Exploring "Downward Spiral" Effect for Video Streaming Rate Selection

Harsh Trivedi, Malvika Modi, Harshwardhan Agarwal

Abstract

This project is mainly adopted from the ideas proposed in [1]. According to the authors, in video streaming services over HTTP, video rate selection algorithms have to face vicious feedback loop which suddenly plummets the perceived bandwidth at client-side and hence the video quality selected. We perform a detailed analysis of this effect which authors call "Downward Spiral". We reproduce and verify this effect with our video streaming client / server following the guidelines of how typical HTTP Client / Server works. We perform several controlled experiments to verify multi-step cause of the this effect on mininet environment. With our experiments, we show that three suggested heuristics by authors to mitigate this effect actually help. Based on our observations, we provide our own change to video client, which completely removes this effect. We justify how our improvement is closest to the ideal scenario and significantly improve video quality without any adverse effects on viewer experience whatsoever. Finally, we also verify that "downward spiral" happens in real environments with test on Vimeo streaming service. Code with instructions to reproduce all the experiments is available at `https://github.com/HarshTrivedi/FCN-VideoStreamingProject`

1 Introduction

For good quality video streaming, the streaming rate (video quality) must be dynamically adapted to provide the best user experience to the viewer. If streaming rate is too high, it will cause the playback buffer to get empty frequently leading to many annoying re-buffering events. If streaming rate is too low, then naturally the viewer will experience low quality video which is definitely not good.

Hence, for good viewer experience, we want:

1. playback rate (video quality) to be high throughout the video play.
2. least re-buffering events during the video play.

Therefore the rate selection must be done based on accurate estimation of available bandwidth.

Generally, the videos of these services are hosted on standard HTTP servers in CDNs. Hence, the video streaming rate selection must be done on client side. Clients decide rate based on perceived available bandwidth. However, in presence of competing flow, rate selection algorithms have to face a weird consequence which authors call "Downward Spiral". In presence of competing flow, client perceives inaccurate bandwidth which further causes a vicious feedback loop continuously reducing the perceived bandwidth making the client to choose lower and lower playback rate (video quality). Finally, client ends up playing at much lower rate (quality) then the rate sanctioned by its fair share.

Figure (1) depicts this effect[1]. Video streaming is happening in presence of 5 Mbps bandwidth. As soon as the competing flow starts, the playback rate suddenly plummets down to the lowest possibility in spite of fair share being 2.5 Mbps.

In this project, we do a detailed investigation of this effect in controlled environment using mininet [2] and also verify its existence on real environment on Vimeo streaming service. First, we reproduce this effect with our video streaming client / server following the guidelines of how typical HTTP Client / Server works. We perform several controlled

[1] **Except figure (1), all other plots in this report are generated by our emulations and experiments**

Preprint. Work in progress.

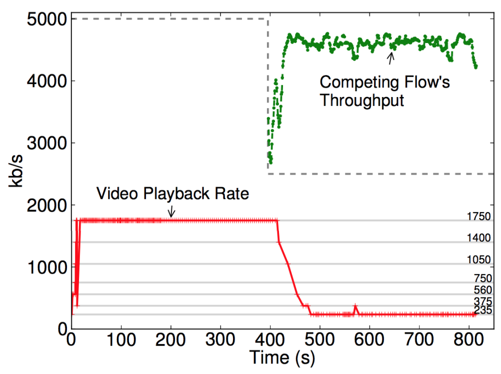

Figure 1: Downward Spiral

experiments to verify multi-step cause of the this effect on mininet environment. With our experiments, we show that three suggested heuristics of authors to mitigate this effect actually help. Based on our observations, we provide our own change to rate selection, which completely removes this effect. We justify how our improvement is closest to the ideal scenario completely recovering fair share behavior and significantly improving video quality without any side-effects. Finally, we also verify that "downward spiral" happens in real environments with test on Vimeo video service.

The rest of this paper is organized as follows. **Section 2** discusses all our experiments in controlled environment (mininet emulation). In **2.1** we discuss the detailed methodology, in **2.2** we explain downward spiral and multi-step causes with our emulations and experiments. In **2.3** we show the motivations and results of author suggested changes on client side algorithm. In **2.4** we discuss motivation and results of our improvement **(extra 30%)** and discuss experiments to show that it completely removes downward spiral and it's closest to ideal scenario that we can get. **Section 3** discusses methodology and results of our experiments on real environment. And **section 4** concludes with summary of our work.

We mark **all statements in bold** which we think we could not explain and convey properly during the presentation time. Last, **section 5** summarizes all the clarifications we want to make.

2 Emulation Experiments

2.1 Emulation Methodology

2.1.1 Typical (Our) Video Client / Server:

Before going into controlled environment setup, we first explain how a typical video client works over HTTP[2]. Figure (2) pictorially explains a typical video streaming client and server over HTTP at high level.

There is a persistent TCP connection between Client and Server. Client has access to a fixed sized playback buffer. If buffer doesn't have enough space it lets playback buffer drain. If it has space, it get the current estimate of throughput and uses that to looks-up what rate it should use. Since video service only provides discrete rates (video qualities), this look-up is generally based on a range. Eg. If estimated b/w available is in 2.0 to 2.5 Mbps, rate selected could be 1.8 Mbps.

Based on the rate, it decides how many bytes it requires for that segment and asks for that many bytes from server and receives them. Based on this request and response it estimates current throughput (b/w) so it can be used next time. A viewer in background constantly keeps draining the playback buffer at rate of 1 second per second.

The range look-up table as shown on right-hand-side of figure (2), was adopted from the paper [1] in which they show this tables characterizes Netflix's video client according to their black-box testing.

We implement the video client and server using the above explained guideline and use them over a controlled environment (mininet topology) for various experiments.

[2]It is not explicitly given in paper so we had to infer things from bits and pieces of information and some trial/error

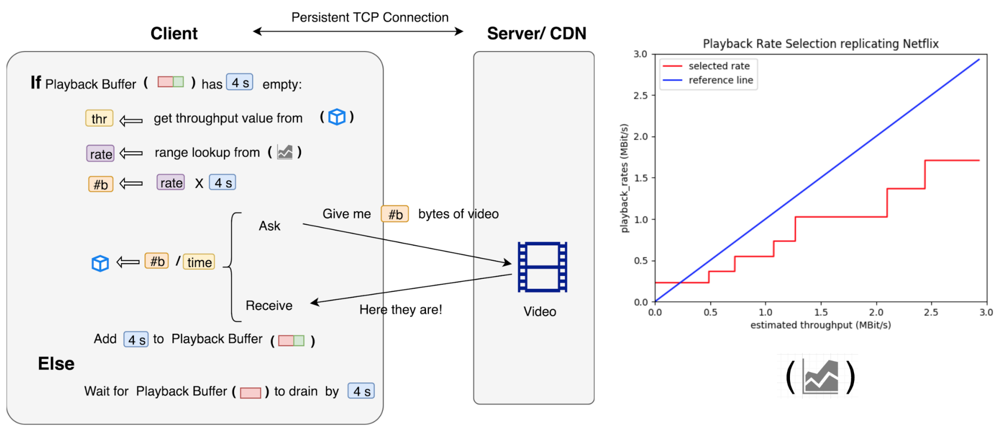
Persistent TCP Connection
Client
Server/ CDN
If Playback Buffer ( ) has 4 s empty:
thr ⟸ get throughput value from ( )
rate ⟸ range lookup from ( )
#b ⟸ rate X 4 s
Ask
Give me #b bytes of video
⟸ #b / time
Receive
Here they are!
Video
Add 4 s to Playback Buffer ( )
Else
Wait for Playback Buffer ( ) to drain by 4 s
Playback Rate Selection replicating Netflix
selected rate
reference line
playback_rates (MBit/s)
estimated throughput (MBit/s)
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Figure 2: Typical HTTP Client for video streaming

2.1.2 Emulation Topology:

We perform our experiments in a controlled environment using mininet. Figure (3) summarizes the experiment topology. Our video client is run on video-client node, video server is run on server node and competing flow is generated by iperf utility server and client. All the parameters as **(1)** Playback Buffer, **(2)** Bottleneck Bandwidth, **(3)** Bottleneck Buffer Queue size and **(4)** Appropriate Start times need to be calibrated to see downward spiral effect. Since many details are not explicitly given in paper, we had to do trial and error and calibrate to see this effect properly.

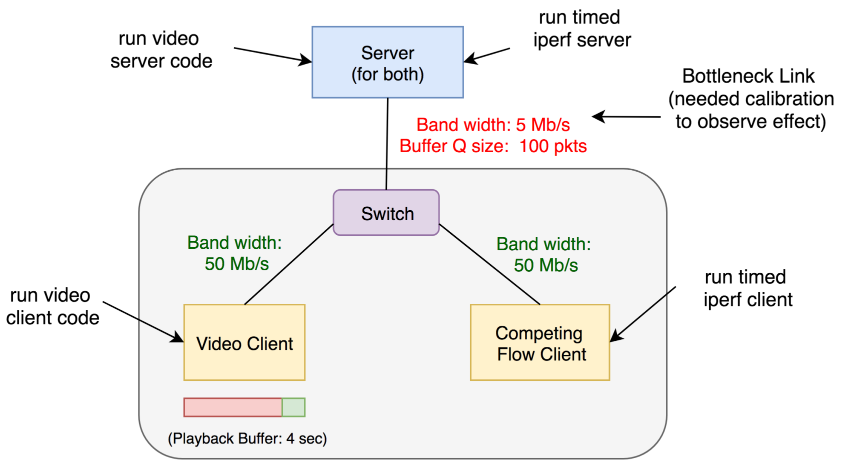

Figure 3: Mininet - Emulation Topology

2.2 Downward Spiral - Our Take

In this sub-section, we explain in-depth multi-step causes and effects of "Downward Spiral" using our emulations. Note that some of these plots are not present in the paper as well.

2.2.1 Downward Spiral:

In figure (4), video streaming is happening in presence of 5 Mbps bandwidth. In spite of fair share being 2.5 Mbps, as soon as the competing flow starts, the playback rate suddenly plummets down to the lowest possibility and is not able to recover until the competing flow ends. In figure (4), video throughput (magenta line) is computed throughput by measuring bytes transferred every second independent of On or Off phase. However, it is a bad estimate of available b/w because client is only waiting in OFF phase, so that time should be ignored. Hence video b/w estimates (red line) are computed by measuring bytes transferred during the ON phase. As shown in figure (2), video client ignores the pause times (times waited to let buffer drain) to have good estimate of available bandwidth.

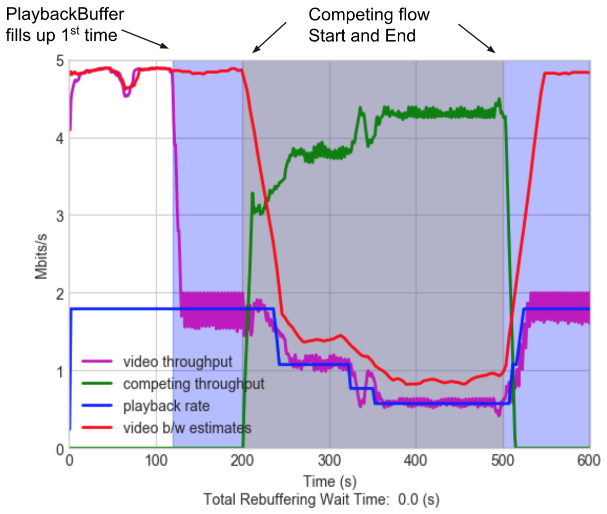

Figure 4: Downward Spiral our Emulation

2.2.2 Effect of Playback Buffer Fill-up - On-Off Sequence:

It turns out that when the playback buffer fills up for the first time, the video client goes in a ON-OFF sequence (figure 5). In ON phase it asks for the video segments at selected playback rate and in OFF phase it waits for playback buffer to drain. Immediate consequence of this is that the interval between my requests increases because now we also have to pause for playback buffer to drain (figure 5).

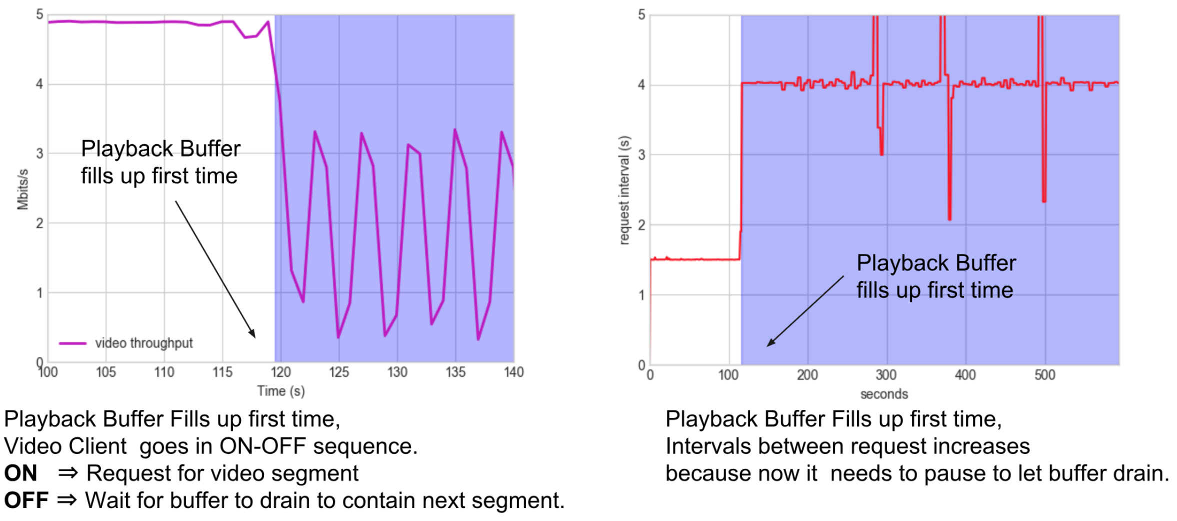

Figure 5: Effect of Playback Buffer Fill-up

2.2.3 Effect of ON-OFF Sequence on Congestion Window:

Now that we know that video flow goes in ON-OFF sequence, we can do a more controlled experiment to see what is happening. So in this experiment (6), we ask for a video segment, pause, and repeat this 3 times explicitly with video flow in presence of competing flow and observe the effects on throughputs of video and competing flow and the congestion windows of video flow.

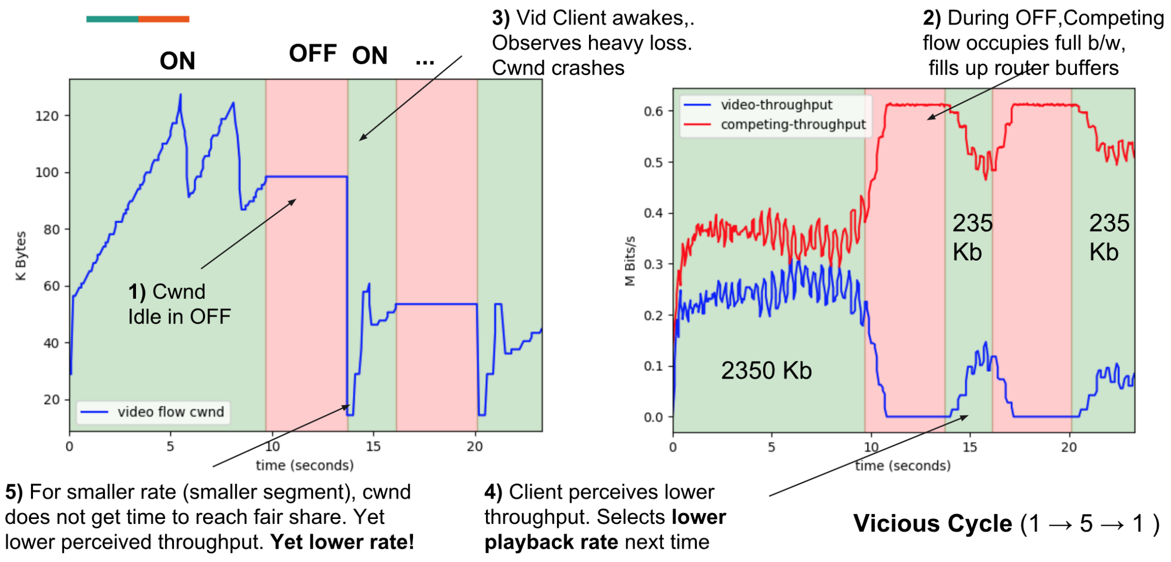

Figure 6: Feedback Loop

1. When client goes in OFF mode, the Congestion window becomes idle.
2. During this period competing flow takes away the full bandwidth and as a result fills up the router (bottleneck link) buffers.
3. Hence, when client comes back to ON mode, it observes significant loss, congestion window crashes and goes to slow start.
4. Because of starting from slow start, next time it perceives lower throughput and hence selects a lower rate.
5. Now lower rate means next time lesser number of bytes will be requested. Hence, congestion window won't get enough chance to raise to fair share from slow start causing an even lower throughput estimate next time. So even lower playback rate is selected next time.
6. It goes into a vicious cycle till the rate plummets down to lowest possibility.

Note that the ON-OFF sequence is leading to **two key effects**:

1. Congestion window crashing and causing video flow to go in slow start at beginning of every ON phase.
2. Smaller picked rate gives less chance for congestion window to raise to fair share from slow start.

Note that the cause of **both** these problems is the On-Off sequence. We will later show as part of our improvement that how **explicitly avoiding On-Off sequence eliminates both these effects completely**.

2.2.4 When will Downward Spiral Occur?:

The downward spiral effect **will not always occur** even if there is a common bottleneck router link shared among the flows. We ourselves had to calibrate the topology parameters by trial and error to see this effect since many things were not given in paper. However, it is generally **very likely to occur** as explained below.

Playback buffer filling up guarantees the On-Off sequence. The On-OFF sequence although necessary condition is not sufficient to guarantee downward spiral effect. The OFF phase should be long enough for competing flow to fill-up the common bottleneck router link's buffer. However, a typical OFF phase period for video streaming client is about 4 seconds (roughly same as segment size we ask for). With high enough available bandwidth, competing flow is very likely to fill up the router buffers in this long period.

One might think that reducing segment size from 4 seconds to a smaller value might mitigate the problem since competing flow might not get enough time to fill-up the router buffer. However, it would aggravate other effect even more: smaller segments would prevent the congestion window to raise to fair share. On the contrary, we will see that given that On-Off sequence is sure to happen, larger segments are better at mitigating the downward spiral effect as shown in next sub-section.

2.3 Author Suggested Improvements:

Authors have suggested 3 heuristics to mitigate the effect of downward spiral. They are as follows:

1. Bigger Segments
2. Optimistic Rate Look-up
3. Wider Moving Average

We explain the motivation of each suggestion and show with our experiments that each of them help to mitigate the downward spiral.

2.3.1 Bigger Segments

Authors suggest that bigger segments allow congestion window to raise to fair share. We verify this with our experiment (7) in which we observe congestion window of video flow over On-Off sequence in presence of competing flow. We can see that asking for larger segments allows congestion window to raise to fair share and so the throughput (bandwidth) estimation is better.

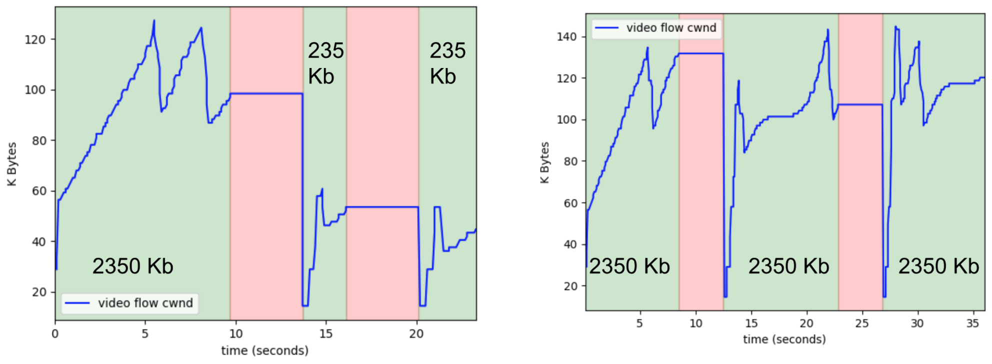

Figure 7: Motivation for Larger Segments

In this experiment (8), we show that increasing the segment sizes actually helps to decrease the downward spiral effect. The sharp downward dip becomes lesser as we increase segment sizes.

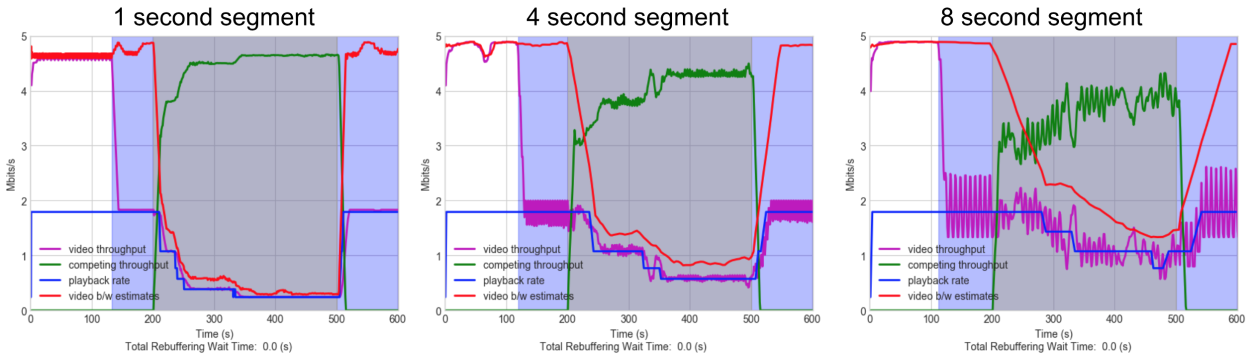

Figure 8: Effect by larger segments

2.3.2 Optimistic Rate Look-up

Since we know lower rate leads to even lower rate, it's better to be optimistic in choose playback rate. Hence, given the estimated throughput, authors suggest that we should choose the highest playback rate available smaller than throughput. By default, the range look-up for rate is pessimistic as shown by original Netflix client. We can make the client optimistic by changing the range look-up as shown in (9).

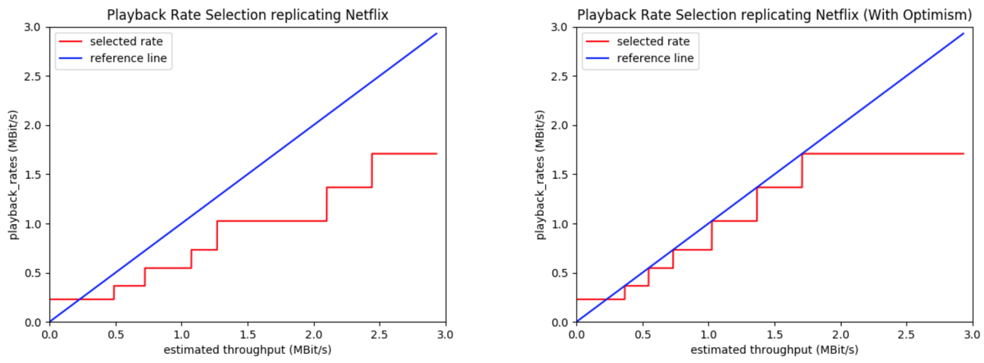

Figure 9: Pessimistic vs Optimistic Rate Lookup

In the this experiment (10), we show that optimistic client decreases the effect of downward spiral.

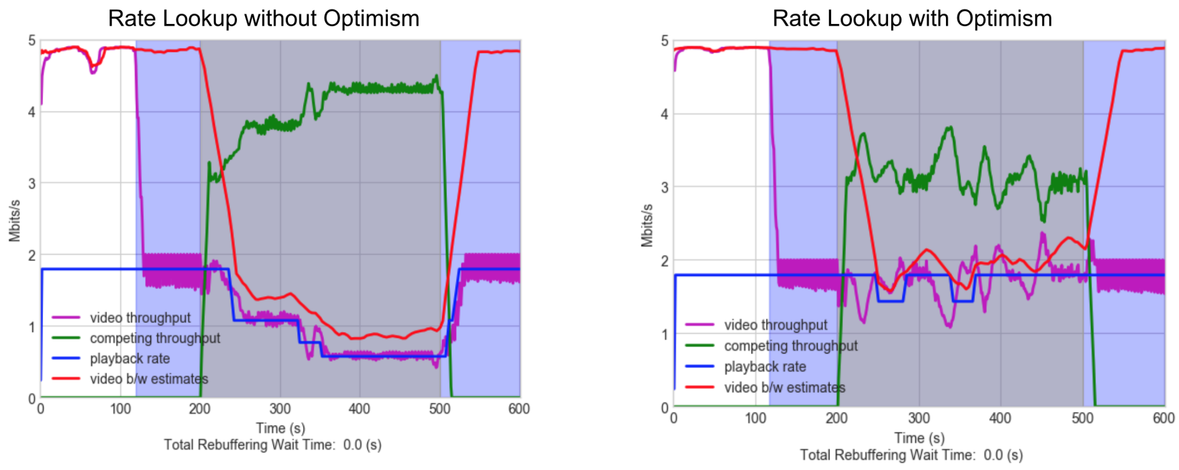

Figure 10: Client with Pessimistic (Left) vs Optimistic (Right) Rate Lookup

2.3.3 Wider Moving Average

Moving average of an estimate helps to smooth out and make the estimate more stable. Hence, the authors suggest that using a wider moving average of estimated throughput should help to decrease the effect. And in this experiment (11), we confirm that increasing the window of moving average window does indeed help to reduce downward spiral. The sharp downward dip becomes lesser as we increase window of moving average.

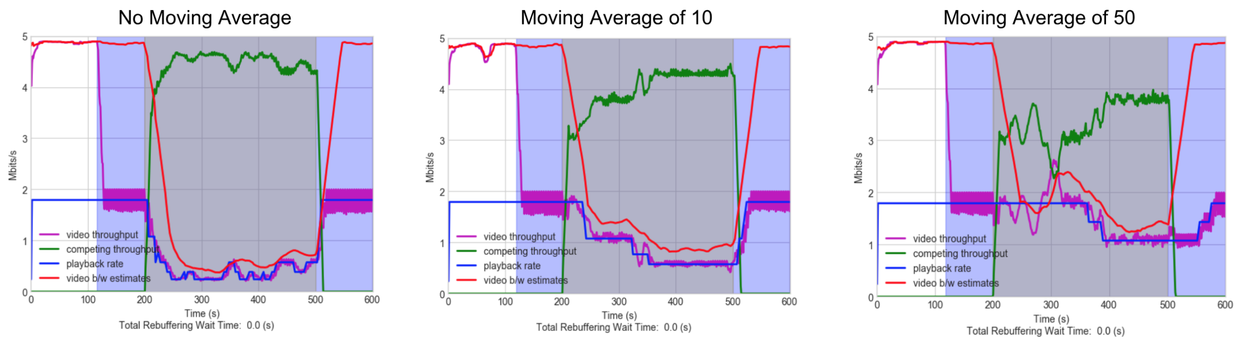

Figure 11: Effect of wider window for moving average

2.4 Our Improvement - Controlled Throw (extra 30%)

2.4.1 Motivating Observation

While we were calibrating our topology parameters to see the downward spiral effect, we observed the following (12): On left is a experiment in which playback buffer fills up before the competing flow starts and on right, the experiment in which competing flow starts before the playback buffer fills up. The key observation here is that in the yellow region, the flows are fair to each other and problem only happens after the playback buffer fill-up the first time. We use this observation as motivation to come-up with our modification of algorithm that completely removes the downward spiral and is closest to the fair share scenario without any ill side-effects.

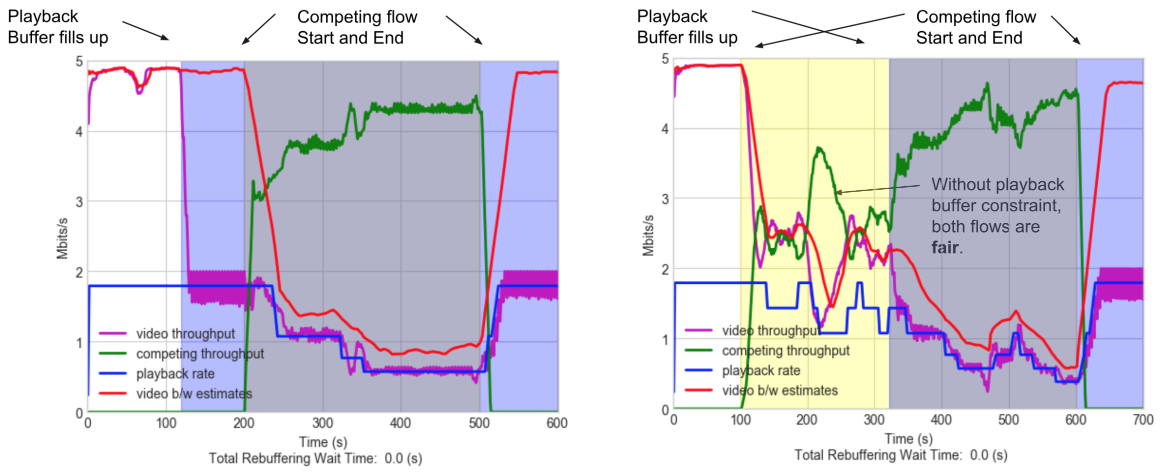

Figure 12: Playback Buffer fill up before Competing Flow start (Left) vs Competing Flow start before Playback Buffer fillup (Right)

2.4.2 Proposed Change

Main culprit of Downward Spiral is ON-OFF sequence and the only cause of ON-OFF sequence is the playback buffer filling-up. We saw in the figure (12) that before the buffer filling up for the first time, both flows were fair to each other which we ideally want. One fix of this is to change the algorithm to explicitly prevent playback buffer from filling-up and hence preventing the ON-OFF sequence.

One simple way to achieve this is by **"controlled"** throwing away of segments from the playback buffer. From figure (2), instead of waiting for playback to drain for 1 segment size (typically 4 seconds), we drop 1 segment which came the latest in the playback buffer queue when enough space is not available. Now since the client doesn't need to wait for the playback buffer to drain, it does not go in OFF phase. Hence, this simple procedure explicitly prevents the On-Off sequence.

Although this is a very counter-intuitive idea, it **completely** removes downward spiral from root cause and gives optimal fair video streaming without any adverse side-effect on video streaming. In terms of quality of video streaming, it chooses the highest rate by maintaining good estimate of fair share bandwidth and yet maintains high playback buffer preventing to incur any extra re-buffering events.

Note that we call it "controlled throwing" because we are discarding minimum segments (only 1) from the playback buffer necessary for preventing OFF phase when required. Discarding 1 segment of playback buffer when it is full is absolutely safe and does not cost any extra re-buffering events. If we had thrown full or large portion playback buffer it would have costed us frequent re-buffering events and bad viewer experience, but that is not what we are suggesting.

Recall that the ideal scenario that we want is that both flows are fair to each other and video streaming happens at highest playback rate with least re-buffering events. Controlled throwing assures best video streaming experience (high quality w/o re-buffering events) recovering a fair sharing of flows completely.

2.4.3 Results

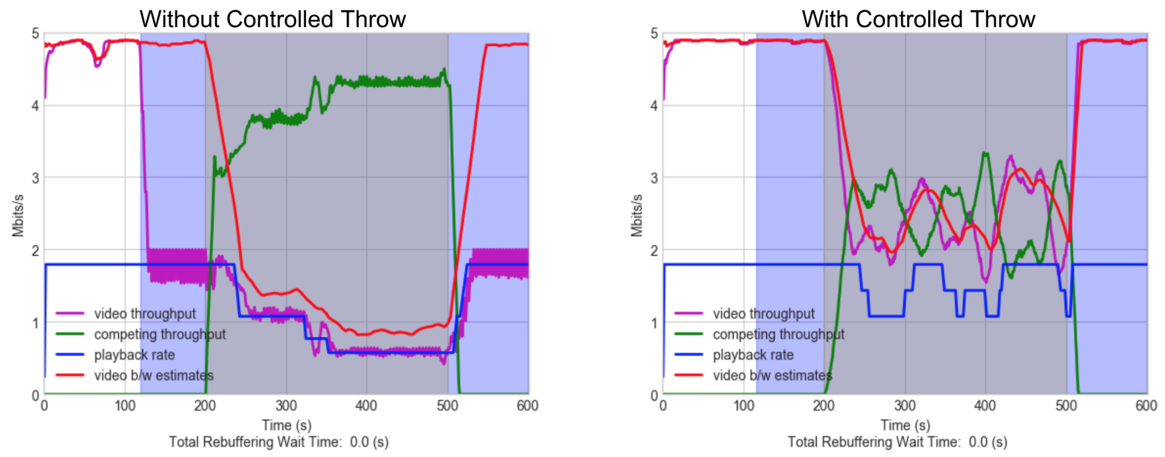

Figure 13: Effect of our improvement: Original (Left), Controlled Throw (Right)

The apparent fluctuation of rate in figure (13) is **not** because of downward spiral at all. It is because of pessimistic range-lookup table that generally services use. The netlfix range-lookup table (left look-up of figure (9)) assigns 1.5 Mbps rate in range of estimated bandwidth being 2.0 to 2.5 and allows 1.7 Mbps only if estimate exceeds 2.5 Mbps. This can be easily be fixed by using optimistic range-lookup table (right lookup of figure (9)) as discussed in 2.3.2. Figure (14) shows the comparison of original client and our improvement with optimistic rate look-up. It can be observed that it significantly improve video rate selection and hence video quality experience without any adverse effects on viewer experience whatsoever.

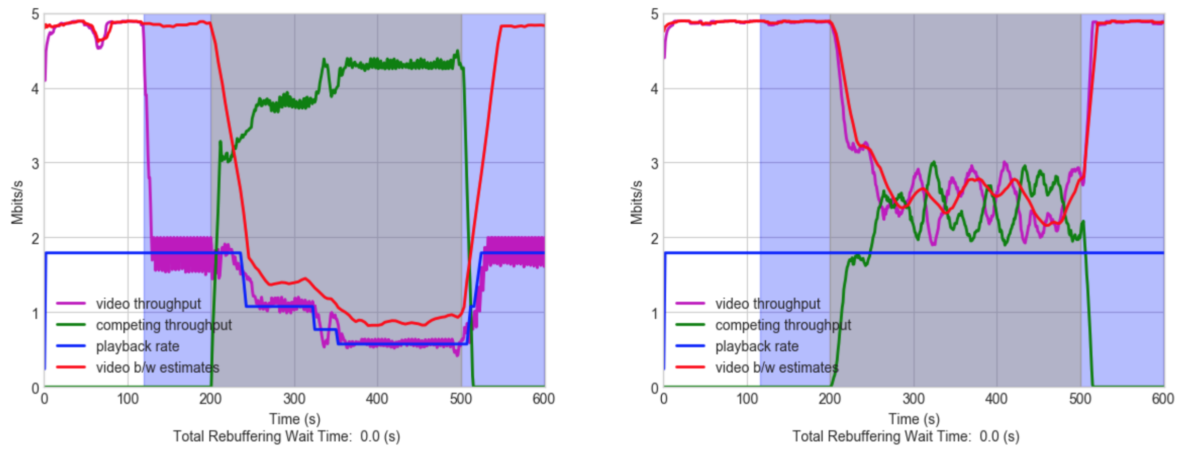

Figure 14: Effect of our improvement with Optimism: Original (Left), Controlled Throw with optimism (Right)

We **do not claim that "Controlled Buffer segment throw" is the best strategy**. Perhaps, there could be a different strategy to explicitly make sure On-Off sequence does not occur. But our experiments **definitely prove that** explicitly avoiding buffer to fillup eliminates the downward spiral from root cause without any adverse affect on video viewer experience whatsoever. We did give careful thought on **alternate strategy** for same purpose : for example being more optimistic in rate look-up when buffer is about to get filled and being pessimistic in rate look-up when buffer is about to be empty. We believe a calibrated algorithm for selecting optimistic rate lookup based on buffer emptiness / filledness should have similar result without the need of controlled throwing. But we leave this for future work.

3 Real Environment Experiments

Our aim is to reproduce the downward spiral effect in any of the popular video streaming services. We run experiments on streaming service - Vimeo and observe the downward spiral effect.

3.1 Experimental Setup

To replicate the experimental setup as in [1], the only network emulation required in a real world setup is a bottleneck link. We have used Mahimahi[3] to emulate the bottleneck link. Unlike traffic control (tc), mahimahi provides an isolated environment for our experiment to run. Hence, we do not have to worry about background processes running on our machine leading to inaccurate measurements due to flows not introduced by us.

mm-link [4] emulates up-links and down-links based on packet delivery schedules i.e: trace files. Each line in the trace file is a number which represents the time (in milliseconds) at which 1500 bytes can be delivered.

In our setup, the client streams a video from a video streaming service. 50 seconds after video streaming starts, a competing flow is introduced. To introduce a competing flow, we use youtube-dl (a tool that downloads videos from YouTube and other services [5]) to download the same video that we are streaming. We manually kill the download (competing flow) after 75 seconds of its starts.

3.2 Reproducing Downward Spiral Effect

In order to reproduce the Downward Spiral Effect, we need client throughput, Competing Flow throughput and playback rate. We briefly go over the implementation details of how they were measured.

1. **Throughputs:** Throughput for the competing flow and client flow is calculated using the –meter option of mm-link in mahimahi which logs the throughput in a log file. We parse the file and plot the values.

2. **Playback Rate:**

   Vimeo uses MPEG-DASH for video streaming. DASH(Dynamic Adaptive Streaming over HTTP) is a combination of server and client side algorithm which adjusts the video quality by changing the bit-rate, resolution based on the client's bandwidth.

   On the server side, the video is divided into segments that are encoded at different bit-rates and resolutions. The segments are hosted on HTTP servers and can be requested by the client by GET requests.

   The bit-rate and resolution is chosen at the client end and is measured each time it requests a new segment. This paradigm makes sense because the client is aware of its throughput and bandwidth and can make an informed decision about the bit-rate and/or resolution it wishes to stream.

   Thus, in order to inform the clients about different bit-rates, resolutions that the server can offer, MPEG-DASH has a Media Presentation Description (MPD) file. MPD is a file in XML format which provides a URL for each segment in different qualities. Hence, all clients, before streaming the video request the MPD file.

   The client, constantly monitors the network bandwidth and requests segments that are best suited to its requirements.

   We do not have a custom client and hence, we need to determine the bit-rates requested by the client for each segment in order to be able to calculate the playback rate.

   - We download the DASH manifest file prior to streaming the video using youtube-dl -F option which lists all available formats of the video.
   - HAR (HTTP Archive Format) has a log of all HTTP Requests made by the client in JSON format. Once the client has stopped streaming the video, we save the HAR File.
   - Matching the requests made by the client during the video streaming against the bit-rates in the MPD file, we calculate the playback rate.

3.3 Result

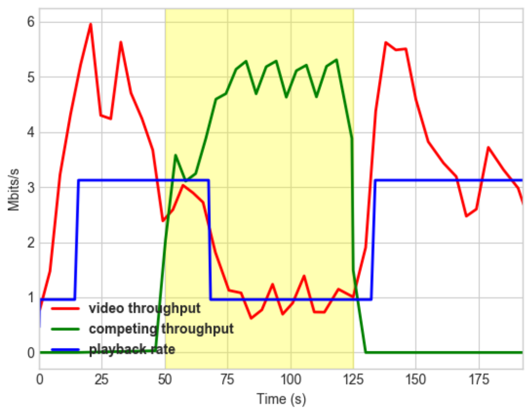

Figure 15: Downward Spiral Effect Reproduced in Vimeo (Right)

Figure (15) shows a classic downward spiral wherein the playback rate and client throughput plummet when the competing flow is introduced and does not recover until we kill the competing flow.

4 Conclusion

Following are the key things we did as part of this project

1. We reproduced the downward spiral effect with our video streaming client / server following the guidelines of how typical HTTP Client / Server works.
2. We perform several controlled experiments to verify multi-step cause of the this effect on mininet environment.
3. With our experiments, we show that suggested 3 improvements of authors to mitigate this effect actually help.
4. Based on our observations, we provide our own change to video client which completely removes this effect from root cause.
5. We justify how our improvement significantly improves video quality without any adverse effects on viewer experience whatsoever and discuss a future direction of our proof-of-concept work.
6. Finally, we also verify that "downward spiral" happens in real environments with test on Vimeo video service.

5 Clarifications Summary

1. The downward spiral effect will not always happen. We ourselves had to calibrate the topology parameters by trial and error to see this effect. However, most of the times it is very likely to occur as explained in **2.2.4**.
2. Our improvement of "Controlled Buffer segment throw" although very counter-intuitive, it solves both effects (1) and (2) of downward spiral by completely removing the ON-OFF sequence which was root cause of both the effects. This fix significantly improves video quality without any adverse effects on viewer experience whatsoever (**2.4**).
3. We do not claim that "Controlled Buffer segment throw" is the best strategy. Perhaps, there could be a different strategy to explicitly make sure On-Off sequence does not occur. But our experiments definitely prove that explicitly avoiding buffer to fillup eliminates the downward spiral from root cause without any adverse affect on video viewer experience whatsoever. In future, we can work on alternate strategies for same purpose as discussed in **2.4.3**.
4. In emulation experiments, our video streaming client is our own code. In spite of this, we were able to calibrate and reproduce all the effects, causes and remedies of downward spiral as explained by the authors (verbally or pictorially in the paper) which proves as a sanity check of our implementation. Our video-client implementation was based on general schema on how the video streaming over http works as partially explained in paper and was calibrated based on how Netflix works.

References

[1] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proceedings of the 2012 Internet Measurement Conference*, pages 225–238. ACM, 2012.

[2] Mininet - An Instant Virtual Network on your Laptop. http://mininet.org/. [Online; accessed 20-April-2018].

[3] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *USENIX Annual Technical Conference*, pages 417–429, 2015.

[4] mm-link. https://github.com/ravinet/mahimahi/blob/master/man/mm-link.1.

[5] Youtube-dl - Download videos from YouTube (and more sites). https://rg3.github.io/youtube-dl/.