A Resource Allocation Controller for Cloud-based Adaptive Video Streaming

Luca De Cicco, Saverio Mascolo, and Dario Calamita

***Abstract*—Video streaming accounts today for more than half of the global Internet traffic. Content Delivery Networks (CDNs) are employed to provide scalable and reliable video streaming services. Typically, the delivery systems are provisioned to meet the expected peak demands which are due to time-of-day effects. However, such a sizing strategy may either not be able to handle unpredictable flash crowd scenarios, or lead to underutilization of the network with a consequent waste of resources and revenues. Cloud computing offers a way to match the users demand by scaling the allocated resources and by billing the service with a pay-as-you-go pricing. In this paper, we focus on the design of a control plane for cloud-based adaptive video streaming delivery networks. By employing feedback control techniques, we design a dynamical Resource Allocation Controller which throttles the number of virtual machines in a Cloud-based CDN with the goal of minimizing the distribution costs while providing the highest video quality to the user. Results indicate that our resource allocation controller is able to significantly decrease distribution costs and to provide a high video quality to the user.**

***Index Terms*—Cloud computing, adaptive video streaming, resource allocation**

I. Introduction and Related Work

Recent years were characterized by a significant Internet traffic increase, mainly due to the worldwide wideband connections diffusion and to the remarkable growth of multimedia traffic, such as in the case of video streaming [1]. *Content Delivery Networks* (CDNs) are employed to provide a scalable and reliable video streaming services [2].

Typically, the delivery systems are provisioned to meet the expected peak demands which are due to time-of-day effects. However, such a sizing strategy may either not be able to properly handle unpredictable flash crowd scenarios, or lead to underutilization of the network with a consequent waste of resources and revenues.

Cloud computing, with its elasticity, offers a way to match the users demand by dynamically scaling the allocated resources and by using a *pay-as-you-go* charging model without the need of large upfront capital investments.

In this context, the design of an effective control plane for the automatic resource allocation of a Cloud is an open research topic. In [3] authors show that the use of a control plane specifically designed for the video streaming is able to remarkably improve the overall user engagement. In [4], by employing time series forecasting methods to predict the user demand, a bandwidth resource reservation algorithm, along with a load balancer, is designed. In [5] a model for elastic media streaming service over a public Cloud is presented: authors propose the Virtual Content Service Provider (VCSP) which has the role of dynamically renting cloud resources by cloud service providers to adjust the bandwidth resources to the user demand. The algorithm takes explicitly into account the minimization of distribution costs by deciding which type of instance to rent, i.e. small, large, spot.

L. De Cicco and S. Mascolo are with the Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Orabona 4, Italy (e-mail: l.decicco@poliba.it, mascolo@poliba.it), Phone: +390805963851, Fax: +390805963410

D. Calamita was a master student at Politecnico di Bari, Via Orabona 4, Italy

This work has been partially supported by the project Platform for Innovative services in the Future Internet (PON 2007IT161PO006) funded by Italian Ministry of University and Research (MIUR)

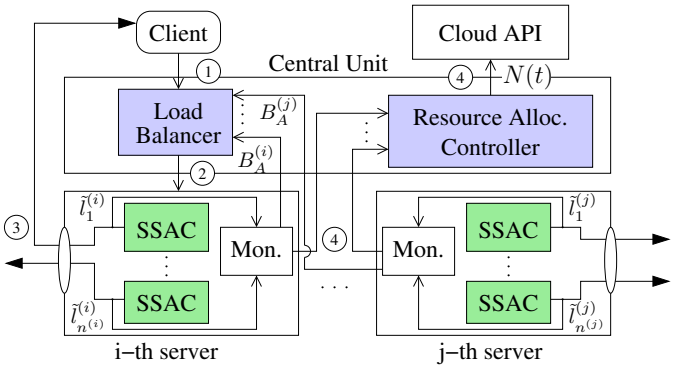


Figure 1: The proposed control plane

In this paper, we focus on the design of a control plane for cloud-based adaptive video streaming delivery networks.

We consider the video *stream-switching* that is getting a wide acceptance, being adopted, among the others, by Akamai, Microsoft, and Apple, and by popular video streaming services such as Netflix, Hulu, Vudu, and Livestream [6], [7]. These algorithms encode the raw video content at increasing bitrates resulting into N versions of the video that form the discrete set $\mathscr{L} = \{l_0, \ldots, l_{N-1}\}$ of video levels; a control algorithm automatically selects a video level $l_i \in \mathscr{L}$ that matches the user's available bandwidth.

In this paper, we employ a feedback control approach to design a dynamical resource allocation controller which throttles the number of virtual machines in a Cloud-based CDN to both minimize the distribution costs and delivery a high video quality to the user.

The rest of the paper is organized as follows: in Section II we propose the control architecture and the dynamic Resource Allocation Controller (RAC); in Section III a performance evaluation is carried out using a discrete event network simulator; finally, Section IV concludes the paper.

II. The Proposed Control Plane

Figure 1 shows the proposed control plane to provide a scalable and adaptive video streaming service that is made

of three components: 1) one *Resource Allocation Controller* (RAC); 2) one *load balancer* (LB); 3) a number of *Stream-Switching Adaptation Controllers* (SSAC). The RAC and LB are centralized, whereas the SSAC is a per-flow algorithm that does not take any input from the RAC or the LB as it is shown in Figure 1.

The overall work-flow can be described as follows: ① the user requests a video; ② the load balancer assigns the request to one of the active servers (see Section II-C); ③ the adaptive video streaming session begins and a SSAC instance is created; ④ periodically, each T_s seconds, every active server provides several feedbacks through a monitor module to the RAC; based on the received feedback , the RAC computes the number $N(t)$ of machines to turn on/off.

Before describing the three algorithms we make the following assumptions: 1) the distribution network is composed of *origin servers,* where all the videos are available, and *surrogate servers,* that store only a fraction of the available videos [2]; 2) all the surrogates are given with the same nominal upstream bandwidth B; 3) the videos are encoded using the same set of video levels $\mathscr{L}$; 4) a centralized server, the *Central Unit* (CU), is able to observe, by periodically probing the nodes, the state of the generic server s comprising the number of active flows, the number of flows obtaining an average video level $\widetilde{l}(t) \in [x, y]$kb/s, and the upstream available bandwidth $B_A^{(s)}$.

In the following we use the function:

$$n(s, \underline{l}, \bar{l}, t) : S \times \mathscr{B} \times \mathscr{B} \times \mathbb{R} \to \mathbb{N} \tag{1}$$

which returns the number of active flows of the server $s \in S$, for which the average video level $\widetilde{l}(t) \in [\underline{l}, \bar{l}] \subseteq \mathscr{B} = \{x \in \mathbb{R} | 0 \leq x \leq l_M\}$, where $l_M = \max \mathscr{L}$. It is worth noticing that $n(s, l, l, t)$ can be used to obtain the number of flows having an average video level equal to l, and $n(s, 0, l_M, t)$ returns the number of active video flows for the streaming server s. For convenience we introduce the notation $n^{(s)}(t) = n(s, 0, l_M, t)$ to refer to the number of active video flows on server s.

A. The stream switching adaptation controller

The SSAC is implemented by employing the *Quality Adaption Controller* (*QAC*) [6], a stream-switching server-side controller designed using a feedback control approach, which throttles the video level $l(t)$ to track the end-to-end available bandwidth. In this paper, we assume that each video is encoded at increasing bitrates resulting into 10 versions of the video that form the discrete set of video levels $\mathscr{L} = \{l_0, \ldots, l_9\}$ with $l_0 < l_1 < \cdots < l_9 = l_M$. The video is sent over a HTTP TCP connection which can be considered as a TCP greedy flow if $l(t) < l_M$. In [6] it is shown that the video level $l(t)$ matches the end-to-end available bandwidth with a transient time of less than 30s and it fairly share the end-to-end available bandwidth with concurrent TCP flows and other QAC video streaming flows. Due to space limitation we are not able to provide a complete description of the algorithm that can be found in [6].

B. The resource allocation controller

In this section we propose the *Resource Allocation Controller* (RAC) which dynamically throttles the number of active servers $M(t)$ to adapt to the users demand load. We focus on scalability with respect to the network bandwidth, since, in a video streaming service the performance bottleneck is mostly due to the available bandwidth within the distribution network and not to the storage or computation resources. The goal of the RAC is to minimize the number $M(t)$ of active servers in our cloud-based CDN to contain costs, while ensuring that the maximum video quality is delivered to the user. We define the *normalized server capacity* as $C = B/l_M$. In other words, C is the maximum number of concurrent video streaming sessions supported by a single server that can obtain the maximum video level l_M. Since QAC video streams are TCP greedy flows, they inherit from the TCP the ability to fairly share the bandwidth and, as a consequence, the video level matches, on average, the available bandwidth [6]. Therefore, when the number of active video streaming sessions on a server is larger than C, the available bandwidth for each connection results less than l_M. This means that QAC will reduce the video level $l(t) \in \mathscr{L}$ to a value lower than l_M in order to match the reduced available bandwidth.

The goal of the controller is to minimize the total number of flows $n_L(t)$ not obtaining the maximum video level, the number of *limited flows*, by throttling the number of active servers $M(t)$. The number of limited flows $n_L(t)$ is the sum of two contributes: 1) the number $n_{UL}(t)$ of *upstream-limited* flows (UL), which is due to the fact that the delivery network upstream bandwidth is saturated; 2) the number $n_{CL}(t)$ of *client-limited* (CL) flows which are due to the fact that the client downlink bandwidth is saturated, i.e. the bottleneck is at the client. Since it is not possible to act on the client-limited flows, the goal of the controller is to steer to zero the number of upstream-limited flows $n_{UL}(t)$.

Towards this end, we need to measure $n_{UL}(t) = n_L(t) - n_{CL}(t)$. However, since there is no way to directly measure $n_{UL}(t)$, in the following we show how to obtain an estimate of the number of upstream-limited flows.

1) Estimating the number of upstream-limited flows: The RAC algorithm employs two thresholds to estimate the number of upstream-limited flows. A *fixed threshold*, defined as $\overline{L} = 0.9 l_M$ is used to estimate the number of flows not obtaining the maximum video level by using the function (1) as $n_L^{(s)}(t) = n(s, 0, \overline{L}, t)$.

In order to obtain an estimate of $n_{CL}^{(s)}(t)$ the RAC uses a *variable threshold*:

$$\underline{L}^{(s)}\left(l_f^{(s)}(t), \alpha^{(s)}(t)\right) = l_f^{(s)}(t) + \alpha^{(s)}(t) \cdot (l_M - l_f^{(s)}(t)) \tag{2}$$

where $l_f^{(s)}(t) = \min(B/n^{(s)}(t), l_M)$ is the *fair video level* that every video should receive in the case $n_{CL}(t) = 0$, and $\alpha(t) \in [0, 1]$ is the fraction of flows sending at the maximum bitrate defined as $\alpha^{(s)}(t) = m^{(s)}(t)/n^{(s)}$, where $m^{(s)}(t) \leq n^{(s)}(t)$ are the streaming session at l_M. It has to be noted that the lower $l_f^{(s)}/l_M$ the more severe is the congestion on the server, since $l_f^{(s)}/l_M = C/n^{(s)}$. Thus, to obtain an estimate of the number of client-limited flows, we count the

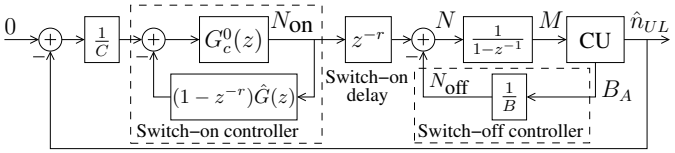


Figure 2: The Resource Allocation Controller block diagram

number of flows that receive an average video level less than $\underline{L}^{(s)}(t)$, i.e. $\hat{n}_{CL}^{(s)}(t) = n(s, 0, \underline{L}^{(s)}(t), t)$. We use the symbol $\hat{n}_{CL}^{(s)}(t)$ to stress that this is an estimate of the actual client-limited flows $n_{CL}^{(s)}(t)$ which is always unknown. Therefore, the estimated number of uplink-limited flows for server s is $\hat{n}_{UL}^{(s)}(t) = n_L^{(s)} - \hat{n}_{CL}^{(s)}(t)$. Finally, the total number of uplink-limited flows is obtained by summing over all the servers, i.e. $\hat{n}_{UL}(t) = \sum_{s \in S} \hat{n}_{UL}^{(s)}(t)$.

2) The controller: In the following $F(z) : \mathbb{C} \to \mathbb{C}$ represents the Z-Transform of the discrete-time signal $f(t) : \mathbb{N} \to \mathbb{R}$, $T_s = 10$s is the sampling interval, and $t_k = kT_s$ indicates the k-th sampling time.

Figure 2 shows the overall system that is made by the following components: 1) a *switch-on controller* $G_c(z)$ which computes $N_{\text{on}}(t_k)$, i.e. the number of servers to turn-on; 2) a *switch-off controller*, which computes $N_{\text{off}}(t_k)$ the number of machines to turn-off; 3) a time-delay block z^{-r} of $r = \lceil \tau / T_s \rceil$ sampling intervals[1] which models the delay to turn on a machine; in this paper the activation delay is $\tau = 30$s, and thus $r = 3$; 4) an integrator block $1/(1 - z^{-1})$ which models the input/output relation between $M(t_k)$, the number of active machines, and $N(t_k)$ the number of machine to turn on/off; 4) the *control unit*, which is provides two outputs, the total available bandwidth $B_A(t_k)$ and the estimated number of the uplink-limited flows $\hat{n}_{UL}(t_k)$ (see Section II-B1); even though we do not provide a model of $\hat{n}_{UL}(t_k)$ it is important to notice that such a block is inverting, i.e. an increase of the input $M(t)$ provokes a decrease of the output, i.e. the number of uplink-limited flows $\hat{n}_{UL}(t)$.

The control algorithm works as follows: at the sampling time t_k, the error $-n_{UL}(t_k)$ is computed and $n_{UL}(t_k)/C$ is given as input to the switch-on controller which gives $N_{\text{on}}(t_k)$; the switch-off controller, based on the total available bandwidth $B_A(t_k)$ computes the number of machines to turn-off $N_{\text{off}}(t_k) = B_A(t_k) \cdot \hat{n}_{UL}(t_k)$; if $N(t_k) = N_{\text{on}}(t_k) - N_{\text{off}}(t_k) > 0$, the controller requests the control unit to switch on $N(t_k)$ machines, otherwise $N(t_k)$ machines are turned off.

The *switch-on* controller is made of a Proportional Derivative (PD) controller plus a Smith predictor. The PD controller transfer function is:

$$G_c^0(z) = K_p + K_d(1 - z^{-1}) \tag{3}$$

where K_p and K_d are strictly negative[2]. The Smith predictor is implemented through the branch $(1 - z^{-r})\hat{G}(z)$: in a nutshell, based on the delay-free model of the plant $\hat{G}(z)$ and on the knowledge of the time delay, it predicts the effect of the delay on the future output of the system and computes a control action that is able to compensate such delay [8], [9]. Here we consider a very simple model for the delay-free transfer function of the plant, i.e. $\hat{G}(z) = 1/(1 - z^{-1})$. This model captures the dominant dynamics of the plant, i.e. the integrator, that is due to the transfer function $M(z)/N(z)$.

C. Load balancing algorithm

The control unit also runs the load balancing algorithm for distributing the user requests to provide a fair load distribution among the servers in terms of bandwidth usage. The CU maintains a table which is updated periodically and it includes the value of the *virtual available bandwidth* (VAB) for each server. The VAB of the server s is computed as $VAB^{(s)}(t) = B - \sum_{j=1}^{n^{(s)}(t)} \hat{b}_j(t)$ where $n^{(s)}(t)$ is the total number of flows on the server s and $\hat{b}_j(t)$ is the virtual bandwidth estimate defined as follows:

$$\hat{b}_j(t) = \begin{cases} b_j(t) & \text{iff j-th flow is steady-state} \\ l_M & \text{iff j-th flow is transient} \end{cases}$$

The CU redirects the user request to the surrogate server with the maximum virtual available bandwidth which has the selected video content in its cache.

In the case the video content is not available in the surrogates, the CU forces the surrogate with the minimum VAB to download the content from the origin server with less active connections. At the same time, the CU redirects the clientrequest to the same origin server.

III. Performance Evaluation

A. Simulation set-up

1) The simulator: With the aim of carrying out a performance evaluation of the proposed control plane, we have modified the simulator CDNsim[3][10] which is based on Omnet++ and the INET framework [11]. In particular, we have introduced the following features which were not available in CDNsim: 1) the adaptive streaming algorithm (QAC) [6]; 2) the dynamic resource allocation controller (RAC) presented in Section II-B; 3) Distribution costs tracking; 4) the load balancing algorithm described in Section II-C. To simulate an elastic cloud, we let the instances to be turned on/off by using a specific signal that can be sent from the control unit to the server.

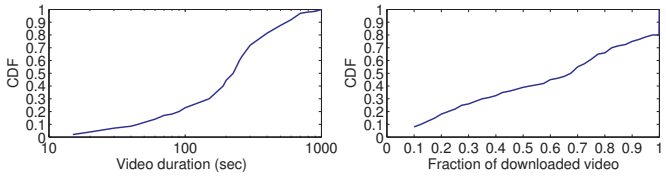
2) Metrics: We measure the following metrics to evaluate the performances of the resource allocation controller: 1) the *number of active machines* $M(t)$; 2) the *fraction of streams obtaining the maximum video level*, i.e. $\alpha(t) = m(t)/n(t)$ where $m(t)$ is the number of flows obtaining the maximum level and $n(t)$ is the number of active streams; $\alpha(t)$ can be considered as an index of the overall quality provided by the delivery network to the users; 3) the *CPU usage costs* that are computed as $C_c(t) = \sum_{s \in S} \gamma^{(s)} \int_0^t a^{(s)}(\tau) d\tau$ where $\gamma^{(s)}$ is the cost rate[4] of the server s which depends on the instance type (i.e. small, medium, large, etc), measured as $ per second, and $a^{(s)}(t)$ is a binary function that is 0 when the machine s is off or 1 when it is on. In this paper we have employed

[1] $\lceil x \rceil$ is the smallest integer not less than x.

[2] The gains are negative since the plant is inverting.

[3] http://oswinds.csd.auth.gr/CDNsim/

[4] In this paper the cost rate $\gamma^{(s)}$ is based on the Amazon's pricing.



(a) (b)

Figure 3: Video duration and fraction of downloaded video

Table I: Request arrival rate $r(t)$

Time	$[0, 200)$	$[200, 400)$	$[400, 600)$	$[600, 800)$	$[800, 1200]$
$r(t)$	20	30	5	30	0

$\gamma^{(s)} = 0.085\$/\text{s}$ for each server, which is the cost rate of an Amazon EC2 medium instance.

3) Workload generation and simulation scenarios: We used the guidelines contained in [12] to generate a realistic workload of user requests. In particular, a total number of 100 videos are available whose popularity are ranked using a Zipf distribution with a parameter $\alpha = 0.8$, the distribution of the video duration and the distribution of the fraction of downloaded video are reported in [12] and shown in Figure 3 (a) and Figure 3 (b) respectively. Since studying the performance of the caching strategy is out of the scope of the paper, in the considered simulation scenarios we make assume that all the servers have all the videos stored in their cache.

The user requests inter-arrival time is modelled using a Poisson process with a variable intensity $r(t)$ as reported in Table I to check to what extent the adaptation algorithm is able to cope with sudden traffic increases and decreases. Furthermore, when the simulation starts the number of active surrogates and origin server is set to 10 and one respectively. Each surrogate has a channel capacity of $B = 100$Mb/s.

Two different scenarios are considered: 1) the clients are connected to the delivery network through 100 Mb/s links; in this scenario the number of client-limited flows $n_{CL}(t)$ is 0; 2) in this scenario we consider the realistic case in which the clients access the delivery network through links at different capacity as reported in [12]; in particular, 40% of the clients have 10 Mb/s bandwidth capacity, 42% have 3.5 Mb/s links, and the remaining 16% are characterized by a downlink capacity of 1 Mb/s.

4) Considered resource allocation controllers: For both the scenarios described above, we compare the performance obtained using the following resource allocation controllers:

1) a *static controller* (SC) which sets the number of server based on the estimated traffic peak value that is supposed the network will manage.
2) A *feed-forward* allocation controller (FF) that dynamically computes the number of machines $N(t_k)$ to turn on/off according to the number of active flows $n(t_k) = \sum_{s \in S} n^{(s)}(t_k)$ using the following control law:

$$N(t_k) = n(t_k)/C - M(t_k). \quad (4)$$

The rationale of (4) is simple: for each sampling time (4) computes the number of machines to be turned on (if $n(t_k)/C > M(t_k)$) or off (if $n(t_k)/C < M(t_k)$) as the difference between the number of machines $n(t_k)/C$ that should be turned on to stream all the $n(t_k)$ flows at the maximum video level, and the current number of machines $M(t_k)$.

3) The proposed *RAC controller* (RAC) with a proportional gain[5] $K_p = -0.7$ and a derivative gain $K_d = -0.3$.
4) The proposed *RAC controller* (RAC no SP) with the Smith predictor action turned off and with gains $K_p = -0.7$ and $K_d = -0.3$.Results

5) Scenario without bandwidth limited clients ($n_{CL}(t) = 0$): We start our investigation by considering that throughout all the duration of the simulation the number of client-limited flows is zero, i.e. $n_{CL}(t) = 0$. Figure 4 shows the results of the simulation.

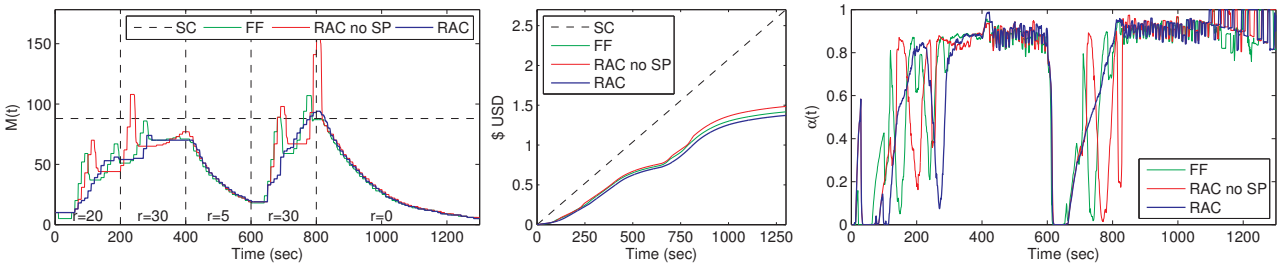
Let us consider Figure 4 (a) that shows the number of active machines $M(t)$ for each of the considered controllers. The RAC exhibits a smooth dynamics of $M(t)$ without the overshoots that can be observed in the case of "FF" and "RAC no SP" after the requests rate increases ($t = 0$s, $t = 200$s, $t = 600$s). The main reason is that both "FF" and "RAC no SP" do not compensate the effect of the delay: thus, when the request rate increases at time $t_{\overline{k}}$ the controllers ask the CU to turn on a number of machines $N(t_{\overline{k}})$; however, the desired effect of the control action, i.e. to decrease $\hat{n}_{UL}(t)$, is delayed of r samples; thus, the next sampling time $t_{\overline{k}+1}$ the controller will not measure a decreased number of uplink-limited flows and, as a consequence, it will compute again a positive control action $N(t_{\overline{k}+1})$ resulting in the overshoot shown in the figure.

Figure 4 (c) shows the fraction of flows $\alpha(t) \in [0, 1]$ obtaining the maximum video level l_M. Let us consider the three transient phases that are triggered at $t = 0$, $t = 200$s, and $t = 600$s when the request rate r increases from 0 to 20req/s, from 20req/s to 30req/s, and from 5req/s to 30req/s respectively. The figure shows that the dynamics of $\alpha(t)$ provided by the two controllers "RAC no SP" and "FF" during the three transients exhibit remarkable oscillations between 0.1 and 0.8 due to the overshoots of $M(t)$ shown in Figure 4 (a). On the other hand, RAC is able to reach the same steady state values of $\alpha(t)$ provided by the other controllers while exhibiting a smooth dynamics. We have computed an average value of $\alpha(t)$ equal to 0.70, 0.69, and 0.74 for RAC, "RAC no SP", and "FF" respectively. It is worth noting that, even though at $t = 600$s $\alpha(t) = 0$ for all the considered controllers, this does not mean that re-buffering phases are occurring. In fact, the adaptation algorithm will sense a decrease of the available bandwidth and will decrease the video level accordingly, avoiding re-buffering [6].

Figure 4 (b) shows the costs, measured in dollars that are due to the CPU usage: when $M(t)$ is provisioned using the peak load, i.e. using the static controller (SC), the highest costs are obtained; the proposed controller is able to achieve up to 42% cost saving *wrt* the static controller and up to 9% *wrt* the other dynamic controllers "FF" and "RAC no SP".

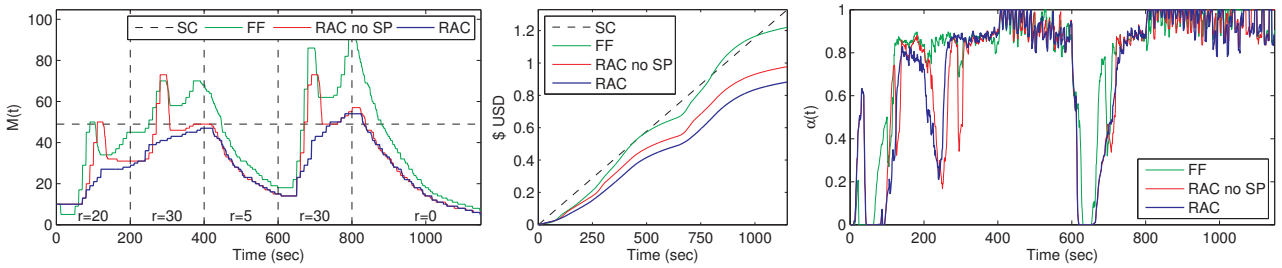
6) Scenario with bandwidth limited clients ($n_{CL}(t) > 0$): In this scenario we consider the realistic case in which the clients access the delivery network through links at different capacity as reported in [12] and already described in Section

[5]The optimal tuning of K_p and K_d will be addressed in future works.



(a) Number of active servers $M(t)$ (b) CPU costs $C_C(t)$ (c) Fraction of streams obtaining the maximum level

Figure 4: Performance of the considered controllers in the case of $n_{CL}(t) = 0$



(a) Number of active servers $M(t)$ (b) CPU costs $C_C(t)$ (c) Fraction of streams obtaining the maximum level

Figure 5: Performance of the considered controllers in the case of $n_{CL}(t) > 0$

III-A3. It is worth to notice that, since in this scenario $n_{CL}(t) > 0$, the total uplink bandwidth required is less than the one required in the case $n_{CL}(t) = 0$. In fact, 16% of the flows have a downlink capacity of 1 Mb/s which does not support the maximum video level l_M. This also means that, on average, we expect a fraction of flows obtaining the maximum video level which is equal to 0.84.

Let us consider the active machines dynamics $M(t)$ which are shown in Figure 5 (a): the proposed controller provides a smooth dynamics of $M(t)$ which is very similar to the one exhibited in the other scenario in which $n_{CL}(t) = 0$; moreover, Figure 5 (c) shows that $\alpha(t)$ is very close to the maximum 0.84; in this scenario the average value of α is equal to 0.73.

The performance provided by "FF" are clearly worse compared with the RAC since a very large number of machines is turned on with the consequence of increasing remarkably the distribution costs; the average value of α is 0.78, which is slightly better than the one provided by RAC; however, this comes at the expense of a 28% increase of the distribution costs *wrt* to the ones required by RAC; (see Figure 5 (b)). The "RAC no SP" controller performs better than "FF" but it exhibits large overshoots compared to the proposed RAC even though they are reach the same steady state; "RAC no SP" provides an average value of α equal to 0.73, which is the same obtained by RAC, but with a 10% increase of the distribution costs *wrt* to RAC.

IV. Conclusions

In this paper we have proposed the *Resource Allocation Controller* (RAC) which automatically throttles the number of active machines in a Cloud-based adaptive streaming delivery system to adapt to the user workload with the goal of obtaining a high video quality while minimizing delivery costs. With the purpose of evaluating its performance, the proposed controller has been implemented in a discrete event simulator and, using a realistic workload, a comparison with two other controllers has been carried out. The results have shown that the proposed controller is able to deliver a high video quality to the users while containing the delivery costs. In particular, it has been shown that the proposed controller is able to save 33%, 28%, and 10% of the distribution costs *wrt* to the static controller, the feed forward controller, and the "RAC no SP" respectively.

References

[1] Cisco, "Cisco Visual Networking Index:Forecast and Methodology 2009-2014," *White Paper*, June 2010.

[2] R. Buyya and M. Pathan, *Content delivery networks*, vol. 9. Springer Verlag, 2008.

[3] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "A case for a coordinated internet video control plane," 2012.

[4] D. Niu, H. Xu, B. Li, and S. Zhao, "Quality-assured cloud bandwidth auto-scaling for video-on-demand applications," in *Proc. of IEEE INFOCOM*, vol. 12, 2012.

[5] J. He, Y. Wen, J. Huang, and D. Wu, "On the cost-qoe trade-off for cloud media streaming under amazon ec2 pricing models,"

[6] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," in *Proc. ACM MMSys*, 2011.

[7] L. De Cicco, S. Mascolo, and C. Abdallah, "An experimental evaluation of akamai adaptive video streaming over hsdpa networks," in *Proc. of IEEE International Symposium on Computer-Aided Control System Design (CACSD)*, pp. 13–18, IEEE, 2011.

[8] O. Smith, "A controller to overcome dead time," *ISA Journal*, vol. 6, no. 2, pp. 28–33, 1959.

[9] S. Mascolo, "Congestion control in high-speed communication networks using the Smith principle," *Special Issue on "Control methods for communication networks" Automatica*, vol. 35, no. 12, pp. 1921–1935, 1999.

[10] K. Stamos, G. Pallis, A. Vakali, D. Katsaros, A. Sidiropoulos, and Y. Manolopoulos, "Cdnsim: A simulation tool for content distribution networks," *ACM TOMACS*, vol. 20, no. 2, p. 10, 2010.

[11] A. Varga *et al.*, "The omnet++ discrete event simulation system," in *Proc. of European Simulation Multiconference (ESM 2001)*, vol. 9, 2001.

[12] J. Summers, T. Brecht, D. Eager, and B. Wong, "Methodologies for generating http streaming video workloads to evaluate web server performance," in *Proc. of SYSTOR '12*, 2012.