UNIVERSITY OF WATERLOO

Software Engineering

Mitigating User Interface Latency Caused by Event Accumulation in an Online Video Editor Built on a Client-Server Architecture

Google Inc.
Mountain View, CA

Prepared by
Michael Chang
Student ID: 20332377
User ID: m9chang
3A Software Engineering

December 15, 2012

Michael Chang
30 Doncrest Rd.
Richmond Hill, ON L4B 1A2

December 15, 2012

Dr. A. Morton, Director
Software Engineering
University of Waterloo
Waterloo, ON N2L 3G1

Dear Dr. A. Morton:

I have completed my fourth work term, following my 3A term. Please find enclosed my third work term report entitled: " Mitigating User Interface Latency Caused by Event Accumulation in an Online Video Editor Built on a Client-Server Architecture " for YouTube, LLC, a subsidiary of Google Inc.. My manager was B. Glickstein, and our team was primarily involved with building tools for video creators and content curators.

This report focuses on the handling of input events in the video enhancement tool on YouTube. This problem is similar to one that I encountered on my work term, but has been modified to protect proprietary design decisions.

I wish to thank B. Glickstein and R. Doshi for their advice on the technical content of this report. I also wish to thank C. Kleynhans and M. Yee for their advice on choosing a word processing environment to write this report, and W. Chang for proofreading this report. Finally, I wish to thank A. Mortezaei for providing feedback on the initial submission of this report.

I hereby confirm that I have received no help, other than what is mentioned above, in writing this report. I also confirm that this report is a resubmission of a report that was previously submitted for academic credit at this academic institution and graded as unacceptable.

Sincerely,

Michael Chang
Student ID: 20332377

Executive Summary

AudioSwap is one of the video enhancement features available to video creators on YouTube. It allows video creators to select a song from a library of over 100,000 songs that YouTube has licensed. The initial implemention of AudioSwap only allowed users to place the audio at the beginning of their video, and to use the song in its entirety. One of the major accomplishments of my work term was to implement user interface changes which allow video creators to position and trim tracks they've placed over their videos using AudioSwap.

This document describes issues which arose when these changes were first publicly deployed. It also describes some ways those issues could be mitigated, and uses the multi-criteria decision making methdology to rank these mitigation strategies.

Table of Contents

List of Figures

List of Tables

1 Introduction

YouTube provides web-based tools that allows content creators to enhance and edit their videos without installing additional software on their computer. One of these enhancement tools is AudioSwap, a tool which allows content creators to select a song that YouTube has licensed,[1] and use it to repluce the audio in their video.

The enhancement tools are generally built on a client-server architecture, with client-side JavaScript code loaded from the front-end server by the web-browser. The main components are the front-end server, the render server, and the database.[2] This split is illustrated in figure 1-1.

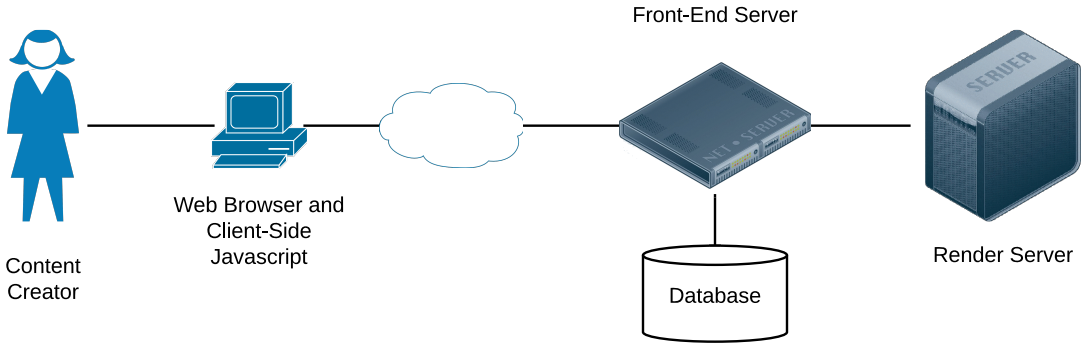

Figure 1-1: Network Diagram.

When a user modifies a video with the tool, the client-side code generates a computer-readable description of the changes made to the video called an edit list. The client-side code then sends the edit list to the preview server. The preview server uses the edit list and the user's original video (which has already been uploaded to YouTube) to generate a low-fidelity version of the video. It returns the location of the low-fidelity preview video to the client-side code, which then displays the preview video to the user.

The client-side code and front-end server communicate by serializing the changes made to the video into an edit list, using the JSON format. A sample edit list is shown in figure 1-2.

[1] YouTube places advertisements on the video, and uses the revenues to reimburse the creator of the audio track that the user selects.

[2] To protect proprietary design decisions, the server-side architecture has been simplified.

```
{
    "vc":[
        {
            "type":"video",
            "id":"jUhj_IKYx10",
            "start_ms":0,
            "end_ms":382882,
            "length_ms":382882,
            "effects":[ ]
        }
    ],
    "ac":[
        {
            "type":"synthetic-video",
            "id":null,
            "start_ms":0,
            "end_ms":15000,
            "length_ms":15000,
            "effects":[
                {
                    "id":"SILENT_AUDIO",
                    "parameters":{ }
                }
            ]
        },
        {
            "type":"audio",
            "id":"bcgo6zlW4X-Rge-ly6quJ-MtO-wEImhmJvVzUQg7JkMcNv6...",
            "start_ms":0,
            "end_ms":30000,
            "length_ms":211000,
            "effects":[ ]
        }
    ],
    "qid":1363,
    "preview_mode":"normal",
    "encrypted_project_id":"kT9c8hmpKFwzLnp9lVuVxdrpxTqzLkDH",
    "update":"1",
    "action_preview":1,
    "session_token":"7F6VaoLRFjaRGAfA7cST-uQqJ3J8MTM1NTUzOTYwM0..."
}
```


Figure 1-2: Sample Edit List.

1.1 Positionable/Trimmable Audio

Positionable/trimmable Audio is a new feature for the AudioSwap tool on YouTube. It allows the user to position and trim audio tracks on their video. For example, right before the proposal in a marriage proposal video, a content creator might choose to overlay the chorus of a romantic song, trimming the audio so that the proposal can be heard when it is made. The user interface for positioning and trimming audio is shown in figure 1-3.

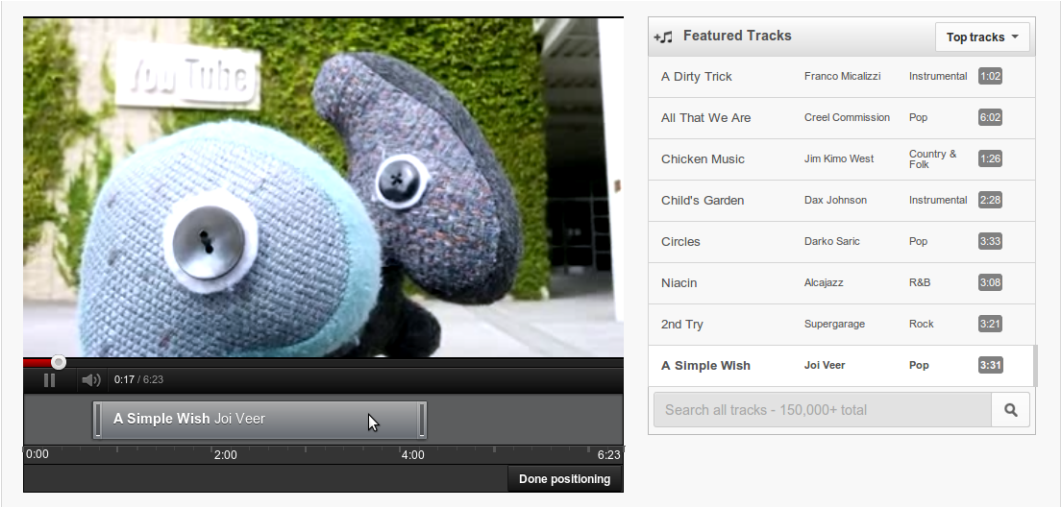

Figure 1-3: Positionable/Trimmable Audio - User Interface.

The user can click and drag an audio track, as shown in figure 1-4 and figure 1-5. They can also drag the trimming handles to trim the song, as shown in figure 1-6.

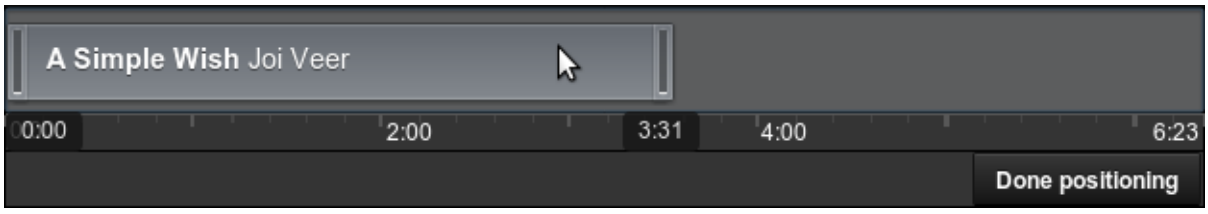

Figure 1-4: Positioning Audio (Fig 1 of 2).

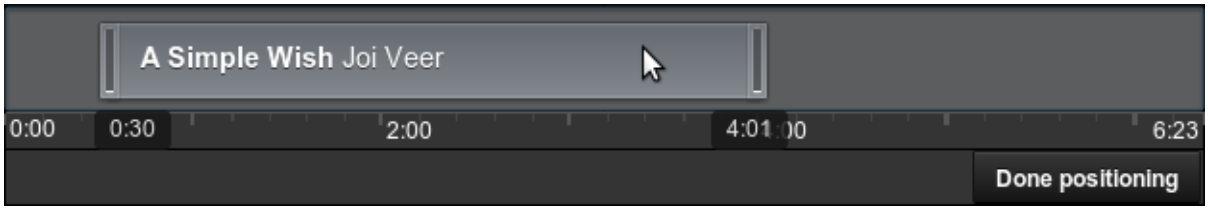

Figure 1-5: Positioning Audio (Fig 2 of 2).

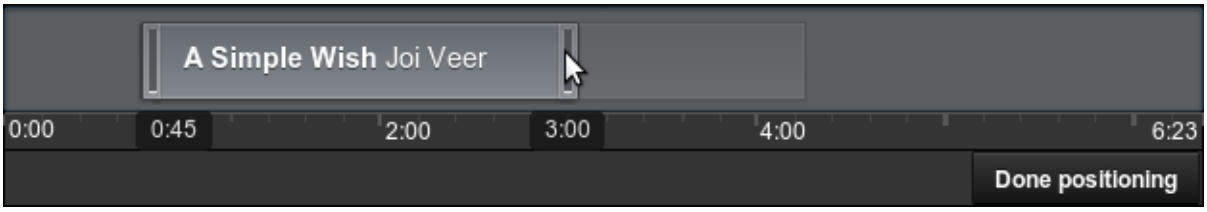

Figure 1-6: Trimming Audio.

1.2 Problem Statement

When positionable/trimmable audio was first released to content creators, some content creators noticed a delay between releasing the mouse when positioning or trimming an audio track and when the preview video would start playing.

Whenever the user positions or trims an audio track, the edits are serialized in an edit list and sent to the front-end server. The front-end server sends a message to tell the render server to render a low-quality preview of the video, before returning the location of the preview video to the front-end code. This flow is illustrated in the swimlane activity diagram in figure 1-7.

A large number of UI events may be generated when the user positions or trims an audio clip on the video enhancements page. An edit list is generated for each UI event, with a round-trip to the server and back to process each event. This worked fine in development, when each round-trip was virtually instantaneous. However, this round-trip is estimated to take 600ms for a typical content creator.[3]. When a large number of events occur, the user experiences a large delay before seeing a preview.

This report seeks to select an appropate mitigation strategy for the delay in the user interface for positionable/trimmable audio.

[3]The calculation to determine this figure is detailed in appendix A.

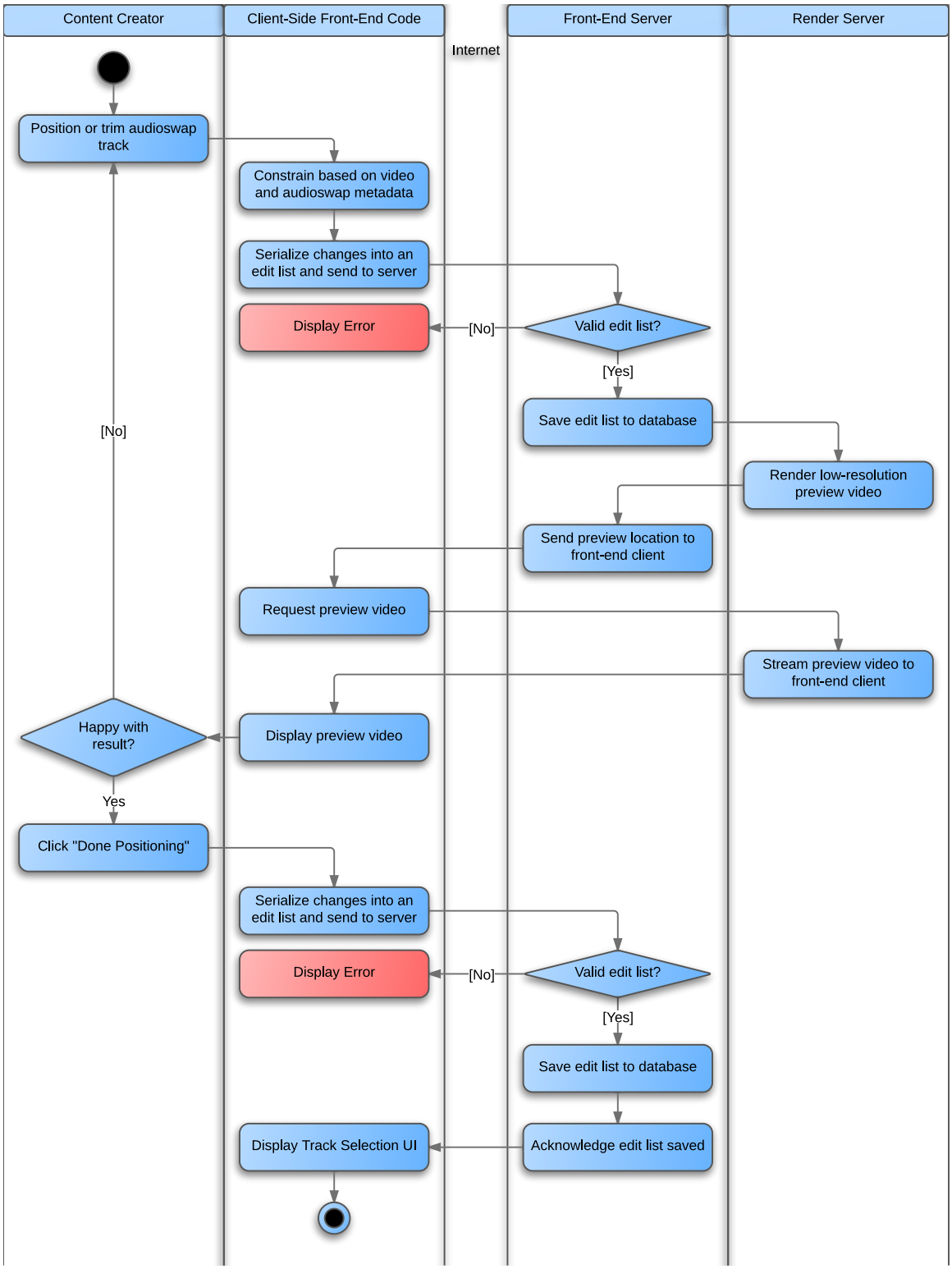

Figure 1-7: Swimlane Activity Diagram.

2 Analysis

First, a simulation is run to further understand the problem. Then, we develop criteria and outline various possible mitigation strategies. We use the weighted sum model,[4] a method of multi-criteria decision making, to select an alternative. Then, we run further simulations to tweak the parameters of the chosen alternative.

2.1 Initial Simulation and Assumptions

In order to protect proprietary design decisions, it is assumed that a user interaction generates 150 events over five seconds at a constant rate of 30 events per second. It is also assumed that requests will be processed one at a time.[5]

A simulation of the request queue length for the first 100 seconds after the start of the interaction is shown in figure 2-1. Note that the user does not see the preview until the request queue length drops to zero, almost 90 seconds after they've stopped adjusting the control.

2.2 Criteria

Criteria were selected by the team lead, based on past experience and overall organization priorities.

2.2.1 Developer Effort

How much work would a developer need to do to complete this alternative? (Ranked from 1 (more than one quarter) to 5 (less than one week).)

[4]This method is covered in MSCI 261, a core course in the software engineering curriculum at the University of Waterloo. If you need a refresher, you can read about it on Wikipedia at http://en.wikipedia.org/wiki/Weighted_sum_model.

[5]According to [1], HTTP user agents should limit themselves to two concurrent requests. It can be seen in [2] that many browsers use more (say, four or six). However, this is omitted from the simulations for simplicity.

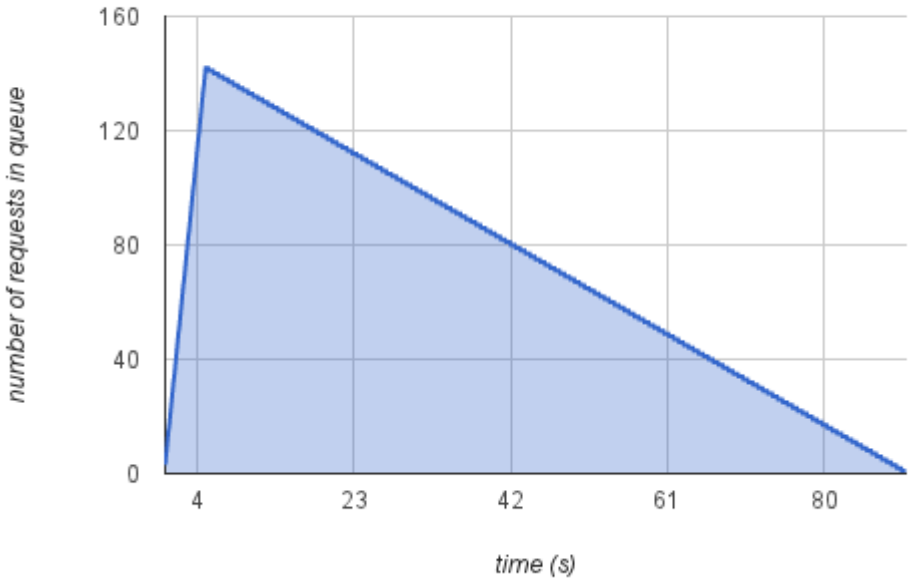

Figure 2-1: Simulated Request Queue Length (Original Implementation).

2.2.2 PR Impact

Could we write about the alternative to generate additional interest in the video enhancement tools? Will pursing an alternative cause others to write negative articles about the video enhacement tools? (Ranked from 1 (negative impact) to 5 (positive impact).)

2.2.3 External Dependencies

Does this alternative depend on deliverables from outside the team? How likely will those deliverables be ready in a timely manner? (Ranked from 1 (dependencies outside Google) to 5 (no external dependencies).)

2.2.4 User Impact

How do we think the user experience will be impacted by this alternative? (Ranked from 1 (negative impact) to 5 (positive impact).)

2.2.5 Interest

Is there someone on the team who is supportive of a particular approach and willing to do the work to get it done? (Ranked from 1 (no interest) to 5 (multiple supporters).)

2.3 Alternatives

Possible solutions were generated through brainstorming.

2.3.1 Client-side Audio Syncing

Synchronize audioswap track and original video playback on the client side, removing the need for a roundtrip to display the preview for audioswap.

2.3.2 Do Nothing

Prioritize other fixes / features instead.

2.3.3 Real-Time Streaming Preview (WebRTC)

Replace the video preview player with WebRTC so that we can stream the preview in real time.

2.3.4 Throttle Events

Don't send an edit list to the server for every event. Instead, send one every Δt seconds, as well as one for the "last" event (mouse button up event).

2.3.5 Wait for Advances in Networking Technology

Limit the use of the application to users on super-high-speed network connections, like Google Fiber.

2.4 Phase 1: Multi-Criteria Decision Making

The results of table 2-1 indicate that event throttling is the best solution. We note that the the solution chosen isn't very robust if the criteria weights are changed – in fact, it is possible to make virtually every solution an "optimal" solution by changing the weights. However, we proceed with the weights chosen by the team lead, noting that we should re-evaluate the criteria weights if we need to make a similar decision in the future.

Table 2-1: Multi-Criteria Decision Making Results.

Criteria	**Wt.**	**Client**		**Nothing**		**WebRTC**		**Throttle**		**Wait**	
Developer Effort	2	2	4	5	10	1	2	4	8	5	10
PR Impact	2	3	6	2	4	5	10	3	6	1	2
External Dependencies	3	2	6	5	15	1	3	5	15	1	3
User Impact	2	2	4	3	6	5	10	4	8	1	2
Interest	1	5	5	1	1	3	3	5	5	1	1
Total			**25**		**36**		**28**		**42**		**18**

2.5 Phase 2: Simulation and Optimization

A number of simulations was performed to determine the best value of Δt, the duration between edit lists being sent to the server. A sample of these simulations is shown below in figure 2-2, figure 2-3, and figure 2-4. Of the simulations performed, the value $\Delta t = 600ms$ provided the most frequent preview updates to the user while maintaining a bounded request queue size.

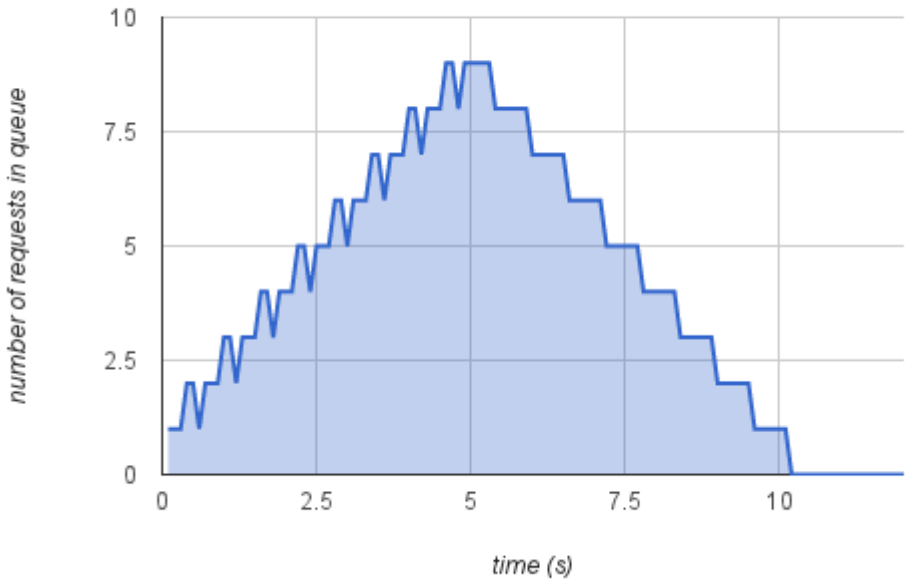

Figure 2-2: Simulated Request Queue Length (300ms delay).

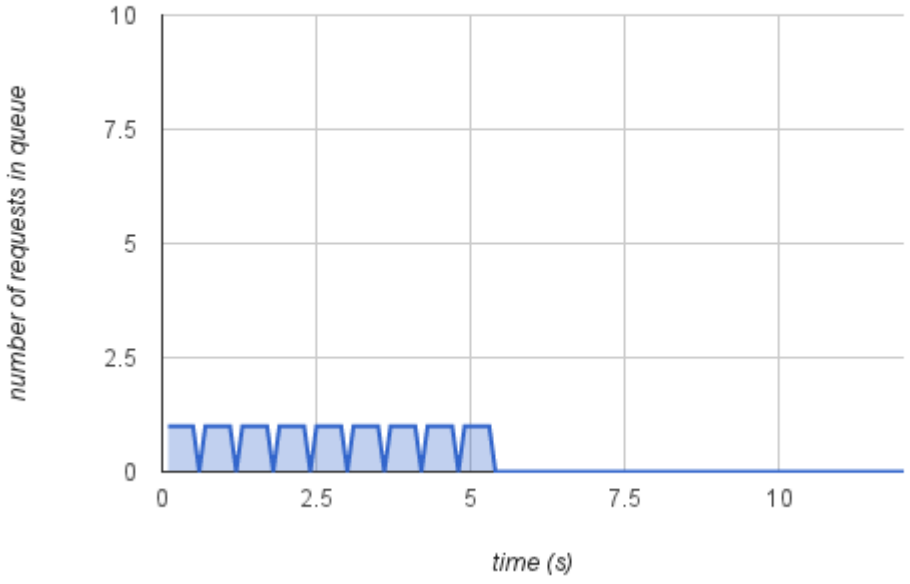

Figure 2-3: Simulated Request Queue Length (600ms delay).

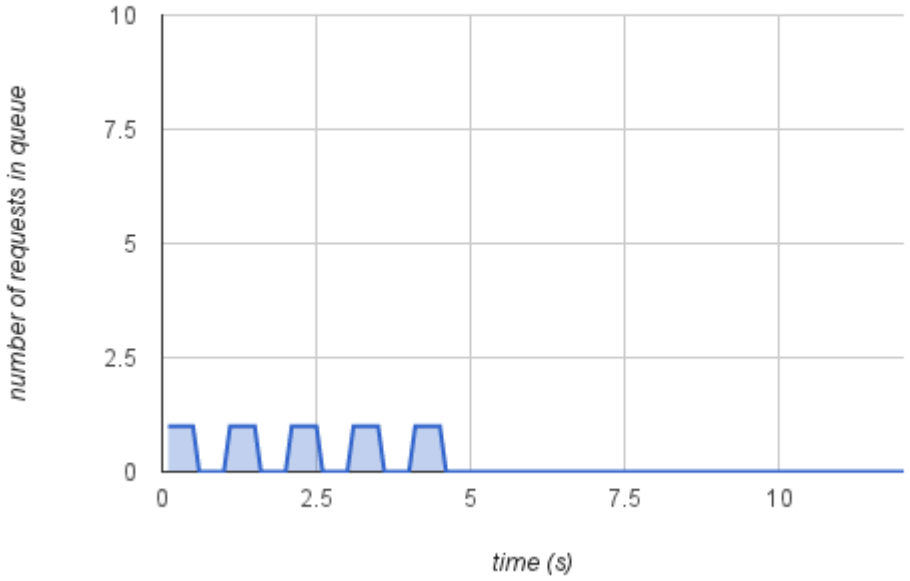

Figure 2-4: Simulated Request Queue Length (1000ms delay).

3 Conclusions

The simulation suggests that edit lists should be sent to the front-end server at most once every 600ms. We note that this value corresponds to the round-trip time we determined in §2.1, so we recommend that an edit list be generated only after the previous one has completed its round trip,[6] and that UI events only update the UI in the interim.

This provides a bound on the size of the number of requests in the queue. (Only one request will be processing at any given time.) This, in turn, gives us a bound on the time between a given user interaction and when the preview is displayed to the user. (This also reduces the total number of preview requests sent to the preview server, which results in a corresponding reduction in CPU and bandwidth usage on the render server.)

[6] With a timeout, of course, and re-transmission with exponential back-off in the event of packet loss / transient network failure, etc...

4 Recommendations

We recommend that the client-side code be modified so that UI events update the UI immediately, but only trigger a network round-trip if an edit list is not already in flight.

We note that this recommendation does not necessarily preclude the pursuit of the other alternatives. Instead, the above alternative should be pursued first, since it provides the best overall trade-off given the criteria. Pursuit of the remaining alternative solutions should then be prioritized within the overall context of the team's objectives and key results (OKRs) for the quarter.

References

[1] R. T. Felding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, http://www.w3.org/Protocols/rfc2616/rfc2616.html, 1999.

[2] B. Sayre, B. Zbarsky, G.Kanai, M. Shaver, M. Beltzner, T. Mielczarek, E. Jung, S. Pasche, R. O'Callahan, B. Elch, M. Hommey, S. Parkinson, K. Hamilton, Y. Goolam-abbas, C. Hofmann, S. Wilsher, and F. Schroeder, *Bug 423377 - (wedidntstartthefire) Change max-persistent-connections-per-server to 6.* https://bugzilla.mozilla.org/show_bug.cgi?id=423377, 2008.

Appendix A Calculating Simulated Server Response Time

The simulated server response time is comprised by summing multiple factors, as noted below.

A.1 Round Trip Time

Round-trip travel times between three locations in North America and youtube.com were sampled.

Table A-1: Round Trip Time.

Location	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Mean**	**Sample StDev**
Richmond Hill, Ontario	7.77	7.16	6.85	7.44	7.305	0.393
Waterloo, Ontario	6.16	6.09	6.28	6.12	6.163	0.083
Fremont, California	1.18	1.30	1.18	1.28	1.235	0.064
Average					4.901	2.759

A.2 Encoding Time

To protect proprietary design decisions, assume it takes 2000 ms to encode 1000 s (= 16 minutes) of audio. (Video encoding time is ignored; since the control only alters the audio track, we assume that existing video encodes can be reused and multiplexed into the result upon being sent to the client.) We take the average of the durations of top featured tracks in the audio enhancement page to get an approximate audio duration of 4 minutes, 53.125 seconds. Based on our assumption, this would take 587 ms to encode.

A.3 Wi-Fi

The use of a wireless network standard such as 802.11n may introduce additional latency in the HTTP request. The amount of this latency varies depending on many factors, and could be the subject of its own report. It is assumed that the user is using a wired connection for the purposes of this report. It should be noted that the final implementation may need further adjustment to compensate for the extra latency that users of these networks will encounter.

A.4 Other Factors

It is assumed that other tasks like multiplexing the audio and video streams on the preview server, and server-side parsing of the edit list are almost negligible. The duration of these tasks should be dwarfed by the time taken to encode the audio on the preview server. We add 8 ms of latency to account for these other factors.