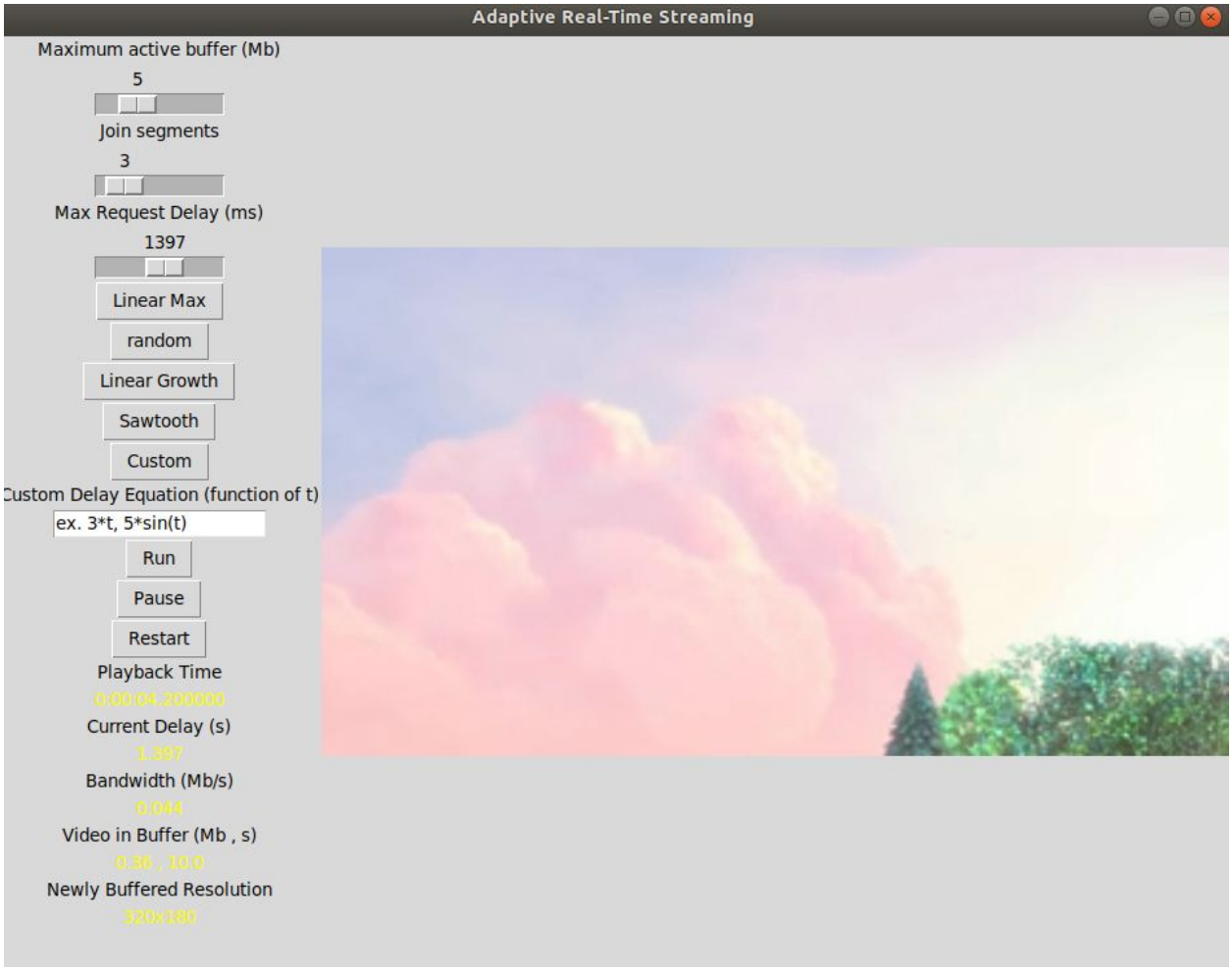
```
    else: #if we have already buffered one segment we can now use the algorithm to determine which resolution we should pull next
        last_buffered = to_play_buffer[-1]
        bandwidth = last_buffered['size']/last_buffered['time_to_pull'] #the availible bandwidth measured from pulling the last segment

        if curr_segment_num < total_segments:

            if representation_chosen!=True:

                for representation in reversed(mpd.representations): #going from highest res to lowest res see what the highest resolution we
                can pull is given our bandwidth
                        curr_seg_duration = representation._segment_duration
                        num_bytes = representation.segment_ranges[curr_segment_num][1]-representation.segment_ranges[curr_segment_num][0]

                        if bandwidth > (num_bytes/float(curr_seg_duration/1000)):
                            curr_representation = representation
                            representation_chosen=True
                            break

                if representation_chosen==False: #if we havent found a resolution which is in our bandwidth limits, select the lowest possible
                resolution
                    curr_representation=mpd.representations[0]
                    representation_chosen=True

            elif max_current_buffer - (bytes_in_buffer - played_segment_bytes) > num_bytes: #if we have found a resolution which matches our
            availible bandwidth pull the next segment at this resolution

                segment=get_segment(hostname,sock,representation,curr_segment_num,out,reslabel,customfield)
                to_play_buffer.append(segment)
                curr_segment_num+=1
                bytes_in_buffer+=segment['size']
                representation_chosen=False
```

5.0 Example:

Running the software with default parameters will allow one's full bandwidth to be utilized. By pressing run, the video will begin buffering and after 'Join segments' segments are buffered, the video will begin playback. Over time, one can see 'Playback Time' increase as the video plays. Additionally, the default current delay affecting one's bandwidth is visible in 'Current Dealy', while measured bandwidth is visible below this. The amount of buffered video content is also measured and visible in both size (Mb) and in time (s). Without throttling the bandwidth, as is default, one can notice that the amount of buffered content is quite large, buffering around 16 seconds of video in advance and being limited by the default artificial limit on buffer capacity of 5Mb. Underneath this, the high video resolution, 1280x720, informs the user that their buffered video is high quality, reinforcing the relationship between a high bandwidth and quality of content.



Next, we will increase both the ‘Max Request Delay’ and ‘Join segments’. Upon pressing ‘Run’, one will notice a period of buffering before streaming begins. This is caused from increasing the number of join segments, which requires a larger amount of video to be buffered before playback. This can directly affect the viewer’s quality of experience, as they become impatient while waiting for the video to play. Upon playing, one may note that the video quality is substantially worse than before, and this can be confirmed by looking at the ‘Newly Buffered Resolution’ statistic. By increasing the ‘Max Request Delay’, you have effectively increased the RTT of your connection with the server, resulting in a lower effective bandwidth and video quality.


Adaptive Real-Time Streaming
Maximum active buffer (Mb)
5
Join segments
3
Max Request Delay (ms)
1397
Linear Max
random
Linear Growth
Sawtooth
Custom
Custom Delay Equation (function of t)
ex. 3*t, 5*sin(t)
Run
Pause
Restart
Playback Time
0:00:04.200000
Current Delay (s)
1.397
Bandwidth (Mb/s)
0.044
Video in Buffer (Mb , s)
0.36 , 10.0
Newly Buffered Resolution
320x180