Dynamic Server Selection Strategy for Multi-server HTTP Adaptive Streaming Services

Niels Bouten*†, Maxim Claeys*†, Bert Van Poecke*, Steven Latré†, Filip De Turck*
*Department of Information Technology, Ghent University - iMinds, Technologiepark-Zwijnaarde 15, B-9052 Ghent, Belgium
†Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, B-2020 Antwerp, Belgium
email: niels.bouten@intec.ugent.be

***Abstract*—HTTP Adaptive Streaming (HAS) has become the de facto standard technology for the delivery of video streaming services. Current adaptation heuristics for HAS focus on the selection of the optimal quality representation to be delivered from a single server. However, many content providers use multiple content servers storing replicas of the segmented video or are deployed over Content Delivery Networks (CDNs). Hence, the problem is not limited to selecting the optimal quality but also consists in requesting the segments from the best performing video server. In this paper a dynamic server selection strategy is proposed that enables the streaming client to select the optimal video delivery server. The proposed mechanism allows any quality adaptation algorithm to be plugged into it. The selection algorithm uses probability-based search strategies to explore the search space of available servers and to gain insights in their characteristics. This prevents the selection strategy to end up in a local optimum. To avoid buffer starvations, the exploration behavior is dependent on the current buffer filling. The proposed approach allows to achieve a Quality of Experience (QoE) that is within 25% of the optimum for which the client has a priori knowledge of the server characteristics.**

I. Introduction

Over the past decades, the consumption of multimedia services such as video streaming has increased considerably and is projected to exceed 75 percent of the mobile data traffic by 2020, causing video streaming to dominate the Internet [1]. Popular Over-The-Top (OTT)-services such as YouTube and Netflix are offering large catalogues of user-generated and professionally created video content. Today, the majority of the video streaming traffic is delivered using HTTP and is mainly induced by the advantages offered by HTTP streaming: the reuse of caching infrastructures, the reliable transmission over TCP and the compatibility with firewalls. Furthermore, to increase the scalability of streaming services and to cope with dynamic network conditions, research and academia shifted towards client-side adaptation schemes. Therefore, HTTP Adaptive Streaming (HAS) is now the de facto standard for video streaming delivery.

In HAS, the video content is split temporally into segments which are encoded at different quality rates. The client side quality adaptation heuristic decides at which quality rate each segment should be downloaded, based on measured network statistics, buffer filling level and device characteristics. This allows HAS to respond to throughput fluctuations by reducing the quality and continuing video playout, whereas non-adaptive HTTP-based streaming techniques would run into a buffer starvation. The client is able to independently choose its playback quality, preventing the need for server-side rate adaptation, which is a major advantage in large-scale OTT scenarios.

To increase the scalability of HAS solutions, the video segments are distributed across multiple replica servers in the network. This allows to cope with the instabilities that arise when multiple HAS clients compete for the same bottleneck bandwidth. However, the current adaptation heuristics were designed with a single-server environment in mind and are therefore not suited to be used in such a multi-server environment. They allow the streaming client so select the best quality which optimizes the Quality of Experience (QoE) perceived by the end-user. In a multi-server environment however, the problem also consists in detecting and selecting the optimal server for the segments to be delivered from.

Therefore, this paper proposes a server selection strategy that is able to detect and select the optimal server based on its network characteristics. To achieve this, the selection algorithm uses probability-based search strategies to explore the search space of available servers. By performing passive measurements of the downloaded segments, knowledge on the network characteristics is collected without requiring overhead of active measurements or adaptations to the server. By using ageing-based Exponentially Weighted Moving Average (EWMA) techniques, the proposed strategy takes into account changes to network conditions that may have occured since the last measurement. The probability-based search strategies prevent the selection strategy to end up in a local optimum, while buffer starvations due to wrong server selection are minimized since the probabilities are dependent from the current buffer filling.

The remainder of this paper is structured as follows. In Section II an overview of existing work on single and multi-server HAS adaptation strategies is discussed. The proposed dynamic server selection algorithm is presented in Section III. Section IV discusses the experimental framework and scenarios, as well as a set of experiments optimizing the different configurable parameters of the selection algorithm and applying them in real world scenarios. The paper is concluded in Section V and includes some directions for future research.

978-3-901882-85-2 © 2016 IFIP

II. Related Work

The increased popularity of video consumption over the Internet has led to the development of a range of protocols that allow adaptive video streaming over HTTP. At the client side, each commercial HAS implementation comes with a proprietary client heuristic. Some of the major industrial players have introduced their proprietary protocols such as Microsoft's Silverlight Smooth Streaming[1], Apple's HTTP Live Streaming[2] and Adobe's HTTP Dynamic Streaming[3]. Furthermore, a standardized solution has been proposed by MPEG, called Dynamic Adaptive Streaming over HTTP (DASH) [2].

Several heuristics have been proposed in literature as well, each focussing on a specific deployment. *Miller et al.* propose a receiver-driven adaptation heuristic for DASH that takes into account a history of available throughput and the buffer level [3]. The quality is adjusted to attain a buffer level between certain target thresholds, which improves the stability of the quality and avoids frequent switching as a consequence of short-term throughput variations. *Jiang et al.* identified the problems that arise when multiple clients share a link [4]. The authors propose a variety of techniques that can help avoid such undesirable behavior, such as harmonic bandwidth estimation, stateful and delayed bitrate update and randomized scheduling of requests, which are grouped in the FESTIVE adaptation algorithm. *Tian et al.* show that there is a trade-off between responsiveness and smoothness for client-side DASH adaptations [5]. The proposed rate-switching logic provides a dynamic control of this trade-off according to the trend of the buffer growth. The approach uses machine-learning based TCP throughput prediction to support multiple servers simultaneously.

Liu et al. discuss a video client heuristic that compares the expected segment fetch time with the experienced segment fetch time to ensure a response to bandwidth fluctuations in the network [6], while *Adzic et al.* present a client heuristic which is tailored for mobile environments [7]. All of the aforementioned adaptation heuristics focus on the optimization of the QoE by selecting the optimal quality representation to be delivered from a single server. In a multi-server environment however, the problem also consists in selecting the most appropriate server to deliver the segments from. Our proposed approach allows to select this server at any moment in time, while allowing each of the aforementioned adaptation heuristics to be plugged into it.

Li et al. propose a collaboration scheme between Content Delivery Networks (CDNs) and Internet Service Providers (ISPs) and peer-assisted CDNs to reduce the load on both peering links and internal ISP links [8]. Distributed CDN servers alter the manifests to associate chunks with regional storage servers or by changing or increasing the available video quality levels by transcoding the video. *Famaey et al.* assess the impact of increased latency on QoE caused by the redirection of HAS requests in CDNs and propose updated request routing schemes in order to reduce the number of redirects [9]. *Georgopoulos et al.* propose an OpenFlow-assisted QoE-framework which aims to fairly maximize the QoE among multiple competing clients [10]. The control plane orchestrates the network-wide QoE-fairness. *Petrangeli et al.* propose an OpenFlow controller that is in charge of introducing prioritized delivery of HAS segments, based on the network conditions and the HAS clients' status [11].

Liu et al. propose a probabilistic chunk scheduling approach considering the time-varying bandwidth [12]. The proposed approach is formulated as a constrained optimization problem with the objective to minimize the total download time. *Zhang et al.* present Presto, which is a protocol designed to improve the user experience by providing better fairness, efficiency and stability in the context of multi-server HAS [13]. *Pu et al.* propose adaptation strategies for Scalable Video Coding (SVC) video delivered over CDNs [14]. A collaborative scheduling algorithm is proposed that balances the load among servers, adapts bandwidth dynamics of each server and optimizes the aggregated streaming quality. The aforementioned approaches tightly couple the quality adaptation with the server selection strategy. In our proposed approach, any adaptation heuristic that is desired could be plugged in, independent of the server selection strategy. Furthermore, there are no adaptations required to the server-side implementations. *Lederer et al.* made available a distributed dataset which is compliant to the MPEG-DASH standard and is mirrored at multiple sites across Europe [15]. During the evaluations, part of this video dataset is used, but the content is hosted on our own infrastructure to allow us to control the throughput and delay characteristics of the interconnecting links.

III. Dynamic Server Selection Algorithm

Current HAS quality adaptation heuristics only consider a single server offering the video segments. In a multi-server environment however, the client has the ability to switch to another content server if the streaming quality degrades. Figure 1 illustrates the structure of the dynamic server selection algorithm proposed in this paper. The server to download the next video segment from is selected based on the current buffer filling level and the estimated throughput to each of the servers included in the manifest file. Once the server is selected, its characteristics such as estimated throughput are forwarded to the rate adaptation algorithm, which decides on the quality representation that needs to be requested. This clear separation between server and quality selection allows to plug in any existing rate adaptation logic.

A. Throughput estimation in a multi-server environment

In contrast to existing selection heuristics, there are multiple servers that can be used to download the segments from. Therefore, an instance of the throughput estimation per server is now required. In literature, multiple adaptation heuristics use

[1]Microsoft Smooth Streaming - http://www.iis.net/downloads/microsoft/smooth-streaming

[2]Apple HTTP Live Streaming - http://tools.ietf.org/html/draft-pantos-http-live-streaming-19

[3]Adobe HTTP Dynamic Streaming - http://www.adobe.com/products/hds-dynamic-streaming.html

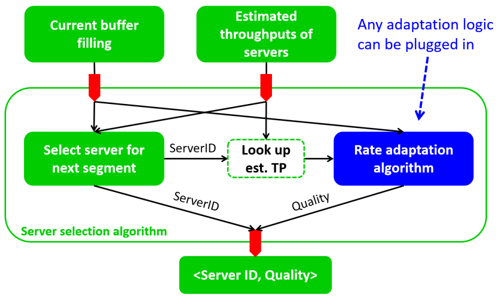
Current buffer filling
Estimated throughputs of servers
Any adaptation logic can be plugged in
Select server for next segment
ServerID
Look up est. TP
Rate adaptation algorithm
ServerID
Quality
Server selection algorithm
<Server ID, Quality>

Fig. 1. Illustration of the dynamic server selection algorithm and its synergy with the rate adaptation algorithm.

EWMA, which uses a weighting factor α ($\in [0,1]$) to define a trade-off between the current measurement (TP_{meas}) or the previous estimation (TP_{t-1}), as illustrated in Equation (1).

$$TP_t = \alpha * TP_{meas} + (1-\alpha) * TP_{t-1} \quad (1)$$

In a multi-server environment, the periods between two consecutive throughput measurements from the same server can be longer if this server is not selected to perform a segment download. This results in less frequent throughput measurements, increasing the probability that the throughput has changed considerably since the last measurement. To reduce the chance that this negatively impacts the decisions, the current measurement value should be favored over the previous estimation. Therefore, the calculation of α is based on the time that has passed between two consecutive measurements ($t_n - t_{n-1}$) and an ageing factor δ:

$$\alpha(t_n, t_{n-1}) = 1 - \exp\left(-\frac{t_n - t_{n-1}}{\delta}\right) \quad (2)$$

The exponential time constant δ determines how fast the smoothing factor α approaches 1 when $t_n - t_{n-1}$ increases.

B. Server selection strategy

As already introduced at the start of this section, the dynamic server selection strategy takes as input the current buffer filling B and the set of estimated throughputs for each server in the manifest. Figure 2 shows the different states and state transitions of the dynamic server selection algorithm. The state transitions depend on the configurable buffer filling levels B_{crit}, B_{high} and B_{max} which respectively indicate the critical buffer level, the high buffer level (indicating the target filling level) and the maximum buffer size. Four states are defined, based on the current buffer filling: the *init* state, the *depleting* state ($B \in [0, B_{crit}[$), the *target* state ($B \in [B_{crit}, B_{high}[$) and the *full* state ($B \in [B_{high}, B_{max}]$). The behavior of the server selection algorithm differs in each of these states, resulting in more conservative behavior when the buffer is depleting and more explorative behavior when the buffer is sufficiently filled.

In the *init* state, the server selection algorithm polls every server that is listed in the manifest file. This is done by downloading a segment from each of these servers and

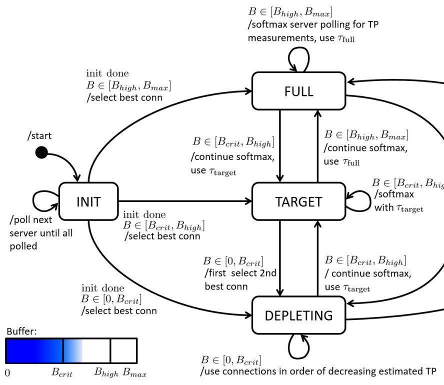
$B \in [B_{high}, B_{max}]$ /softmax server polling for TP measurements, use τ_{full}
init done $B \in [B_{high}, B_{max}]$ /select best conn
FULL
/start
$B \in [B_{crit}, B_{high}]$ /continue softmax, use τ_{target}
$B \in [B_{high}, B_{max}]$ /continue softmax, use τ_{full}
$B \in [0, B_{crit}]$ /first select best conn
$B \in [B_{crit}, B_{high}]$ /softmax with τ_{target}
INIT
TARGET
$B \in [B_{high}, B_{max}]$ /continue softmax, use τ_{full}
/poll next server until all polled
init done $B \in [B_{crit}, B_{high}]$ /select best conn
$B \in [0, B_{crit}]$ /first select 2nd best conn
$B \in [B_{crit}, B_{high}]$ / continue softmax, use τ_{target}
init done $B \in [0, B_{crit}]$ /select best conn
DEPLETING
Buffer:
0
B_{crit}
B_{high}
B_{max}
$B \in [0, B_{crit}]$ /use connections in order of decreasing estimated TP

Fig. 2. Graphical illustration of the server selection strategy showing the different states and state transition policies.

registering the measured throughput. In this way, the algorithm has an initial view of the server space and their respective characteristics. Once every server has been polled, the server selection procedure switches to either the *depleting*, *target* or *full* state, based on the current buffer filling level.

If the buffer filling level B is less than the critical buffer filling level B_{crit}, the algorithm is in the *depleting* state. This indicates that the streaming client is close to running into a buffer starvation and a more prudent strategy should be taken. When this state is entered, the throughput estimates of each server are taken as input and the list of servers is sorted in decreasing order of estimated throughput. The algorithm cycles through this list and downloads segments for each of the servers. If the perceived throughput for the server exceeds the bitrate of the segment, the strategy continues downloading segments from this server. If this is not the case, the next server in the sorted list is selected. When the end of the list is reached, the servers are ordered again using the new information on the estimated throughputs. The algorithm keeps on cycling through the sorted lists until the buffer filling level B exceeds the critical level B_{crit}.

The *target* state is entered when the buffer filling level B is in the interval $[B_{crit}, B_{high}[$. In this state, the server providing the best estimated throughput is selected to download the next segment from. However, since the network conditions can vary over time, continuously selecting the same server could lead the selection strategy to end up in a local optimum. Therefore, the server space is randomly explored using a probability search which is based on the softmax action selection rule, first proposed in the field of reinforcement learning [16]. The Boltzmann distribution is used to determine the probabilities based on the normalized throughputs $\widehat{TP_s} = \frac{TP_s}{\max_{s' \in S}(TP'_s)}$. The probability that a server $s \in S$ is selected, is expressed in Equation (3). The parameter τ represents the temperature and can be configured to either stress exploration or exploitation. A large τ value leads to increased exploration, since the probability of selection is comparable for each of the servers. A small value for τ puts the focus on exploitation, since the differences in probability are much larger. An illustration of

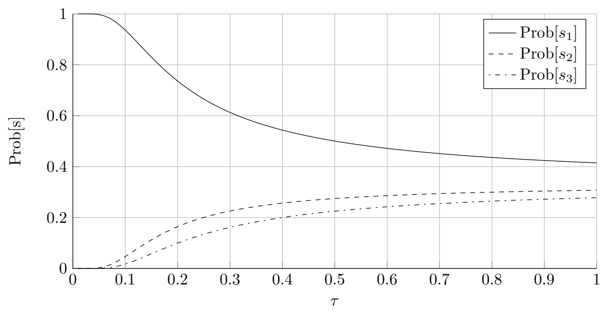

Fig. 3. Illustration of the probabilities subject to various values of τ for 3 servers with normalized throughputs $\widehat{TP_1} = 1$, $\widehat{TP_2} = 0.7$ and $\widehat{TP_3} = 0.6$.

the probabilities as a function of τ for 3 servers s_1, s_2 and s_3 is shown in Figure 3 where the normalized throughputs $\widehat{TP_s}$ are 1, 0.7 and 0.6 respectively. The value of τ used for the *target* state is indicated as τ_{target} and should be set to favor exploitation over exploration, since the buffer is not fully filled and thus the margin for error is not that large.

$$Prob[s] = \frac{\exp \frac{\widehat{TP}}{\tau}}{\sum_{s' \in S} \exp \frac{\widehat{TP_{s'}}}{\tau}} \quad (3)$$

Once the buffer filling level exceeds the B_{high} threshold, the *full* state is entered. The same probabilistic server selection approach is used as in the *target* state. Since the streaming environment is dynamic, it is possible that other streaming origins become better suited than the current one. Therefore, in order to avoid getting stuck in a local optimum, the proposed approach also performs the probabilistic search while in the full state. However, since the margin for error is now larger, the algorithm can afford to take more risk and thus increase the exploration behavior. Therefore the τ_{full} value that is used during the *full* state will be higher, increasing the probability that servers with smaller estimated throughput could also be selected.

IV. Evaluation

This evaluation section is structured as follows. First, the experiment setup is discussed, followed by an elaboration on the metrics that are used during the evaluations. Third, the different configurable parameters δ, τ_{target} and τ_{full} are optimized in terms of reaction time, stability and optimality. Finally, the proposed dynamic server selection heuristic is evaluated using the proposed configurations for various scenarios using 3 or 5 servers and subject to variable bandwidth.

A. Experiment setup

Figure 4 gives an overview of the experimentation framework that was developed to evaluate the proposed dynamic server selection approach. The client is based on the libdash[4] library and is extended to also include the server selection strategy proposed in the previous section. To this end, a

[4]Bitmovin - https://github.com/bitmovin/libdash

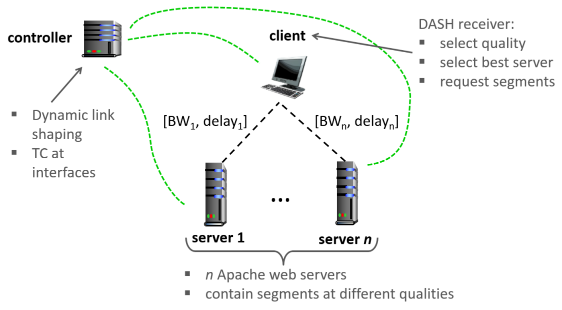

Fig. 4. Evaluation setup showing a client connected through configurable links to multiple servers. All components in the setup are configured by the controller.

TABLE I
Bit rates and corresponding resolution for each of the selected quality levels [15].

Level	Bit rate (kbps)	Resolution
0	250	360p
1	500	480p
2	1200	480p
3	2000	720p
4	4000	1080p

list of bandwidth estimators for each server proposed in the manifest file is maintained. The quality adaptation heuristic is plugged in into the server selection algorithm. During the evaluations the FINEAS algorithm proposed by *Petrangeli et al.* was used [17]. Note that any other adaptation heuristic could be plugged in as well. The buffer levels are based on the evaluations of *Petrangeli et al.* [17] and *Famaey et al.* [18] and are set to be equivalent to those of the quality adaptation heuristic: $B_{crit} = 30\%$ and $B_{high} = 80\%$. The maximum buffer size B_{max} is set to $20s$.

The video segments are hosted onto a set of n Apache[5] HTTP servers. The bandwidth and delay of the interconnecting links are configured using LARTC[6]. The controller uses a configuration file to set the different traffic traces on each link and perform the LARTC shaping on each node simultaneously. Furthermore, this node is responsible for starting and stopping the streaming scenarios and to collect the relevant statistics at the end. The evaluations were conducted using the iMinds iLab.t Virtual Wall infrastructure[7].

The DASH dataset provided by the Information Technology department of the University of Klagenfurt is used as a test set [15]. The content is a video recording of the Red Bull Playstreets, a freestyle skiing competition. Table I lists the quality level, bit rate and resolution for the selected representations. A segment duration of $2s$ was selected. The number of servers offering the video content was varied between 3 and 5, to evaluate the convergence speed subject to an increasing number of servers.

To evaluate the algorithm in scenarios with realistic network

[5]Apache - http://www.apache.org

[6]Linux Advanced Routing and Traffic Control - http://www.lartc.org

[7]iMinds iLab.t Virtual Wall - http://ilabt.iminds.be/iminds-virtualwall-overview

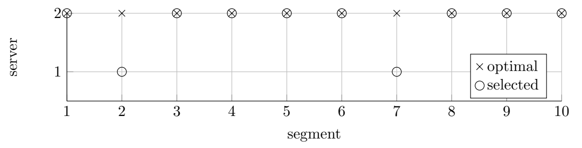

Fig. 5. Illustration of metric $M_{optDownload}$. s_2 is the optimal server, nevertheless, the algorithm selects s_1 two times (i.e., for segment 2 and 7). Hence, $M_{optDownload} = 0.8$.

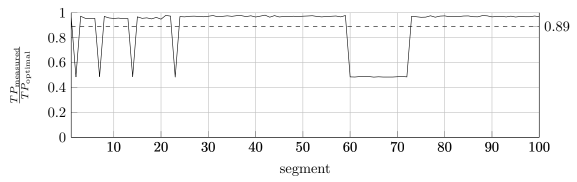

Fig. 6. Illustration of the $M_{TPratio}$ metric where the throughput of both servers is close. The average is indicated by the dashed line: $M_{TPratio} = 0.89$.

behavior, bandwidth traces that represent fluctuations in the network were constructed. The work of *Riiser et al.* is used as a starting point [19]. They present real-world throughput measurements of TCP streaming sessions over the mobile network in Oslo (Norway). Measurements are done using mobile devices while traveling with different types of public transportation (e.g., train, ferry, bus, etc.)[8]. Not all of the available traces are used directly. Some of them are scaled with a factor so that the values are in the same order of magnitude. Furthermore, some traces are shortened because not all traces have the same length. The resulting test traces have a duration of $400s$, which is equivalent to the video duration that is considered during the experiments. Unless otherwise stated, the experiments are repeated 6 times and the average values are shown in the graphs.

B. Evaluation metrics

Several metrics are used to assess the performance of the dynamic server selection algorithm. We will compare the proposed approach with the theoretical optimum which selects the optimal server in terms of bandwidth based on a priori knowledge of the available bandwidth towards each server. Both the proposed and optimal implementation use the same quality selection heuristic. As a consequence, only the multi-server aspect is evaluated, irrespective of the quality selection heuristic's settings.

A first metric that will be used, $M_{optDownload}$, evaluates the fraction of segments that are downloaded from the optimal server. Figure 5 shows an example of how this metric is calculated. The $M_{optDownload}$ metric only awards the selection procedure when the optimal server is selected. However, if the bitrate of two servers is close, there is only a small drop in performance when selecting the lower throughput server.

To account for the aforementioned example, a second metric $M_{TPratio}$ is proposed which evaluates the average ratio of the selected server's throughput ($TP_{measured}$) to the throughput of the optimal server selection strategy ($TP_{optimal}$). Figure 6 shows an example of the $M_{TPratio}$ for a scenario where the throughput of two servers are close to each other.

The previous metrics are agnostic of the actual QoE that is provided to the end-user. Therefore, a third metric evaluating the QoE is expressed as an estimated Mean Opinion Score (MOS)-score. The calculation of this score is based on a QoE-metric defined by *Claeys et al.* [20]. For a video with K segments, N quality levels and played quality level Q_k for each segment k, the estimated MOS is defined as:

$$eMOS = \max\left(5.67 * \mu - 6.72 * \sigma - 4.95 * \phi + 0.17, 0\right) \tag{4}$$

Where μ, σ and ϕ are defined as shown in Equations (5), (6) and (7) respectively. In these equations, F_{freq} and F_{avg} respectively represent the frequency of buffer starvations and the average duration of video freezes.

$$\mu = \frac{\sum_{k=1}^{K} \frac{Q_k}{N}}{K} \tag{5}$$

$$\sigma = \sqrt{\frac{\sum_{k=1}^{K}\left(\frac{Q_k}{N} - \mu\right)^2}{K-1}} \tag{6}$$

$$\phi = \frac{7 * \max\left(\frac{\ln F_{freq}}{6} + 1, 0\right) + \left(\frac{\min\left(F_{avg}, 15\right)}{15}\right)}{8} \tag{7}$$

The third metric M_{MOS} measures the ratio of the estimated MOS obtained by using the proposed server selection algorithm to the estimated MOS of the optimal implementation.

C. Parameter selection

The algorithm proposed in Section III defines several configurable parameters that impact the dynamic server selection algorithm. In this section, the influence of these parameters is evaluated and the optimal parameter configuration is determined. First, the τ parameter of the softmax function that is used in the *target* and *full* state is evaluated. Subsequently, the δ parameter of the aged EWMA throughput estimator is evaluated.

The impact of the temperature parameter τ will be evaluated in terms of the reaction time that is required for the algorithm to detect that another server is outperforming the currently selected server in terms of throughput. This reaction time metric is illustrated in Figure 7. To evaluate the impact of τ on T_{react}, a similar scenario is used, where the throughput of s_2 evolves from half of the throughput of s_1 to twice the throughput of s_1 in the middle of the experiment. Before starting the evaluation, the τ parameter is isolated from the other parameters of the algorithm so that these do not have an influence on the results. Therefore, only one quality level is made available on the servers to avoid an impact of the quality selection algorithm. Figure 8 shows the impact of

[8]Dataset - http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/

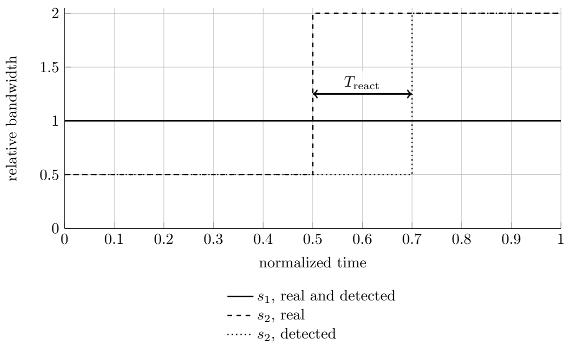

Fig. 7. Applied relative bandwidth traces of two servers, s_1 and s_2, in function of the streaming time. T_{react} is defined as the time that passes between the moment that the throughput of s_2 exceeds the one of s_1 and the moment that the algorithm first notices this. In the example $T_{react} = 0.2$.

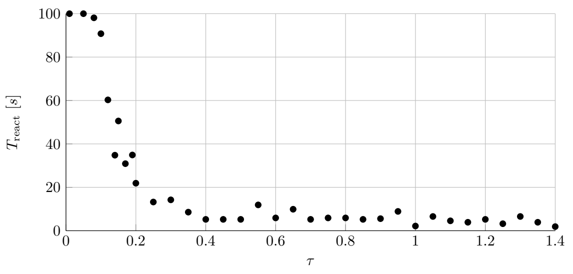

Fig. 8. Reaction time T_{react} subject to temperature parameter τ.

τ on the reaction time T_{react}. There is an inverse relation between τ and T_{react}. Increasing τ, decreases the differences in probabilities for the various servers, and consequently, servers that have a lower throughput will have an increasing probability of being selected. Consequently, the changes in throughput will be noticed much faster as the alternative server will be selected more frequently. If τ exceeds 0.4, the reaction time becomes constant.

However, besides the reaction time, the optimality of the server selection can be impacted by τ as well. Although increasing τ has a positive impact on the reaction time thanks to the increased exploration, it also entails that the optimal server is selected a smaller fraction of the time due to the reduced exploitation. Figure 9 confirms these statements and shows that for increasing values of τ, the fraction of the time that the optimal server is selected decreases, as well as the relative throughput to the optimal server. For $\tau \in [0.0, 0.1]$, both metrics are at a minimum because the jump in the bandwidth of s_2 is never detected. When τ is in $[0.1, 0.2]$, the metrics start to increase. A maximum is obtained when τ equals 0.2. At this point, the algorithm does enough exploration to detect the jump of s_2 without selecting the non-optimal server too frequently. If τ exceeds 0.2, the ratios decrease while the reaction time saturates. The loss in throughput ratio is not compensated by a similar gain in reaction time.

As 0.2 can be considered as the optimal value for τ, τ_{target} is set to 0.2 during the *target* state. In the full state, a larger τ value can be selected to increase the exploration of the server

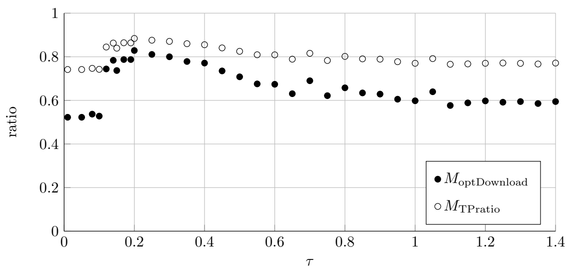

Fig. 9. The average ratio of the measured throughput to the optimal throughput ($M_{TPratio}$) in function of τ, and the fraction of segments that are downloaded from the optimal server ($M_{optDownload}$) in function of τ.

space. Since the buffer filling B exceeds B_{high}, the margin for wrong decisions is larger and τ_{full} can be set to 0.333. The performance for this value is slightly lower than for $\tau = 0.2$ due to the increased exploration, but the reaction times T_{react} are halved.

The second parameter that needs to be configured is the exponential time constant δ of the aged EWMA estimator. A smaller value of δ leads to a higher α and thus a larger weight of the current measurement. There exists a trade-off between how fast changes in throughput are picked up and how well noise is canceled out. Therefore, two scenarios are constructed to evaluate the impact of δ on both aspects.

To evaluate the impact of δ on the reaction time, a scenario is constructed as shown in the first plot of Figure 10 in which TP_{s_2} evolves from $0.5 * TP_{s_1}$ to $2 * TP_{s_1}$ at $10s$. As before, only one quality level is present at the servers (i.e. $500kbps$). This way, the results will not be influenced by the quality selection algorithm. The two other plots of Figure 10 show the impact on the reaction time for $\delta = 0.5$ and $\delta = 30$ respectively. For a low value of δ, a single throughput measurement suffices to detect the increased throughput of s_2. For a higher value of δ, the weight of the last measurement is smaller, requiring multiple measurements (i.e. 3) to detect the increased bandwidth.

Figure 12 gives an overview of the reaction times T_{react} and the required number of measurements to detect the increase, for various values of δ in the interval $[0, 60]$. As expected, T_{react} increases for increasing values of δ.

Another effect of the EWMA estimators is that traffic peaks are smoothed out. Increasing δ increases the weight of the historic estimation, thus decreasing the variation in the estimated throughput. To evaluate the impact of δ on the estimation variability, a scenario is constructed in which s_1 has a larger throughput than s_2 on average, but where the variability of the traces causes the best server to frequently switch between s_1 and s_2. The first plot in Figure 11 illustrates the behavior of both traces over time. The second and third plot show the estimated throughput for $\delta = 0.5$ and $\delta = 30$ respectively. Smaller values of δ lead to a more varying estimation of the available throughput, while larger values even out the traffic spikes.

The standard deviation σ_{TP} is used as a metric to examine

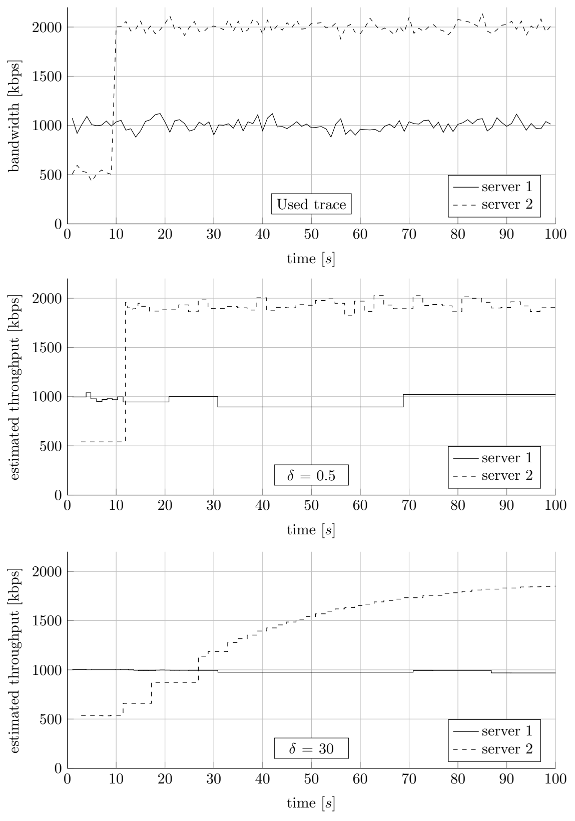

Fig. 10. Impact of δ on reaction time.

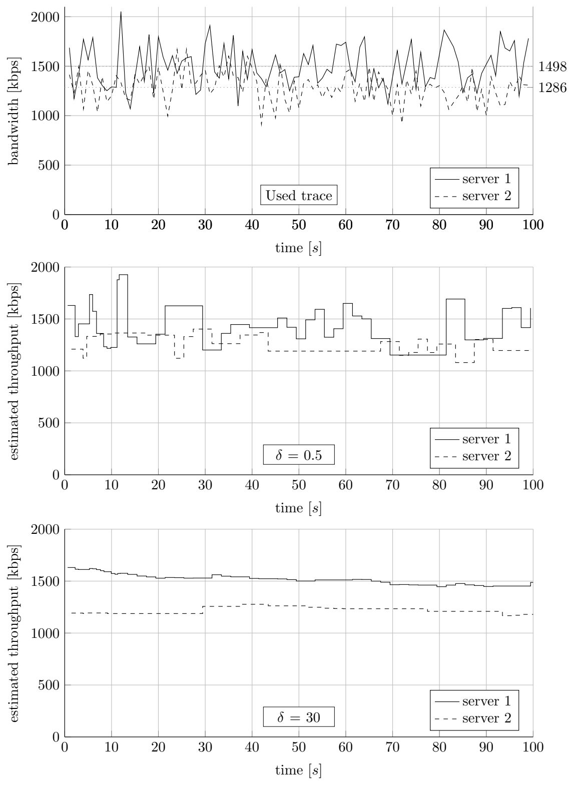

Fig. 11. Impact of δ on stability.

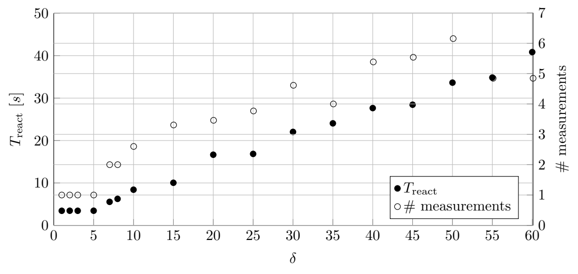

Fig. 12. Time T_{react} and amount of measurements needed to detect the increase of the throughput of s_2 in function of δ.

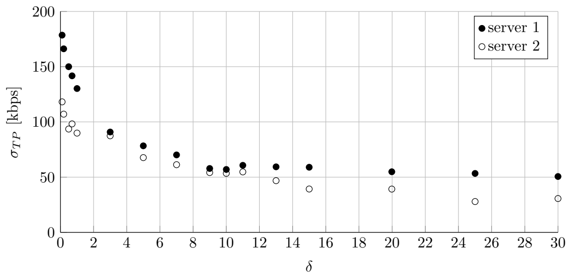

Fig. 13. Standard deviation σ_{TP} of the estimated throughput in function of δ.

the degree of fluctuation in the estimated throughput. Figure 13 shows that there is indeed a decrease in the standard deviation σ_{TP} of the estimated throughput for increasing δ. In the considered scenario, the standard deviation approximately becomes constant when $\delta > 10$.

The cases discussed above are extreme scenarios that were used to clarify the trade-off for the value of δ. The obtained results can differ reasonably in other scenarios. To generalize, both a very small and a rather large value for δ are not advised and will affect the algorithm's behavior negatively. It is not recommended to choose $\delta < 1$ or $\delta > 10$, but there are no strict bounds. In the subsequent evaluations, the value of δ is set to 3, this allows a sufficient smoothing behavior of the estimations, while at the same time reducing the reaction time to detect sudden throughput changes.

D. Server selection performance

Using the optimal parameter configurations that were selected in the previous section, the performance of the proposed approach is evaluated using realistic scenarios with 3 and 5 servers. The first plots of Figure 14 and Figure 15 show the throughput traces for each server that was used during the evaluations for a single run. The second plots show the buffer filling levels, while the third plots show the probability that each server will be selected, based on the softmax policy.

Table II lists the average and standard deviations of the various metrics that were evaluated. These results were obtained over 8 iterations during which the throughput traces are varied. First, we consider the $M_{optDownload}$ metric. Both for 3 and 5 servers, the optimal server is only selected half of the time. It can be seen from the rates in Figure 14 and 15

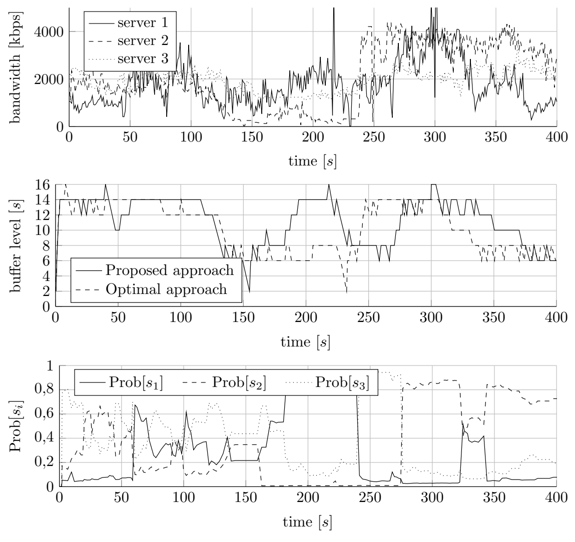

Fig. 14. Evaluation of dynamic server selection for 3 servers, showing the throughput traces, the buffer filling and the selection probabilities.

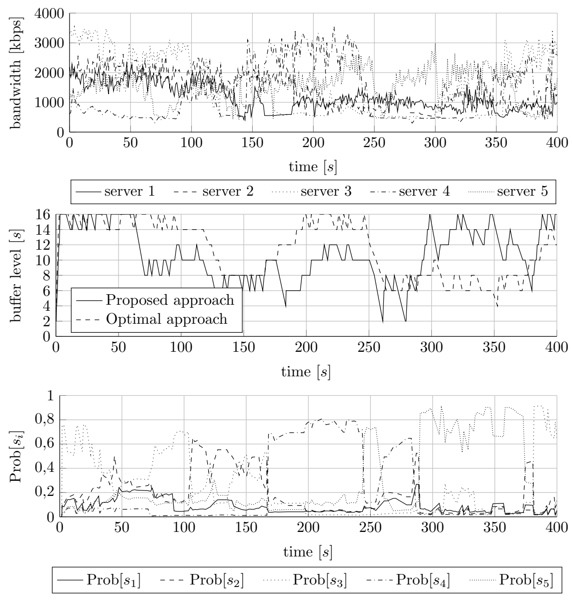

Fig. 15. Evaluation of dynamic server selection for 5 servers, showing the throughput traces, the buffer filling and the selection probabilities.

that for a large fraction of the traces, multiple servers offer a comparable throughput, hence explaining the low values of the $M_{optDownload}$ metric. Looking at the relative throughput that is achieved compared to the optimal throughput (i.e. $M_{TPratio}$), the results show that on average a throughput is achieved that is 81.02% and 74.26% of the optimal throughput for 3 and 5 servers respectively. The value is significantly lower if the number of servers increases. This is due to the larger search space which leads to a higher fraction of the time in which the non-optimal server is selected.

Table II shows a MOS score of 3.20 for the proposed approach and 3.63 for the optimal approach for 3 servers. Consequently, the metric value M_{MOS} amounts to 0.8809. This shows that the proposed algorithm performs quite good as its MOS is only 12% below the optimum. Again, for 5 servers, the proposed approach suffers from the increased search space, showing a M_{MOS} value of 0.7665, which indicates that the achieved MOS is 23% below the optimal achievable MOS. This is mainly caused by a lower streaming quality compared to the optimal solution. Figure 15 shows that the buffer level in the proposed approach is in most cases below the level of the optimal approach. Also, the observed throughput is on average 26% below the optimal throughput (cfr. $M_{TPratio}$). Because the quality selection heuristic takes decisions based on both the buffer level and the estimated throughput, the average quality level of the proposed approach will indeed be lower.

TABLE II
SUMMARY OF OBTAINED METRIC VALUES IN BOTH REAL-WORLD SCENARIOS.

	3 servers		5 servers	
	avg	stdev	avg	stdev
$M_{optDownload}$	0.4925	0.0861	0.4055	0.0564
$M_{TPratio}$	0.8102	0.0195	0.7426	0.0272
MOS_{real}	3.1979	0.2819	2.8501	0.1383
$MOS_{optimal}$	3.6301	0.0168	3.7185	0.0389
M_{MOS}	0.8809	0.0776	0.7665	0.0372

V. Conclusions and Future Work

In this paper, a dynamic server selection strategy is proposed that enables the streaming client to select the optimal video delivery server. The proposed mechanism allows any adaptation algorithm to be plugged into it. Furthermore, no adaptations to the server implementation are required. The selection algorithm uses probability-based search strategies to explore the search space of available servers and to gain insights in their characteristics. This prevents the selection strategy to end up in a local optimum. To avoid buffer starvations, the exploration behavior is dependent from the current buffer filling. The proposed approach is able to select the server of which the throughput is within 25% of the server providing the highest throughput. Furthermore, this allows the proposed strategy to achieve 88% and 76% of the optimal MOS for a scenario of 3 and 5 servers respectively. These results show that the proposed strategy is impacted by the increased search space when the number of servers grows. In future work, mitigation strategies to avoid these scalability issues could be considered.

Acknowledgment

Niels Bouten and Maxim Claeys are funded by a Ph.D. grant of the Agency for Innovation by Science and Technology (IWT). This work was partly funded by FLAMINGO, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme. The research was performed partially within the ICON SHIFT-TV project (under grant agreement no. 140684).

References

[1] Forecast, Cisco VNI, "Cisco visual networking index: Global mobile data traffic forecast update 2012-2017," Cisco Public Information, Tech. Rep., May 2013.

[2] T. Stockhammer, "Dynamic adaptive streaming over HTTP: standards and design principles," in *Proceedings of the second annual ACM conference on Multimedia systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 133–144.

[3] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz, "Adaptation algorithm for adaptive streaming over HTTP," in *Proceedings of the 19th International Packet Video Workshop (PV)*. IEEE, 2012, pp. 173–178.

[4] J. Jiang, V. Sekar, and H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. ACM, 2012, pp. 97–108.

[5] G. Tian and Y. Liu, "Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12, 2012, pp. 109–120.

[6] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj, "Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network," *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 288 – 311, 2012.

[7] V. Adzic, H. Kalva, and B. Furht, "Optimized Adaptive HTTP Streaming for Mobile Devices," in *SPIE Optical Engineering+ Applications*. International Society for Optics and Photonics, 2011, pp. 81 350T–81 350T.

[8] Z. Li, M. Sbai, Y. Hadjadj-Aoul, A. Gravey, D. Alliez, J. Garnier, G. Madec, G. Simon, and K. Singh, "Network friendly video distribution," in *Network of the Future (NOF), 2012 Third International Conference on the*, Nov 2012, pp. 1–8.

[9] J. Famaey, S. Latré, R. van Brandenburg, M. O. van Deventer, and F. De Turck, "On the Impact of Redirection on HTTP Adaptive Streaming Services in Federated CDNs," in *Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security Volume 7943*, ser. AIMS'13, 2013, pp. 13–24.

[10] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide qoe fairness using openflow-assisted adaptive video streaming," in *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, ser. FhMN '13. ACM, 2013, pp. 15–20.

[11] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck, "Software-defined network-based prioritization to avoid video freezes in http adaptive streaming," *International Journal of Network Management*, vol. 26, no. 4, pp. 248–268, 2016.

[12] L. Liu, C. Zhou, X. Zhang, Z. Guo, and C. Li, "Probabilistic chunk scheduling approach in parallel multiple-server dash," in *Visual Communications and Image Processing Conference, 2014 IEEE*, Dec 2014, pp. 5–8.

[13] S. Zhang, B. Li, and B. Li, "Presto: Towards fair and efficient http adaptive streaming from multiple servers," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 6849–6854.

[14] W. Pu, Z. Zou, and C. W. Chen, "Dynamic adaptive streaming over http from multiple content distribution servers," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, Dec 2011, pp. 1–5.

[15] S. Lederer, C. Mueller, C. Timmerer, C. Concolato, J. Le Feuvre, and K. Fliegel, "Distributed dash dataset," in *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 2013, pp. 131–135.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[17] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck, "QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 12, no. 2, pp. 1–24, Oct. 2015.

[18] J. Famaey, S. Latré, N. Bouten, W. Van de Meerssche, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck, "On the merits of SVC-based HTTP Adaptive Streaming," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2013, pp. 419–426.

[19] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*, Feb. 2013, pp. 114–118.

[20] M. Claeys, S. Latré, J. Famaey, and F. De Turck, "Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client," *Communications Letters, IEEE*, vol. 18, no. 4, pp. 716–719, April 2014.