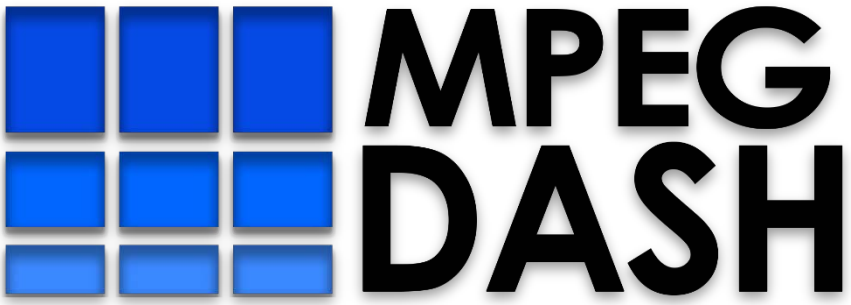
MPEG-DASH livestream in an unstable environment



Project in computer networks -

Winter 2019/2020

Team Members: Avraham Ayaso, Lavy Raz

Instructors: Itzik Ashkenazi, Aviel Glam (Rafael)

Table of Contents

Introduction

In this era, media is being consumed on a regular basis all over the world. At home, work, school, malls, everywhere. In some uses, receiving the media in a delay is not an issue. While in other uses, a delay in the consumed media might be crucial.

For example, watching live sports, consultation of a remote doctor who observes a surgery online, etc. Therefore, in such cases, live-stream is not a nice to have feature – it's a must.

General description

MPEG-DASH protocol main goal is to stream media files, while taking into consideration the clients' variant BW. It does so by constantly adjusting the streaming quality, which the client receives, in accordance to his current network BW.

The protocol main goal was to stream media files and wasn't meant for live streaming.

The research examines the protocol, with emphasis on live streaming in an unstable environment.

Previous work

- Explore several open sources of MPEG-DASH
- Setup a stable basic environment which includes a server and a client on a LAN network
- Build a GUI

Objectives

Goals

1. Implement algorithms that manage the buffer and adjust it to the changing bandwidth, while maintaining live streaming.

2. Comprehensive analysis on each streaming session:
    a. algorithm's specifications
    b. streaming statistics
    c. bandwidth utilization

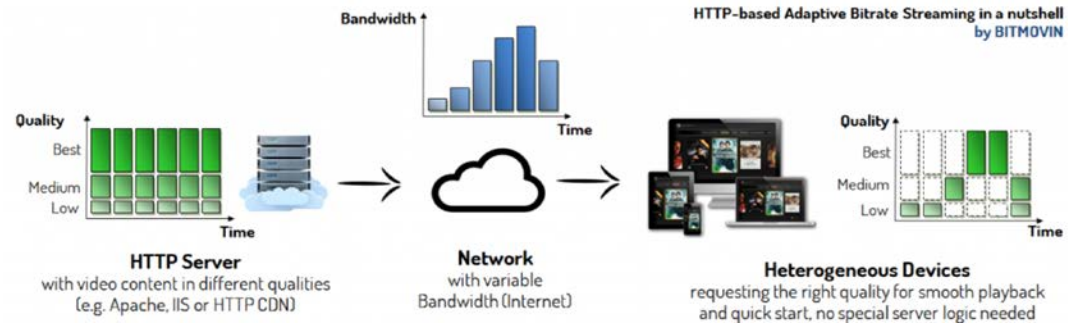
3. Create a user-friendly GUI:
    a. choose the algorithm specifications
    b. choose testing environment
    c. show live results and conclusions

Estimated timetable

- November
    - Research the MPEG-DASH protocol
    - Learn about the previous research and adapt to it
- December
    - Converse possible algorithms to achieve the project's goal
    - Update the open sources with the implemented algorithms
- January
    - Choose KPI (key performance indicator)
    - Analyze the algorithms in different environments
    - Demo and presentation delivery.

Background

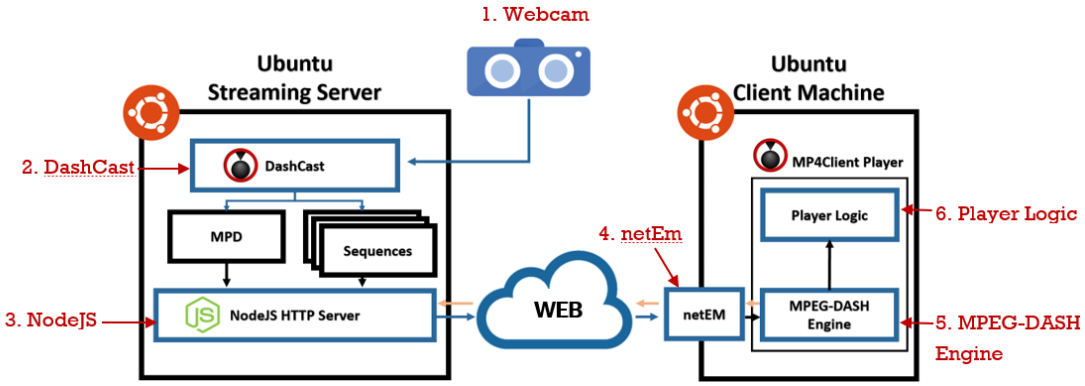
Dynamic Adaptive Streaming over HTTP (DASH), also known as MPEG-DASH, is an adaptive bitrate streaming technique that enables high quality streaming of media content over the Internet delivered from conventional HTTP web servers.



MPEG-DASH works by breaking the content into a sequence of small HTTP-based file segments, each segment containing a short interval of playback time of content, such as a movie or the live broadcast of a sports event. The content is made available at a variety of different bit rates. While the content is being played back by an MPEG-DASH client, the client uses a bit rate adaptation (ABR) algorithm to automatically select the segment with the highest bit rate possible that can be downloaded. Thus, an MPEG-DASH client can seamlessly adapt to changing network conditions and provide high quality streaming.

MPEG-DASH is the first adaptive bit-rate HTTP-based streaming solution that is an international standard. The transport protocol that MPEG-DASH uses is TCP. MPEG-DASH uses existing HTTP web server infrastructure that is used for delivery of essentially all web content. It allows devices to consume multimedia content (video, TV, radio, etc.) delivered via the Internet, coping with variable Internet receiving conditions. Standardizing an adaptive streaming solution is meant to provide confidence to the market that the solution can be adopted for universal deployment, compared to similar but more proprietary solutions like Smooth Streaming by Microsoft, or HDS by Adobe.

Architecture

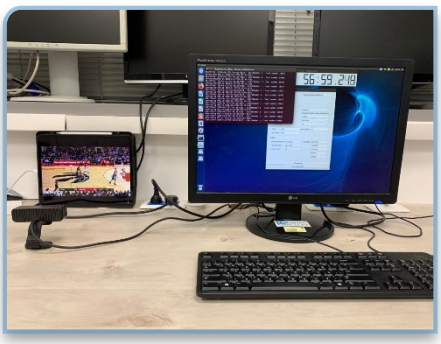


Components

1. Webcam: Installed on top of the server pc. Generates the live video stream.
2. DashCast: Breaks the input stream into sequences of small segments and stores them on the server. Each segment is stored in several different bitrates. It also creates an MPD file which defines the available qualities, and more features of the segments stored on the server.
3. NodeJS: Exposes the MPD file and the sequences through Web interface. The client receives the segments data using HTTP GET commands.
4. netEM: Simulates an unstable network behavior on the client's computer. Capable of adjusting the BW, packet loss and bit error rate.
5. MPEG-DASH engine: Generates HTTP GET commands to the server asking for the best quality the client can stream while trying to maintain live stream.
6. Player Logic: Plays the live video stream. ☺

The project's setup

Server



Client

Streaming adaptation methods

The research focuses on three main streaming adaptations methods.

1. Throughput based:
   This method relies on the client's network. The prediction of the network's BW is made based on the download rate of previous segments.

2. Buffer based:
   This method relies on the client's streaming buffer. The client's buffer is a buffer which holds the segments downloaded from the server but hasn't been played yet. The prediction of the network's BW is made based on the remaining segments in the buffer.

3. Hybrid:
   a combination of the above-mentioned methods.

Algorithms

Rate adaptation

- Throughput based algorithm
- Already existed in the GPAC open source
- It adapts the streaming quality according to the download rate of the last downloaded segment
    - No previous knowledge is kept

Rate adaptation algorithm has shown poor results in an unstable environment, which resulted in losing a significant number of frames and even image freeze. The main goal of the project is to prefer live stream over quality; Thus, buffer regulation was needed in order to stabilize the stream.

Buffer regulation

This is NOT a streaming adaptation algorithm. It is a buffer management method, which sets thresholds to identify the state of the client's buffer and acts accordingly.

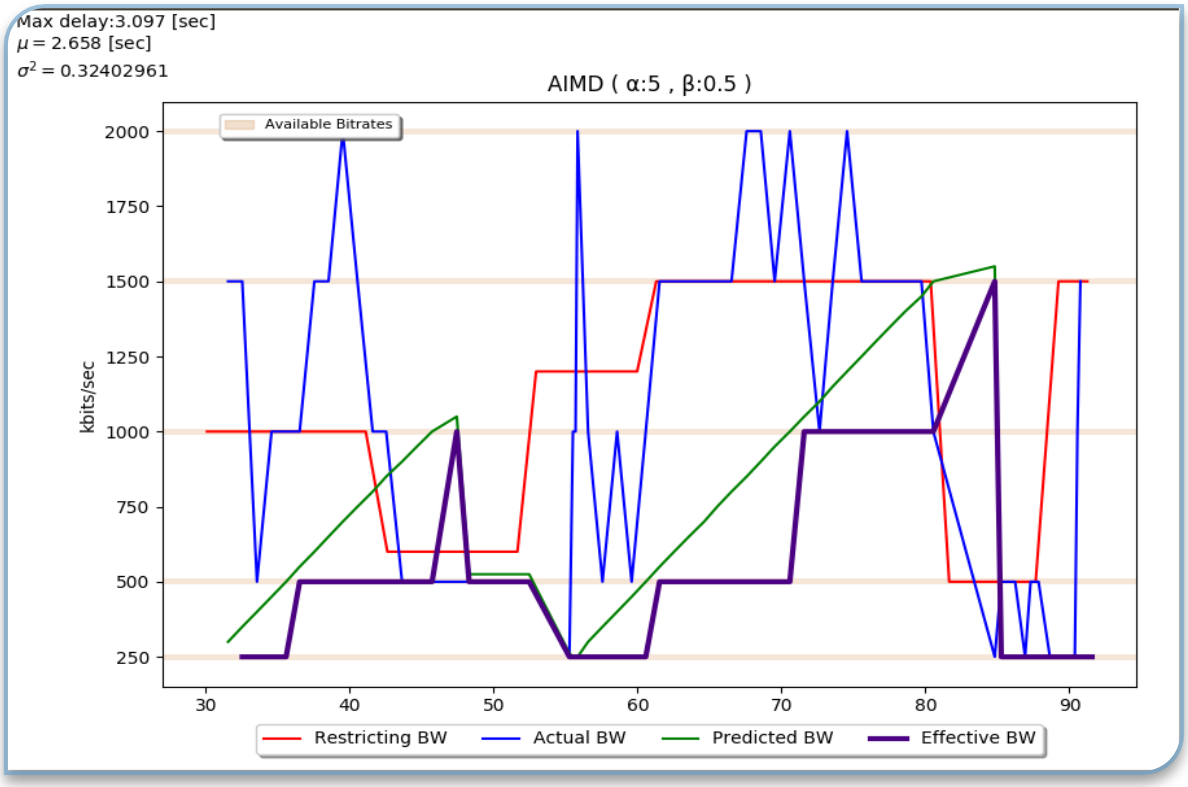
Recovery mode – a buffer state which indicates that the current quality cannot be upgraded because the buffer state is not stable, hence, an upgrade will cause an interruption to the live stream. This action overrides any algorithm prediction. The recovery from this state occurs only once the client's buffer surpasses buf_low_threshold.

The thresholds are:

- CRITICAL_OCCUPANCY – The critical threshold which indicates an interruption to the live stream. It directly changes the client's BW prediction to the lowest possible quality and enters recovery mode.
- buf_low_threshold – This threshold indicates a high probability that an interruption to the live-stream will happen if the number of segments in the buffer reaches below this value. When the number of segments reaches below this value, it lowers the client's BW prediction by the downgrade parameter given by the client and enters recovery mode.
- buf_high_threshold – This threshold indicates that the segments download rate is very high, thus the client is probably capable of downloading a better quality.

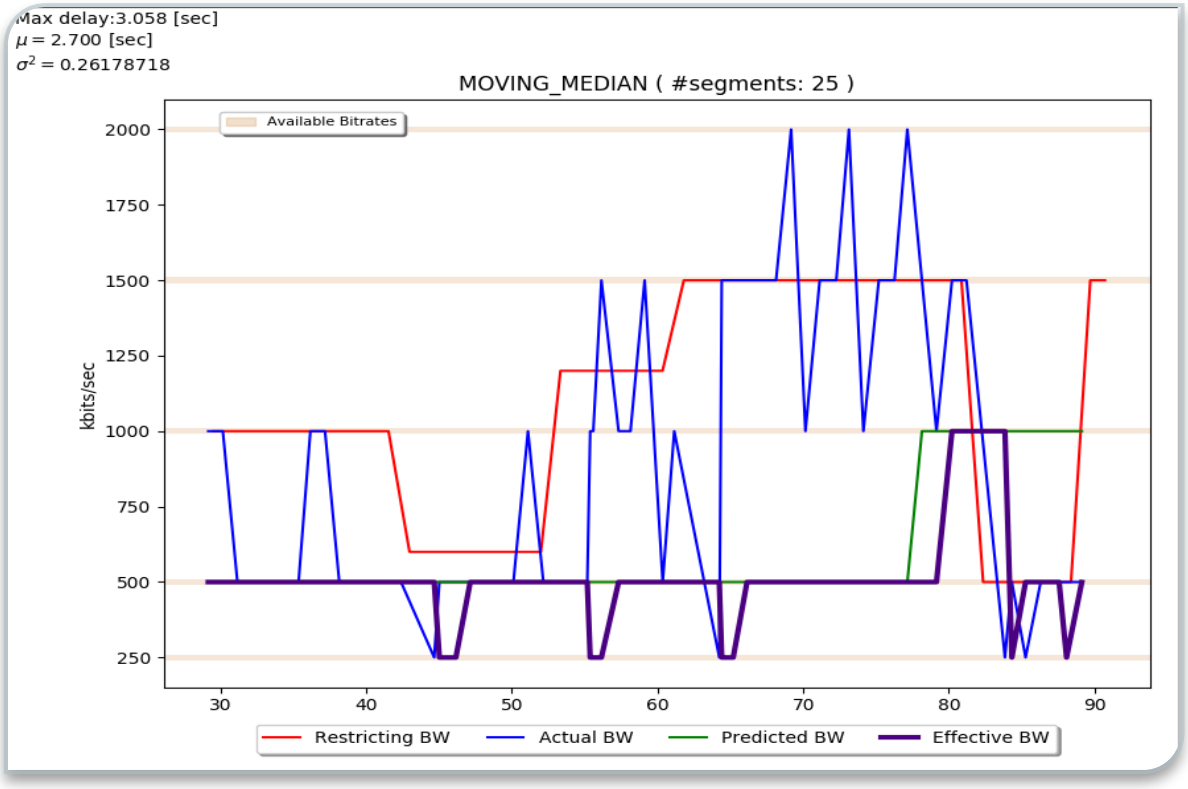
AIMD (Additive Increase Multiplicative Decrease)

- Buffer based algorithm
- Buffer regulation is used
- The algorithm was inspired by TCP congestion control:
  - With each successful downloaded segment, the prediction is increased by a α value
  - Packet failure is determined according to the buffer occupancy. Once a failure is detected, the prediction is decreased by a factor β value
  - α and β are constants given by the client



Moving median

- Hybrid algorithm
- Buffer regulation is used
- The algorithm is based on statistical analysis:
  - It samples the last N downloaded segments rates. Calculates their median, and the result is the algorithm's prediction
  - N – is a constant that is given by the client



KPI (Key Performance Indicator)

Hazard

Hazard is a period, in which the buffer is found in recovery mode.
It indicates the time in which the client's buffer has few segments to play, thus the live stream is more probable to freeze, hence the received traffic during that period is considered squandered.

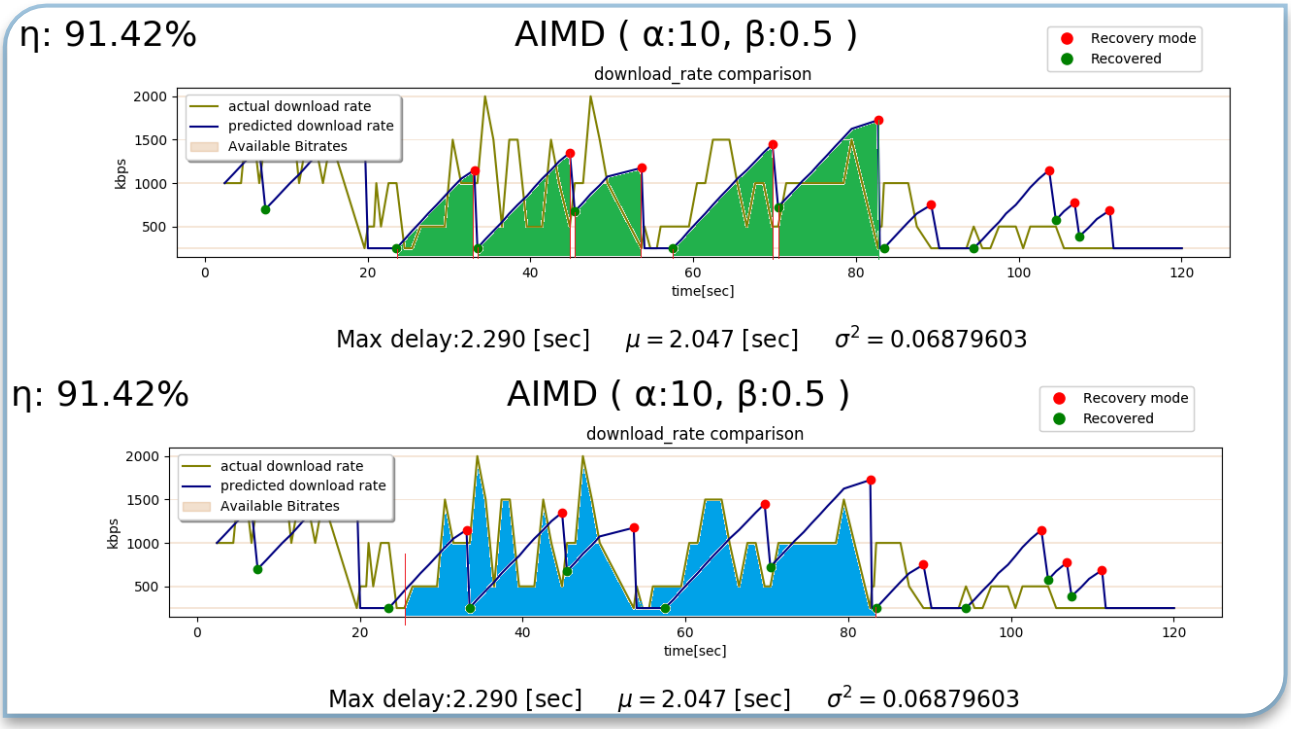
In order to determine the quality of a given algorithm, the following KPI were defined.

Comparison of traffic volumes

The comparison variable will be denoted as: η

Motive - measures the utilization of the algorithms from the client side.

- The comparison is between:
  - actual traffic volume received
  - the predicted traffic volume of the algorithm without calculating hazards traffic volume



Optimal BW

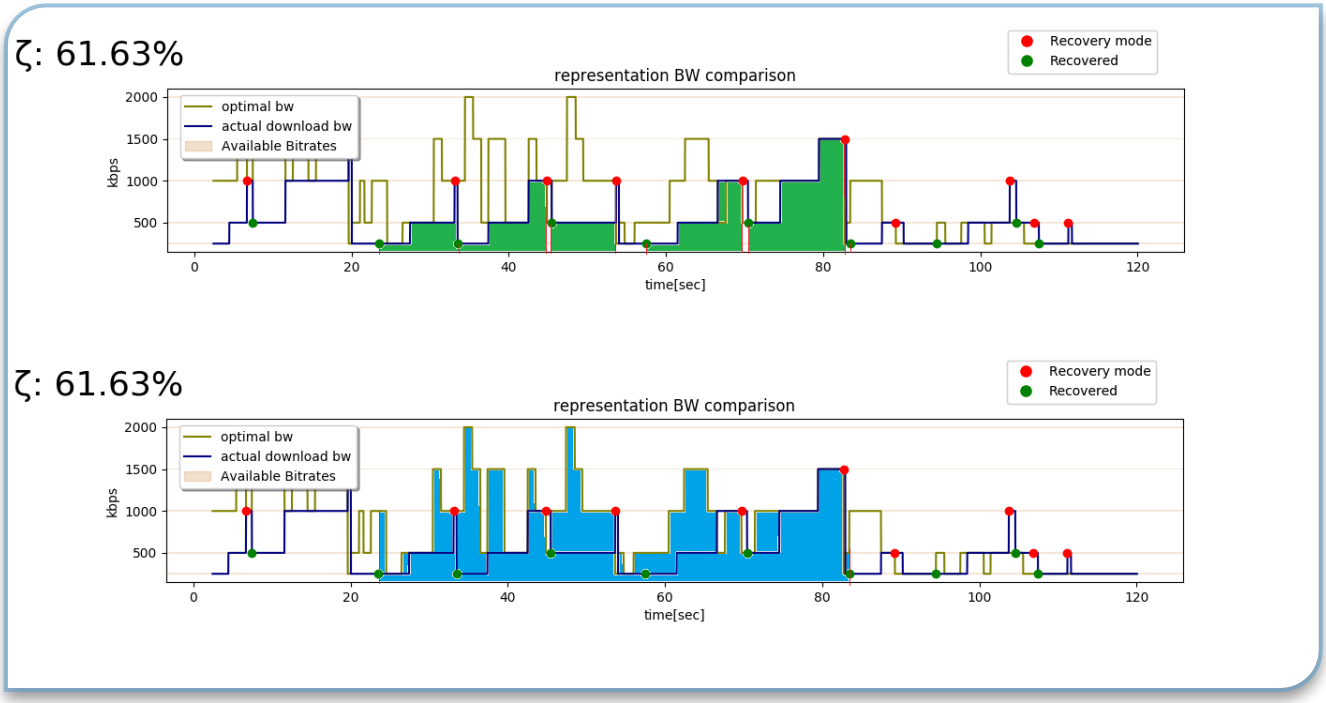
Highest BW the server provides, which is lower than the actual download rate

Comparison of server's BW

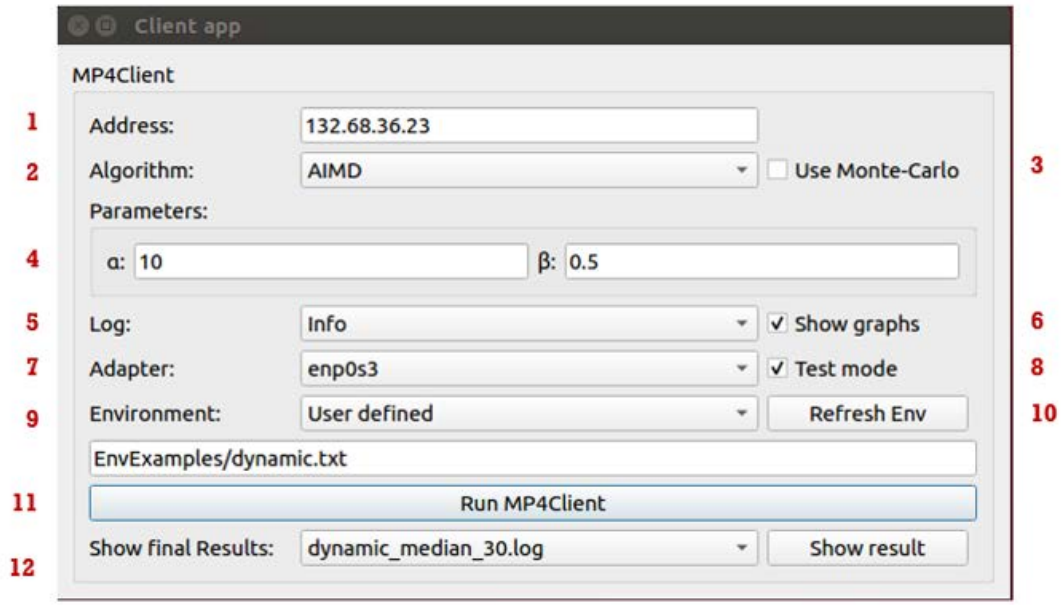
The comparison variable will be denoted as: ζ

Motive - measures the utilization of the algorithms from the server side.

- The comparison is between:
  - Optimal traffic volume – the optimal traffic volume, that could have been sent by the server and potentially be received by the client, if each requested BW was the ideal BW
  - Requested traffic volume – the traffic volume that is sent by the server without calculating hazards traffic volume



Client's GUI



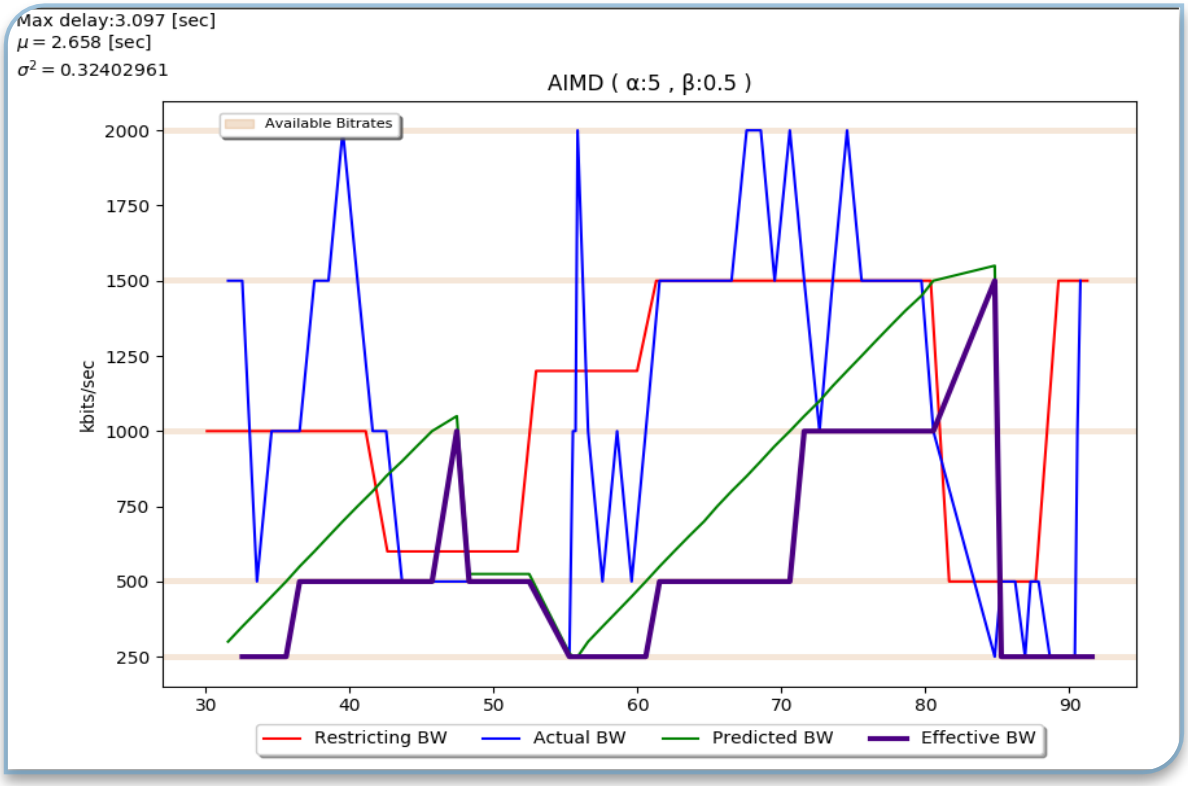
1. Address - The IP address of the streaming server
2. Algorithm - The prediction algorithm used to decide the stream quality
3. Use Monte-Carlo - Allows to add probability factor to the given algorithm
4. Parameters - Allows to choose the parameters the algorithm requires
5. Log - Decides how many details will be traced during the stream in the log file
6. Show graphs - Allows to show the graphs during run-time
7. Adapter - choose the client's adapter that will connect to the server
8. Test mode - Allows to auto close the run after two minutes
9. Environment - Allows to choose controlled environment using NetEM from a file
10. Refresh Env - Allows refreshing the environment with the current NetEM file
11. Run MP4Client - Allows the client to tune in a current stream
12. Show final results - Allows the client to show the results graph of the given log

Graphs Description

Runtime graph

The graph that is displayed during runtime if "Show graphs" is checked.

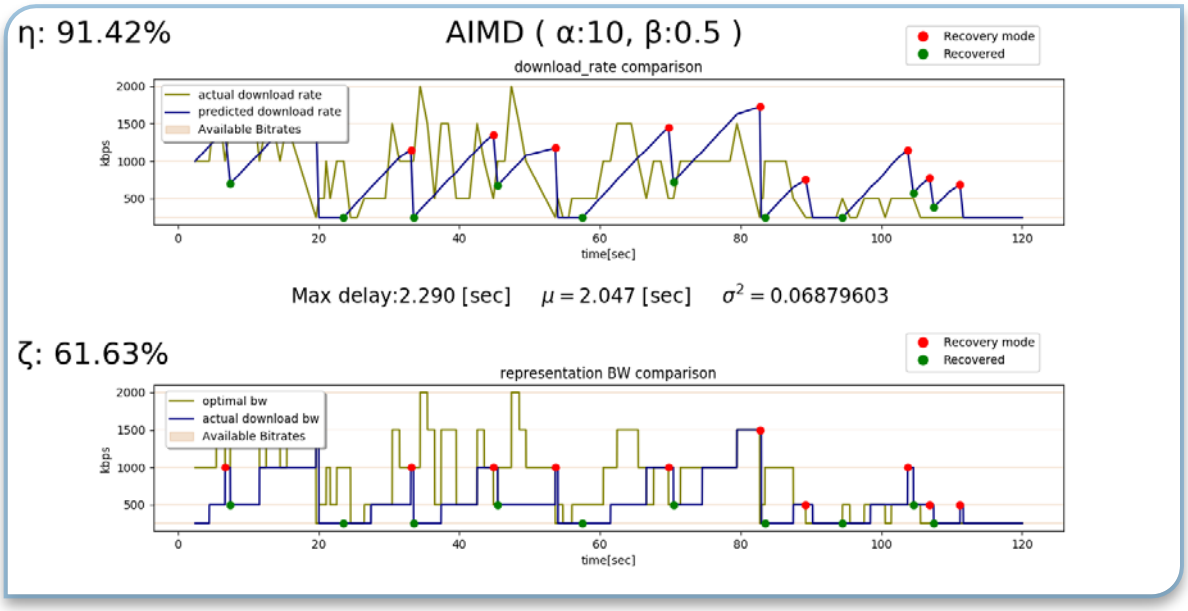
It gives live indication about the state of the stream.



- Restricting BW - The restriction set by the NetEM
- Actual BW - The actual download rate of every segment
- Predicted BW - The predicted download rate of the given algorithm
- Effective BW - The requested BW from the server at any given time
- Max delay - Approximation of the delay according to the client's buffer occupancy with an estimated delay between the server-client.
- μ - The expectancy of the delay approximation
- σ^2 - The variance of the delay approximation
- Available Bitrates - The available streaming qualities the server offers
- The name of the algorithm with its parameters are displayed at the top

Results graph

The graph that analyzes the results of a given streaming session after it is finished.



- η, ζ are the KPI as explained in p.11-12
- actual download rate - The actual download rate of every segment
- predicted download rate - The predicted download rate of the given algorithm
- optimal BW - Defined in p.12
- actual download BW - The requested BW from the server at any given time
- Max delay - Approximation of the delay according to the client's buffer occupancy with an estimated delay between the server-client.
- μ - The expectancy of the delay approximation
- σ^2 - The variance of the delay approximation
- Available Bitrates - The available streaming qualities the server offers
- The name of the algorithm with its parameters are displayed at the top
- Recovery mode - indicates the entrance to recovery mode as explained in p.8
- Recovered - indicates the exit from the recovery mode

Environments Definition

In order to have a good quality tests there is a need to simulate the same environment multiple times. There are five different environments that are chosen to test the algorithms.

$\underline{\Delta t}$ - maximum delay prior to changing the restricting BW

$\underline{\Delta BW}$ – the maximum difference in BW (absolute value) in a single change

Static: $\Delta t \rightarrow \infty$, $\Delta BW \equiv 0$

Quasi-static: Δt is large, ΔBW is small

Dynamic: Δt is medium, ΔBW is medium

Bursty: Δt is medium, ΔBW is mostly small, with uncommon peaks that are large

Noisy: Δt is small, ΔBW is large

The above-mentioned environments cover many common cases; hence this set of environments is a good set to test the algorithms.

Results

η - theoretical utilization (%)

	AIMD			MOVING-MEDIAN			PREF.ALG
Parameter / Environment	α=5	α=10	α=15	#segments=10	#segments=20	#segments=30	
Static	72.27	74.37	72.75	85.19	75.69	88.51	MEDIAN (30)
Quasi-static	74.93	91.42	94.81	80.06	83.82	75.46	AIMD (15)
Dynamic	55.09	70.34	64.39	95.73	88.39	72.11	MEDIAN (10)
Bursty	56.22	79.8	83.37	86.96	74.42	86.82	MEDIAN (10)
Noisy	42.82	71.48	65.05	62.49	71.48	82.3	MEDIAN (30)

ζ - practical utilization (%)

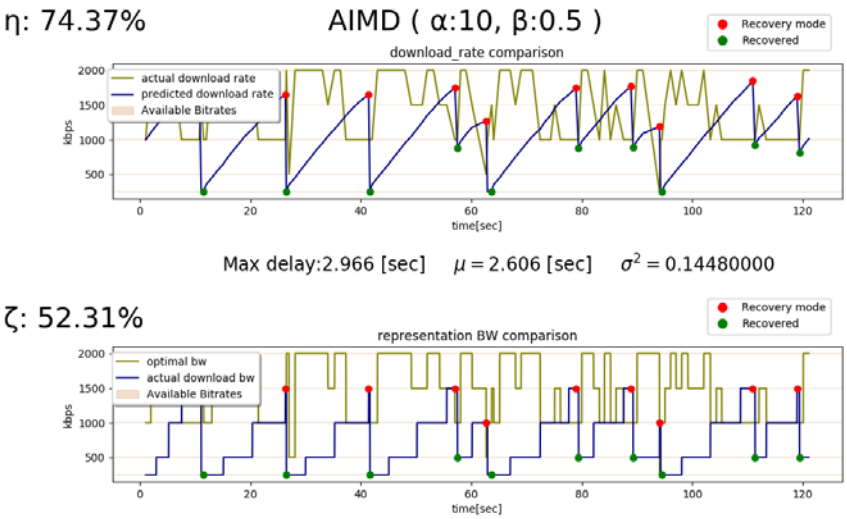
	AIMD			MOVING-MEDIAN			PREF.ALG
Parameter / Environment	α=5	α=10	α=15	#segments=10	#segments=20	#segments=30	
Static	54.19	52.31	49.3	70.8	67.6	77.11	MEDIAN (30)
Quasi-static	50.61	61.63	62.08	67.52	64.74	65.21	MEDIAN (10)
Dynamic	40.1	49.94	44.5	68.54	66.94	60.88	MEDIAN (10)
Bursty	40.6	53.96	57.87	57.52	44.41	55.71	AIMD (15)
Noisy	25.66	42.9	39.12	33.47	42.48	45.44	MEDIAN (30)

The following graphs present the best results in each environment.
If not specified differently, the given graphs represent the best results for both KPI.

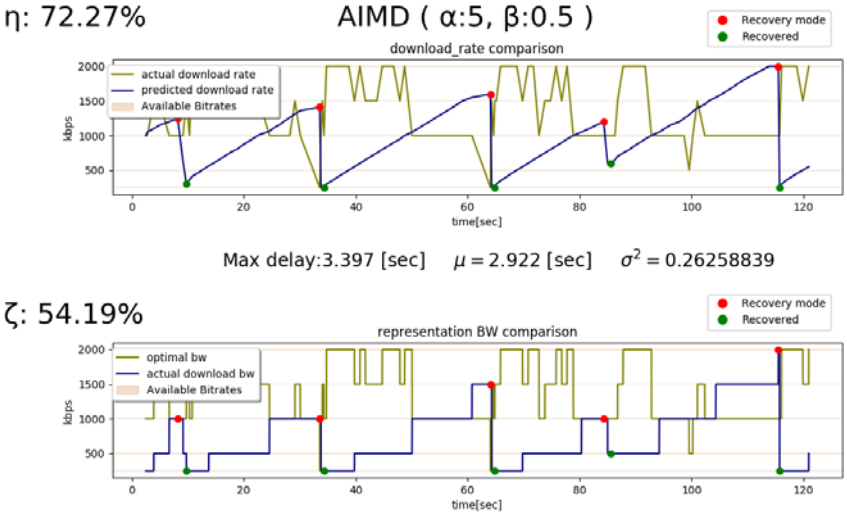
Static -

AIMD:

According to η:

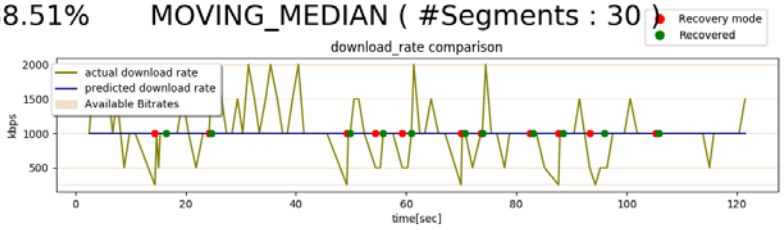


According to ζ:



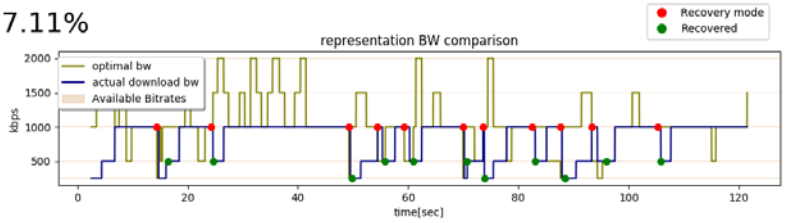
Moving median:

η: 88.51% MOVING_MEDIAN (#Segments : 30)



Max delay:2.854 [sec] $\mu = 2.419$ [sec] $\sigma^2 = 0.16471030$

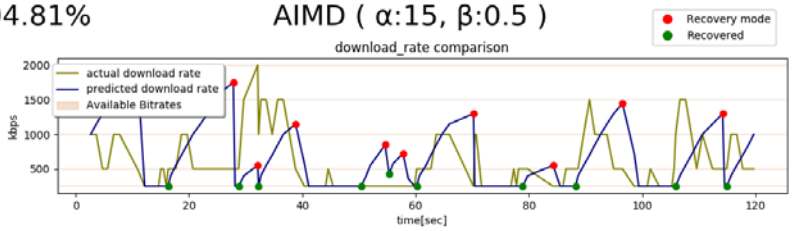
ζ: 77.11%



Quasi-Static -

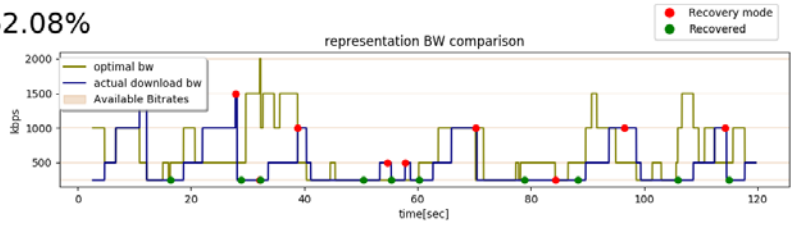
AIMD:

η: 94.81% AIMD (α:15, β:0.5)



Max delay:2.922 [sec] $\mu = 2.169$ [sec] $\sigma^2 = 0.32543362$

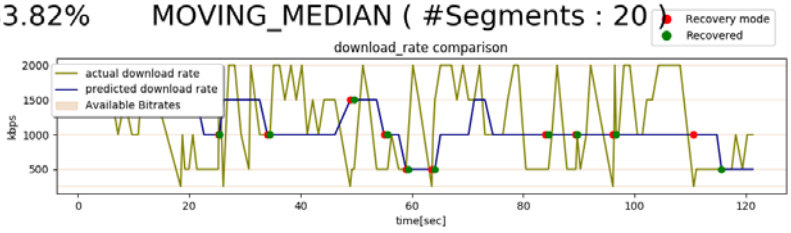
ζ: 62.08%



Moving median:

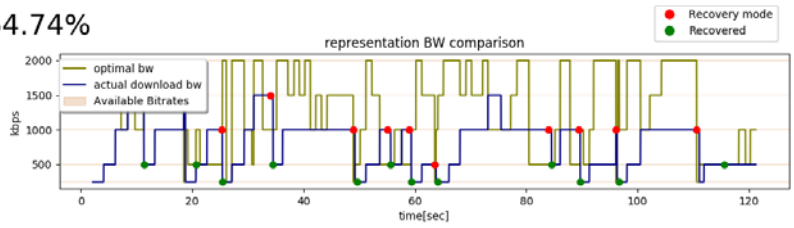
According to η:

η: 83.82% MOVING_MEDIAN (#Segments : 20)



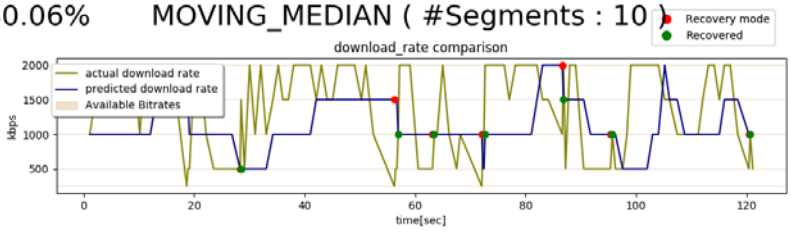
Max delay:2.797 [sec] $\mu = 2.398$ [sec] $\sigma^2 = 0.16184529$

ζ: 64.74%



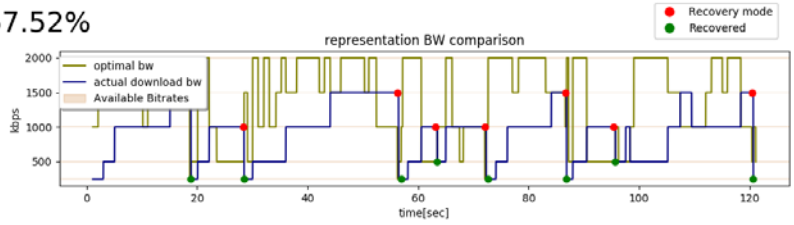
According to ζ:

η: 80.06% MOVING_MEDIAN (#Segments : 10)



Max delay:3.083 [sec] $\mu = 2.737$ [sec] $\sigma^2 = 0.20503182$

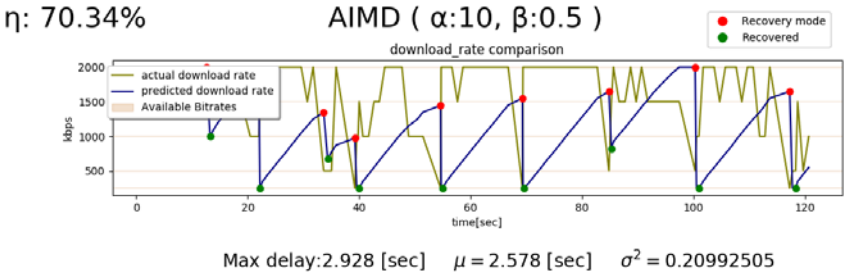
ζ: 67.52%



Dynamic -

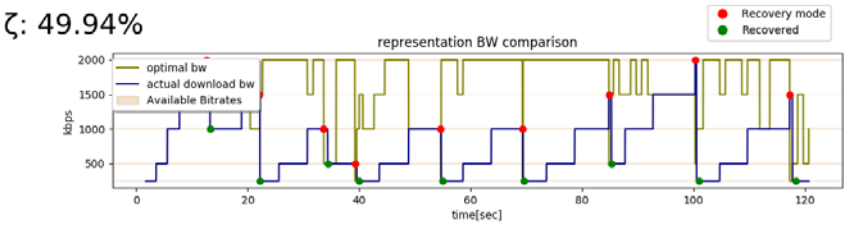
AIMD:

η: 70.34%



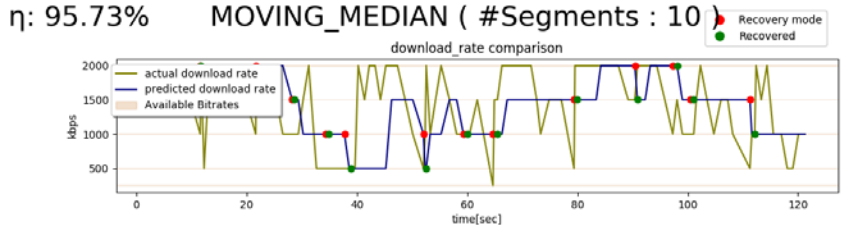
Max delay:2.928 [sec] $\mu = 2.578$ [sec] $\sigma^2 = 0.20992505$

ζ: 49.94%



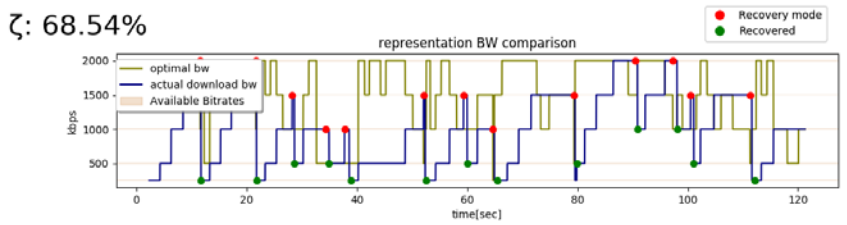
Moving median:

η: 95.73%



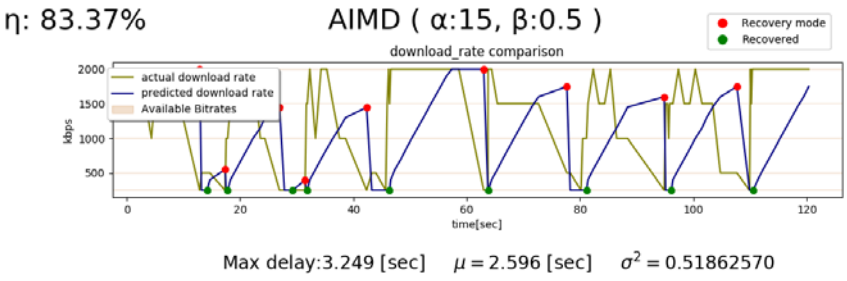
Max delay:3.066 [sec] $\mu = 2.560$ [sec] $\sigma^2 = 0.23920399$

ζ: 68.54%

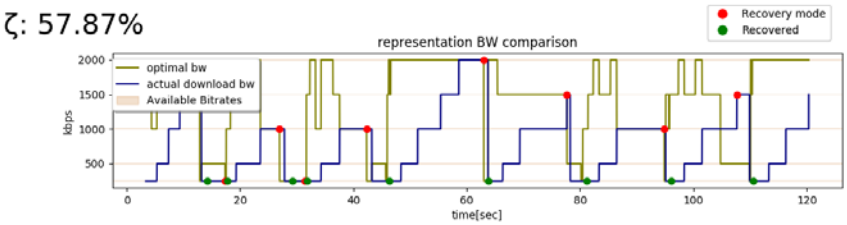


Bursty -

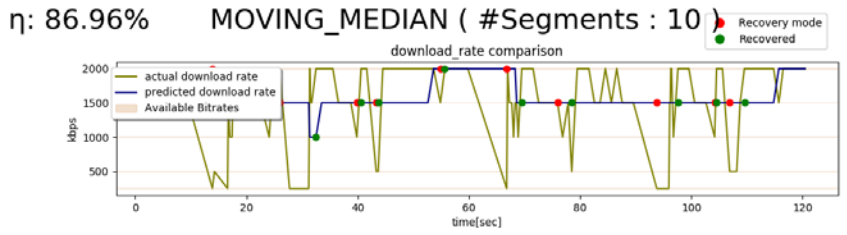
AIMD:



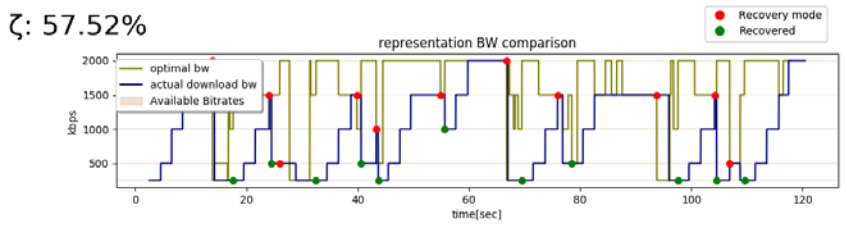
Max delay:3.249 [sec] $\mu = 2.596$ [sec] $\sigma^2 = 0.51862570$



Moving median:

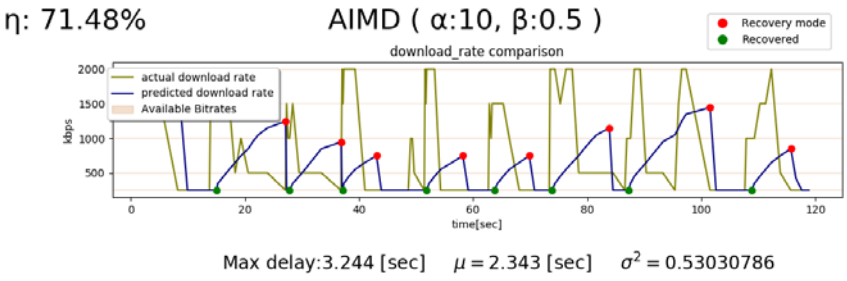


Max delay:2.388 [sec] $\mu = 2.076$ [sec] $\sigma^2 = 0.10857852$

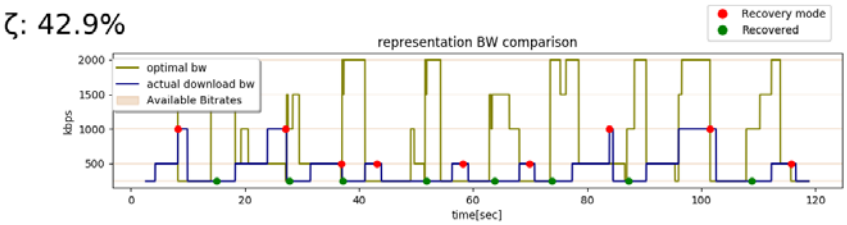


Noisy -

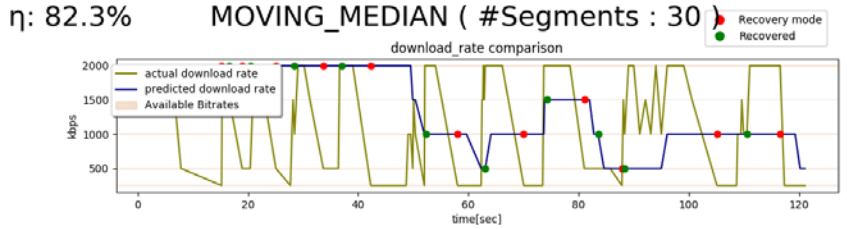
AIMD:



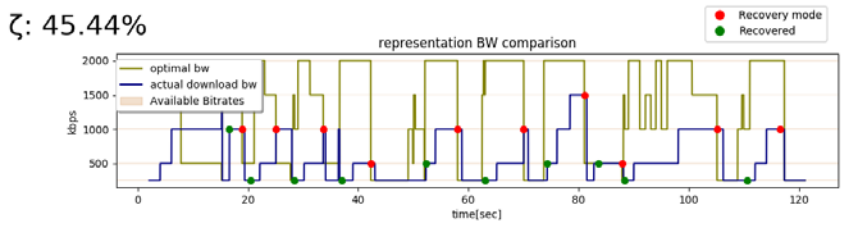
Max delay:3.244 [sec] $\mu = 2.343$ [sec] $\sigma^2 = 0.53030786$



Moving median:



Max delay:2.778 [sec] $\mu = 2.026$ [sec] $\sigma^2 = 0.28990419$



Conclusions

According to the above results, the following conclusions were deduced.

Environmental conclusions

Static - AIMD doesn't learn the network and always tries to improve the quality, hence is more prone to hazards in a static environment, in contrast to Moving median that can learn the network's BW quite accurately.

Quasi-static - This environment behaves a lot like a static environment. There are long periods with a constant BW in which Moving median learns the network's BW. The greater memory the Moving median algorithm has, the longer it will take to the algorithm to learn the new BW. Therefore, lower number of segments are preferred in this environment for a quicker learning process.

Dynamic - AIMD is a conservative algorithm, therefore it is more prone to drops. A dynamic environment changes its BW frequently, thus, every drop result in a long period before returning to the average BW. In contrast to Moving median which learns the network's BW.

Bursty - Each sudden drop in the network's BW leads to a drastic lower prediction, which results in a very long delay before returning to the average BW. Unlike it, moving median manages to learn the network and even after such drops, once recovered, it returns to the average BW rapidly, leading to a better overall performance.

Noisy - In such an environment, none of the algorithms gives decent results. Usually leads to a very poor quality and longer hazards.

General conclusions

- The open source's algorithms didn't fit for live-stream purposes
  - Due to improper regulation on the client's buffer
- Moving-Median adapts to the client's network better
  - Due to learning the network's BW via the download rate history
- Being more conservative lead to sudden changes and to hazards
  - Due to immediate changes that were short lasting, with long recovery
- There is a big gap between the prediction of the BW to the actual BW received
  - Due to limited qualities given by the server
- Drastic changes in the environment leads to longer hazards
- In order to preserve live-stream the requested stream quality is usually substantially lower than the optimal quality that could have been received

Learning & future ideas

Learning

- MPEG-DASH protocol, different algorithms
- Work with open sources and GIT
- Research and Analyze different algorithms in multiple environments
- How to approach a research subject
- Present research results formally

Future ideas

- Improve the research to be more scalable
    - Decentralize the load on the server to multiple servers
- Implement an automatic algorithm controller which will choose the best algorithm in runtime according to the client's network
- Implement a learning method according to the client's network

References

- https://gitlab.cs.technion.ac.il/lccn/w2019-mpeg-dash-livestreaming-unstable-env/tree/master
- https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP
- https://gpac.wp.imt.fr/
- https://nodejs.org/en/
- https://github.com/magnific0/wondershaper
- https://www.ffmpeg.org/