Adaptive Bitrate Streaming Project Report - Content Distribution in Wireless Networks

IMPLEMENTATION OF BBA-0 AND IMPROVED ALGORITHM (ADAPTATION OF BBA-1)

SUPERVISED BY:
PROF. RAMON APARICIO PARDO

SUBMITTED ON: 22/11/2018
SUBMITTED BY: (UBINET)

- CYRIL NAVES
- ZAUWALI SABITU PAKI

Objective: To understand the tradeoff in terms of Quality of Experience between Video Quality and rebuffering time by designing our own adaptation algorithm and implementing an existing algorithm BBA-0 **[T.Y.Huang ., et al]**

Implementation of BBA-0:

We implemented the BBA-0 algorithm as given by the pseudo code in the paper for Algorithm 1 "***A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service"***.

- We try to ascertain through this implementation that these buffered based algorithms should not experience rebuffering events but at the cost of requesting more conservative video rates.
- We implement the pseudo code by implementing with the TapasPlayer a controller which is invoked by the player.
- Our main methods where our entire logic will be held is in the function **calcControlAction()** where we dynamically vary based on the available video rates and the buffer occupancy.
- Given video rates are [12500.0, 33104.0, 85815.0, 108682.0, 259240.0, 523845.0] which are of 6 levels
- Also, we try to obtain a linear function f(B) given by linear interpolation by the points:

$(x_1, y_1) = (r_{min}, reservoir)$

$(x_2, y_2) = (r_{max}, cushion+reservoir)$

$M=y_2-y_1/x_2-x_1$

Then Linear equation given by $(y-y_1) = m\ (x-x_1)$

Value = (r_max * buf_occup-r_max *reservoir r_min* buf_occup + r_min* reservoir+ rmin * cushion)/cushion;

- We used different values of the lower reservoir and the upper reservoir. For example, when we set the value of the lower reservoir to 30% of the buffer size and the upper reservoir to 20% of the total buffer size, we plotted the result and we obtained plot of the figure 1 below:

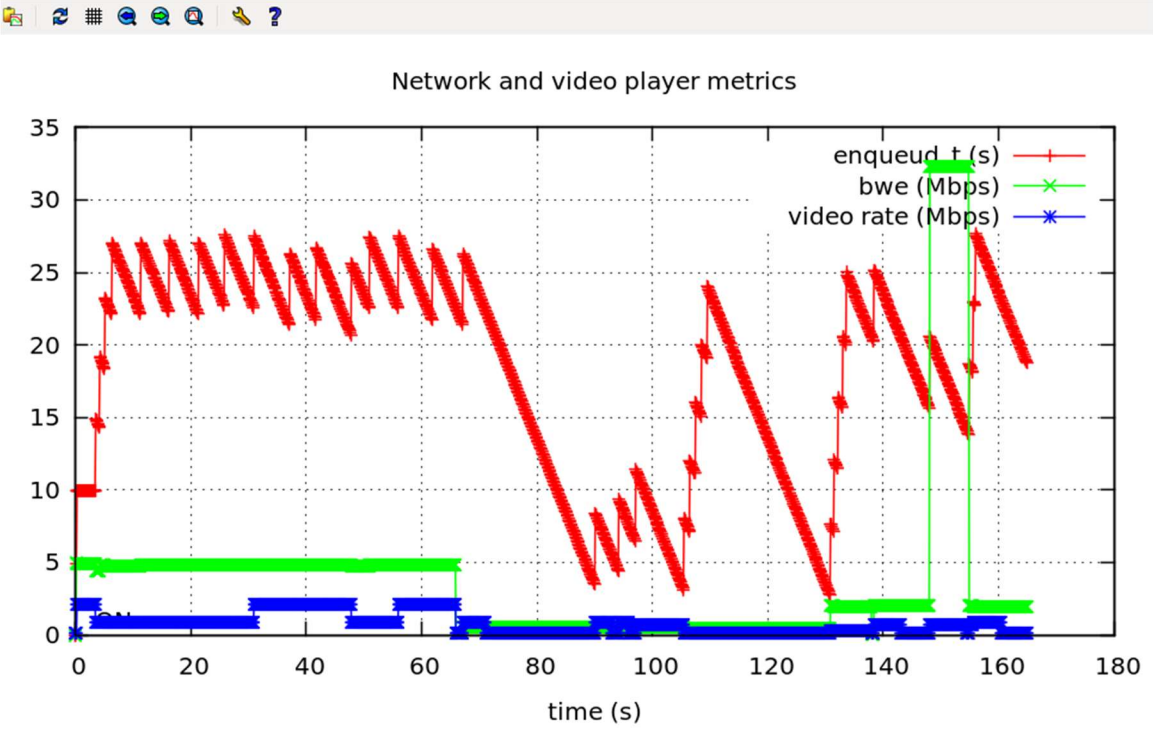

Figure 1: BBA-0 with reservoir 30% of the buffer size and the upper reservoir 10% of the total buffer size. Cushion is the remaining 50% of the total buffer size

- We observed no rebuffering event as can be seen from figure 1. It can also be observed from figure 1 that we noticed early playout start.
- However, we changed both the lower and upper reservoirs to 30% of the total buffer size each, we noticed few rebuffering events as indicated by the figure 2 below.

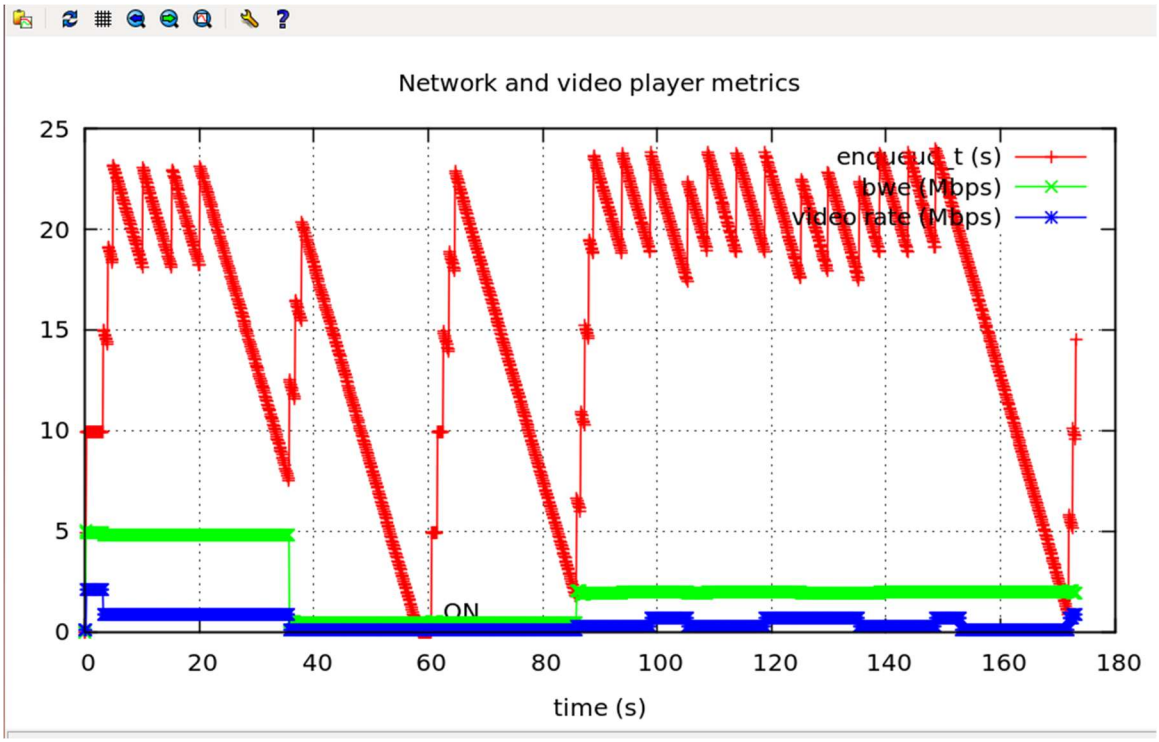

Figure 2: lower and upper reservoir fixed to 30% of the total buffer size each.

IMPROVED VERSION OF ALGORITHM (AN ADAPTATION FROM BBA-1):

We try to implement a better algorithm to BBA-0, which is not a perfect adaptation to BBA-1 but follows a similar methodology. Although we set the reservoir to an arbitrary value in the previous algorithm we try to improve the dynamics of setting the reservoir size as well as the linear function based on the incoming next chunk sizes for an expected segment index.

Goal: We try to avoid /minimize the re-buffer rate and improve the average video rate achieved by BBA-0 through:

1) Dynamic Reservoir
2) Chunk Map

I) **Dynamic Reservoir:** We modify the dynamic reservoir based on the current video rate as well as the bandwidth where we try to obtain the effective difference which gives us the net buffer inflow and we try to calculate the net time in buffer which is needing to accommodate at least one chunk of incoming data.

EffectiveRate = Bandwidth - CurrentVideoPlayingRate

Reservoir = ChunkSize / EffectiveRate

II) **Chunk Map:** We try to Map a Linear function which is based on the chunk size rather than the video rate.

Chunk Size Derivation:

- We obtain the Chunk sizes through the **PlayList** Data Structure available in the ***TapasPlayer.py***. We add to the feedback dictionary this **Playlist** data structure as well the current playing index **cur_index.**
- Based on the current index which ranges from 1 to 37 we obtain, the **byte_range** and then we can effectively calculate the chunk sizes.
- After calculating the various chunk-sizes for each of 6 level we get:

Level 0: [[62067, 61016, 61049, 61024, 61032, 60908, 61003, 61011, 60969, 60960, 61058, 60910, 61180, 60991, 61029, 61017, 60904, 61049, 60958, 61060, 60945, 61021, 61116, 60967, 61109, 61167, 61010, 61145, 61021, 61031, 61359, 61088, 48263, 60927, 61387, 29581, 5963],

Level 1: [171873, 169399, 169684, 169686, 169937, 172205, 170251, 170003, 169598, 168603, 170146, 169650, 169748, 169402, 169325, 169234, 170005, 171552, 169474, 170409, 170325, 171080, 170125, 169893, 170385, 171263, 169318, 168892, 169844, 170434, 168462, 169676, 131260, 169716, 169969, 89832, 14306],

Level 2: [696496, 686457, 560288, 337405, 411468, 616375, 611703, 513546, 517568, 296396, 357761, 382207, 293730, 494251, 358540, 327741, 342819, 320792, 394811, 540846, 562997, 494879, 319256, 522205, 422554, 473994, 513326, 489346, 418793, 533064, 444014, 472072, 171287, 413867, 168818, 94046, 17470],

Level 3: [727646, 723259, 713277, 441643, 545360, 703478, 720088, 649713, 707045, 388947, 480640, 506462, 390025, 638974, 483502, 448555, 437941, 430988, 538779, 707934, 705096, 618632, 403435, 693758, 556120, 635623, 679346, 597048, 539315, 652661, 573083, 595883, 221618, 530739, 193901, 119509, 23133],

Level 4: [1848732, 1891159, 1755018, 1056119, 1224947, 1853318, 1863361, 1467858, 1716051, 923281, 1167127, 1190189, 957762, 1567638, 1159046, 1098672, 1030333, 1038851, 1304478, 1671128, 1718970, 1425577, 939680, 1574986, 1264621, 1467969, 1569169, 1335385, 1247178, 1431133, 1302142, 1249882, 536499, 1218961, 611644, 348210, 60281],

Level 5: [3901497, 3798568, 3541977, 2349797, 2551695, 3695870, 3928180, 2898383, 3471435, 1923558, 2416071, 2445849, 2058194, 3258135, 2490834, 2317504, 2077665, 2251482, 2889612, 3495883, 3478395, 2860544, 1928270, 3141548, 2580525, 3106305,

3232102, 2644482, 2442798, 2534357, 2295766, 2355741, 796584, 2466481, 931705, 624210, 104306]]

- Each level will have 37 index segment chunks which vary in size dynamically set by the player.
- Then we map these chunk sizes depending on the linear function versus the buffer occupancy which is obtained by the interpolation of the two points:

(x1, y1) = (chunk_min, reservoir)

(x2, y2) = (chunk_max,cushion+reservoir)

M=y2-y1/x2-x1

- Then Linear equation given by (y-y1) = m (x-x1)
- Our function will have the parameter based on **buffer_occupancy and chunk_sizes for a particular video index** at any given point of time which is derived as below, and it will vary with the chunk
- **Value = (chunk_max * buf_occup - chunk_max* reservoir - chunk_min* buf_occup + chunk_min * reservoir + chunk_min * cushion)/cushion;**

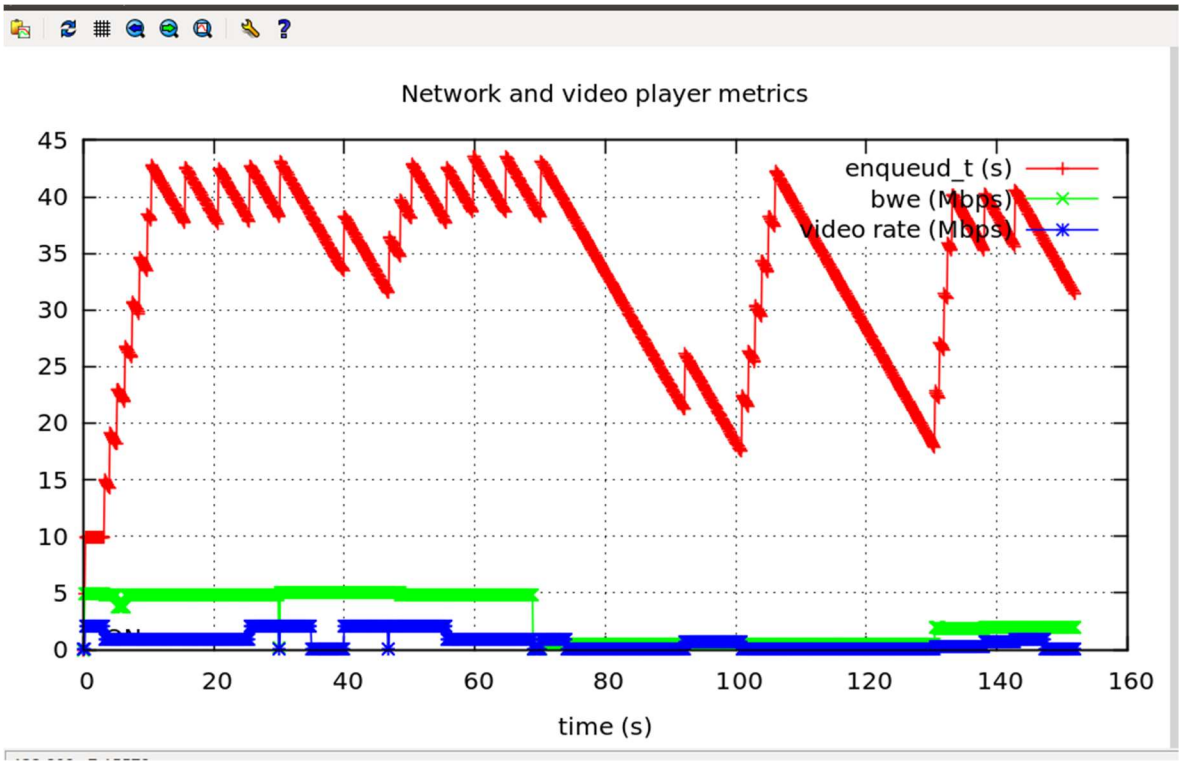

Figure 3: Improved Algorithm Plot (A modification of BBA1)

Result Outcomes:

- We notice higher average video rate compared to the BBA-0 in this adaptation of BBA-1 and start-up time is higher also.
- Also, there is a pause time which is lower compared to BBA-0
- We also notice no rebuffering event in this case which is an added advantage for us.
- By varying the rate based on chunk sizes (6 * 37 (for each level) =222) rather than the available 6 video rates gives us a major improvement in the average video rate adopted.
- Dynamic Reservoir based on the effective rate which inflows into the player client over the incoming fragment size gives us an estimate of the exact reservoir we need to avoid any rebufferings

Comparison of BBA-0 and Improved Algorithm (Adaptation from BBA-1):

Parameters	BBA-0	Improved Algorithm (Adapted from BBA-1)
Higher Average Video Rate	0.24310402	0.731429831
Lower Total Paused Time	3.553	3.535

Table 1: Comparison of BBA-0 and BBA-1 Results

Conclusion:

- We study the BBA-0 algorithm and we understand that we avoid rebufferings at eh cost of lower video rate by fixing arbitrary reservoir and cushion values and a linear function which varies with the video rate
- Also, we propose an algorithm which is modified from BBA-1 to vary based on the chunk size rather than the video rate in the linear function. We also get the value of the reservoir dynamically based on the available effective rate and incoming chunk which gives us better result in average video rate and not compromising on pause time and rebuffering which is desirable in any adaptive bitrate streaming algorithm