Content Delivery System for Optimal VoD Streaming

Sasho Gramatikov, Fernando Jaureguizar, Julián Cabrera, Narciso García
Grupo de Tratamiento de Imágenes, Universidad Politécnica de Madrid, Madrid, Spain
{sgr, fjn, julian.cabrera, narciso }@gti.ssr.upm.es

Abstract— **The demand of video contents has rapidly increased in the past years as a result of the wide deployment of IPTV and the variety of services offered by the network operators. One of the services that has especially become attractive to the customers is real-time video on demand (VoD) because it offers an immediate streaming of a large variety of video contents. The price that the operators have to pay for this convenience is the increased traffic in the networks, which are becoming more congested due to the higher demand for VoD contents and the increased quality of the videos. As a solution, in this paper we propose a hierarchical network system for VoD content delivery in managed networks, which implements redistribution algorithm and a redirection strategy for optimal content distribution within the network core and optimal streaming to the clients. The system monitors the state of the network and the behavior of the users to estimate the demand for the content items and to take the right decision on the appropriate number of replicas and their best positions in the network. The system's objectives are to distribute replicas of the content items in the network in a way that the most demanded contents will have replicas closer to the clients so that it will optimize the network utilization and will improve the users' experience. It also balances the load between the servers concentrating the traffic to the edges of the network.**

I. INTRODUCTION

The wide spread of the IPTV and the advance of the network technologies made a solid ground for offering a real-time video on demand service (VoD). The fact that it enables the clients to watch any video at any time made it a significantly popular service which has been continuously gaining on number of users. This personalized service, however, requires a dedicated unicast flow of data for every request, which significantly increases the traffic in the network. The tendency of rapid growth of the popularity of VoD, its high bandwidth demand and the clients' appetite for higher quality videos and better service has congested the networks and made them a weak point in the realization of the service. Therefore, the network architecture and the strategies for placement of the video contents have become a considerable challenge for many telecommunication operators. One of the initial approaches that has been considered for solving the problem of network congestion is placing servers in various points of the network and caching replicas of certain videos so that they are closer to the users and they can serve more clients. This approach was originally implemented in distribution of web contents in various network architectures [1], but one of the most accepted concepts were the content delivery networks (CDN) [2]. As the videos started to dominate in the internet traffic, the CDNs become a convenient solution for hosting such contents and many of the approaches for distribution of web contents emerged as an acceptable solution. The research in the CDNs was mainly concentrated in solving the replication problem which consists of determining the number of replicas that have to be made for a given video and the servers where to be placed in order to optimize certain cost and quality of service. The replica placement problem has been extensively investigated in the past years. In [3] the replication is set as k-median problem of storing replicas in a manner that a certain cost is optimized. It analyses a static case of user requests pattern and offers a variety of algorithms for replica placement. This solution, however, does not limit the number of request that can be served by one server and does not consider the traffic that is generated while the replicas are being distributed. A more specific formulation of the problem where the quality of service is considered is given in [4]. The dynamic character of the user behavior is taken into consideration in the work presented in [5][6].

Apart from the CDN, there are many other Internet based architectures for partially caching the streaming contents. One such solution is proposed in [7] where the initial parts of the video are cached on proxy servers and the rest of the video is loaded while the prefix is being played. An improved approach of this solution is proposed in [8] where the prefixes of the popular content items are pushed on the clients' side.

Nevertheless, the high traffic demands of VoD services cannot be always satisfied by these architectures because of the uncontrollable and unpredictable character of the Internet. Therefore many VoD solutions move towards development of new architectures in managed networks. These networks are a convenient solution because their size and capacity can be adjusted according to the number of the subscribed users and the traffic can be controlled and spread over the network using technologies that are not possible in the Internet. The IPTV [9] emerged as one of the most implemented solution by many operators. Apart from the main service of multicast linear TV, the IPTV offers more personalized services [10] like VoD and Time-Shifted TV (TSTV) which require dedicated data flow to each request. Because of their growing popularity, the problem of the optimal network utilization and the provision of an improved quality of service is a main concern for many research works. Following this direction, the authors in [11] propose an algorithm for optimal placement of video contents for various IPTV services based on the popularity of the contents without considering the state of the network. Few different

algorithms for replication and placement of VoD contents within a cluster of media servers based on the user request pattern are proposed in [12][13], but they cannot be entirely used in architectures which contain streaming servers in different locations of the network.

In this paper we propose a solution for optimal and efficient content delivery in a managed network. We have developed a new hierarchical content delivery system that implements replication and placement algorithms for redistribution of the contents in the system and a redirection strategy for unicast delivery of content items to the clients. The main objectives of our proposed system are to keep the popular content items close to the clients concentrating the traffic to the outer bands of the network, to reduce the service time, to keep the load balanced among the servers and to optimize the distribution of the replicas among the servers and the clients. We use the replica demand on every server and the state of the network as main parameters for obtaining the above objectives. The model is highly responsive to the user behavior and network conditions, following the dynamic of the popularity of the offered video contents.

The rest of this paper is organized as follows. In section II we describe the proposed system's architecture and the interactions between its entities. In section III we define a set of parameters essential for the system and propose a method for estimation of the replicas' streaming demand. In section IV and V we present the principles of the redistribution algorithm and the redirection strategy. Afterwards, in section VI we present the experimental environment and the results obtained by the simulation. Finally we conclude with a summary in section VII.

II. System Description

For the purpose of optimal video delivery, we developed a network model capable of serving large amount of streaming requests and managing the network according to the users' behavior. It consists of streaming servers responsible for serving the clients and management servers which are responsible for the automatic content distribution and service selection. The streaming servers have a double role: they serve the clients and they deliver content items to other servers in the network. These servers have resources to host and serve only part of the global content items that are offered to the clients. From a structural point of view, the streaming servers are organized as an n-tier hierarchical architecture (Fig. 1). Starting from the top of the hierarchy, which consists of a Central Repository (CR) server, every level downwards is closer to the clients. The CR server contains all the contents that are offered by the operator and unlike the rest of the streaming servers, it does not serve clients. It serves as an entry point for new videos in the system and as an origin point for distribution of replicas to the streaming servers down the hierarchy. The servers store the content items in compressed format and stream the packets with data rate that is sufficient for the client to get uninterrupted video sequence. They provide true streaming [9] i.e. they deliver the packets in real-time and have the capacity to simultaneously serve large number of clients.

The clients in the system use a PC or STB to view the content items. These devices have internal buffer that stores the received packets for decoding and prevents interruptions when there is network congestion. The buffer size will have an important role in viewing an uninterrupted video sequence in congested network conditions, when the packets arrive with big delays and out of order.

The management servers are represented by the Operator, the Automatic Content Movement (ACM) server and the Service Selection (SS) server.

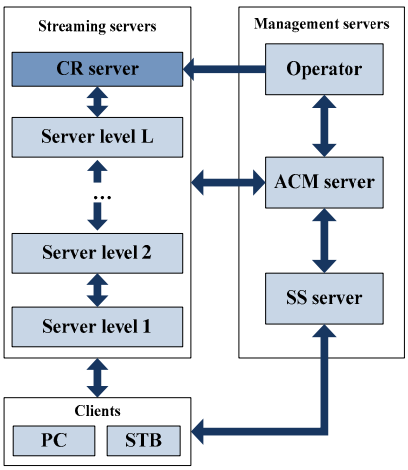

Figure 1. General logic structure of the model

The Operator is an entry point for new contents and also serves for configuration of the system. Anytime a new content item is introduced in the system, it is set on the CR via the Operator and upon the first request from the clients it is pushed to any of the streaming servers.

The ACM server has a central role in the entire system. It communicates with all the servers, monitors the system, takes redistribution decisions and issues commands to the servers. The ACM monitors the state of the network by periodically issuing commands to the streaming servers. Upon reception of the state information from all the streaming servers, it forwards it to the SS server for redirection purposes. Whenever it detects that there are overloaded servers, it runs an algorithm for content redistribution. Using popularity data for the contents in the recent past, previously obtained from the SS server, the algorithm decides whether a replica of a content item should be moved to another server, cloned, removed or left as it is. The execution of the algorithm results with a new distribution of the content items in the system which is deployed by execution of the set of removal, replication and movement commands issued by the ACM server. Along with issuing the commands, the ACM server sends the new availability of the contents to the SS server.

The SS server's role is to accept the clients' requests and to redirect them to the most appropriate streaming server. For every received request, there are three possible situations regarding to the availability of the content item: a replica of the content item exists on some of the streaming servers which is normally loaded, a replica of the content exists on the streaming servers, but they are all overloaded and none of the streaming servers contains a replica. In the first situation, the SS server implements the redirection strategy which chooses the best server. When the best server is chosen, the SS forwards the address to the client, which in turn resends the request to the indicated server. In the second case, the SS server rejects the request and lets the client request the same content

item after a certain time. The process continues until there is a server that can serve the client. In the third case the SS server does not know where to redirect the client because there is no replica of the desired content item.

The whole process of handling the situation when there is no replica on any server is fully shown in Fig. 2, where the numbers attached to the arrows mark the sequence of each action. After the client makes a request for an unavailable content (1), the SS servers asks the ACM server (2) to issue a replication command for the missing content item to the CR server and sends a response to the client (3) indicating it to retry after a time that is long enough for a sufficient part of the content to be provided from the other servers. Because of the large size of the multimedia objects, to avoid long waiting time for complete distribution, the replicas are pushed to the servers with data rate higher than the streaming rate and the streaming is initiated when there is enough buffered streaming data on the server where the replica is pushed. The ACM server chooses the best server to host the replica and issues a push command to the CR server (4). After the delivery of the content has been initiated (5), the ACM server informs the SS server about the new location of the replica (6). Later, when the client resends the request (7), it is redirected to the new streaming server (8). Once the client has the address of the server that can best serve it, it makes a request (9) and immediately initiates a streaming session (10).

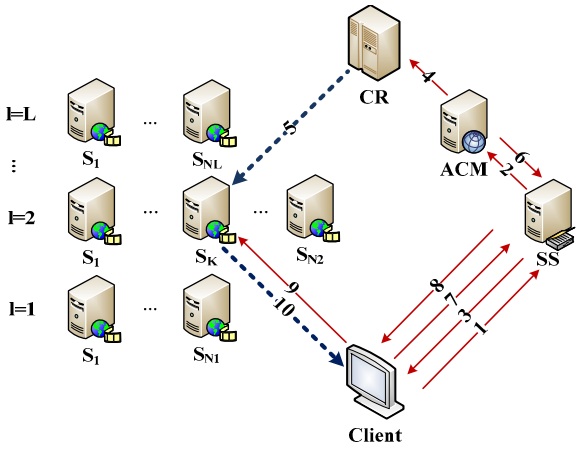

Figure 2. Redirection process for the miss scenario

Having the control over the redirection, the SS server gathers information about the popularity of the replicas on different streaming servers and sends it to the ACM server right before the redistribution algorithm is run. The redirection role of the SS server is such that it optimizes the utilization of the network according to the current placement of the replicas and the current state of the network. The optimal network utilization in this contest is referred to the maximal utilization of the servers at the edge of the network so that most of the traffic generated for serving the client requests is concentrated closer to the clients.

III. SYSTEM PARAMETERS

In this section we define the parameters of the system which we use to estimate the demand of the replicas of the content items on each server and to run the algorithm. We propose an estimation method that achieves to calculate the demand of each replica in the system using only the redirection data obtained from the SS server and the streaming utilization obtained by the streaming servers. This contributes to the reduction of management data in the network and the reduction of the time necessary to obtain it.

The content delivery network that we propose consists of a set of streaming servers S placed in one of the L different levels of the hierarchy. The state of each server $s \in S$ is defined by the streaming and memory utilization. The streaming utilization $u(s)$ is defined as the percentage of up-link streaming capacity $U(s)$ that is occupied for serving the requests of the clients and distribution of replicas to other servers. The value of this parameter lets the ACM server determine whether a redistribution algorithm should be run. An important measure tightly coupled to the triggering of the algorithm, is the utilization trigger threshold $T(s)$ defined as the maximum value of $u(s)$ which can be tolerated for considering the server as normally loaded. Whenever this value is exceeded, the ACM server initiates a procedure for new redistribution in the system. The server storage utilization $m(s)$ is defined as the percentage of the storage capacity $M(s)$ used for hosting the replicas on a given server. The vicinity of the server related to the clients and the other streaming servers is defined by its level within the hierarchy $l(s)$. It can have minimum value 1 if it is directly connected to the clients or value L if it is in the last level of the hierarchy. The only server that has the maximum level is the CR server.

Each server can host one replica from a set of different content items C. Each content item present in the system $c \in C$ has size $s(c)$ and streaming rate $r_s(c)$.

The system also maintains information about the replicas of content items on different streaming servers. The presence information of the replicas is kept in an availability matrix of size $S \times C$ where each element $a(s,c)$ has value 1 if a replica of content item c is present on server s, or 0 otherwise. The local popularity of the replicas is stored in a popularity matrix of the same size, where each element $p(s,c)$ represents the number of times a replica of content item c has been accessed on server s. The popularity information is gathered by the ACM server before the execution of the algorithm and it refers to the activity of the users during the inter-execution interval ΔT, defined as the time between the previous execution of the algorithm and the current execution. In order to control the frequency of the algorithm execution, we also define a minimum inter-execution interval ΔT_{min} which is the minimum time that has to pass between two consecutive executions, no matter the value of the up-link utilization of the servers. The system also keeps the local popularities of the replicas in the previous execution of the algorithm $p'(s,c)$.

In addition, we introduce a global popularity $P(c)$ of the content item c defined as a portion of all the requests in the system during the interval ΔT that belong to its replicas and a minimum level global popularity $P_{Lmin}(l)$ defined as the average value of the minimum global popularities of every server of level l. The later is defined as the global popularity of the first replica that does not enter in the most popular replicas on the server that make 70% of the total load.

In order to quantify the generated traffic between two executions of the algorithm, we introduce a demand parameter $r(s,c)$, defined as an average load on server s,

generated as result of serving requests for the replica of content item c within an inter-execution interval. We calculate the demand as a ratio of the total amount of streamed data $B(s,c)$ sent between two consequent executions of the algorithm and the duration of that interval

$$r(s,c) = \frac{B(s,c)}{\Delta T} = \frac{n(s,c)s(c)}{\Delta T} \qquad (1)$$

where $n(s,c)$ is the number of completed streams for c on s. This number cannot be easily determined by the local popularity $p(s,c)$, because in indicates only how many times the content has been requested within ΔT, but it doesn't say anything about how many of these requests have been completed. In order to determine this number, we propose an estimation method based on the popularity data and the duration of the last inter-execution times.

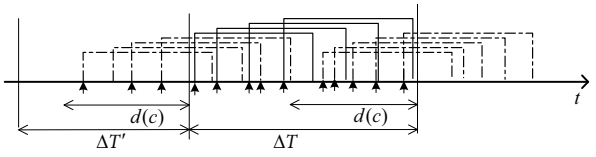

Figure 3. Timeline of streaming a replica from single server

In our analysis we assume that the time of two consecutive executions of the algorithm is such that the streams initiated in one interval will end in the next interval (Fig. 3). In this case, the streaming duration $d(c)$ is less than the inter execution interval and therefore some of the initiated $p(s,c)$ requests will be completed, and some not. The overall number of completed streams will be expressed as

$$n(s,c) = n_c(s,c) + n_p(s,c) + n'_p(s,c), \qquad (2)$$

where $n_c(s,c)$ is the number of initiated and completed streams within the interval ΔT, $n_p(s,c)$ is the number of partially completed streams that were initiated in the interval but were interrupted by the execution of the algorithm and $n'_p(s,c)$ is the number of partially completed streams that were initiated in the previous interval and interrupted by the previous execution of the algorithm. In order to obtain these values we assume that each request in the system occurs according to a Poisson process $N(t)$. Since the requests for a given content item are independent of the requests for other content items, we represent the main Poisson process $N(t)$ as a sum of independent Poisson processes $N_c(t)$ with intensity λ_c where each one represents the process of requests for content item c. We furthermore divide the process $N_c(t)$ as a sum of Poisson processes $N_{sc}(t)$ with intensity $\lambda_{s,c}$ where each process is the request for content item c from server s. Since the expected number of events for a Poisson process within a time interval ΔT is $\lambda_{s,c}\Delta T$, we determine $\lambda_{s,c}$ as the ratio

$$\lambda_{s,c} = \frac{p(s,c)}{\Delta T}. \qquad (3)$$

We also consider that $\lambda_{s,c}$ has a constant value within the interval ΔT, but its value can change after every execution of the algorithm because there might be a different rate of requests due to the possibly different number of replicas for the content items.

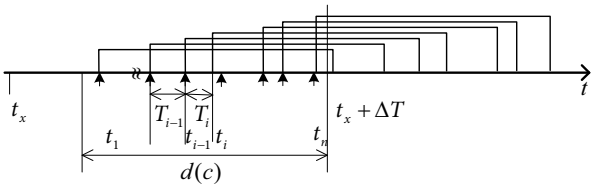

Figure 4. Representation of partially completed streams

The initiated and completed streams within ΔT are those that were requested one content item duration $d(c)$ before the algorithm execution, and therefore their number $n_c(s,c)$ is calculated as the expected number of events within the interval $\Delta T - d$

$$n_c(s,c) = \lambda_{s,c}\left(\Delta T - d\right) = \left(1 - \frac{d}{\Delta T}\right) p(s,c). \quad (4)$$

The number of partially completed streams $n_p(s,c)$ is calculated by finding the completed fractions $f(i)$ of each stream i initiated within the interval $[t_x+\Delta T-d,\ t_x+\Delta T]$ (Fig. 4), defined as

$$f(i) = \frac{d(c) - E[t_i]}{d(c)} \qquad (5)$$

where $E[t_i]$ is the expected time of arrival of the i-th request regarding to the time $t_x+\Delta T-d$. This time can be expressed as a sum of exponentially distributed independent inter-arrival times T_i, which leads to

$$E[t_i] = E\left[\sum_{k=1}^{i} T_k\right] = \sum_{k=1}^{i} E[T_k] = \frac{i}{\lambda_{s,c}} \qquad (6)$$

where $E[T_k]$ is the expected inter-arrival time between the $(k-1)$ -th and k-th request and is expressed as $E[T_k] = \lambda^{-1}$.

By summing the fractions of all initiated and interrupted streams, $q(s,c)=p(s,c)-n_c(s,c)$, and substituting (4)-(6), the number of partially completed streams will be eventually calculated as

$$n_p(s,c) = \sum_{i=1}^{q(s,c)} f(i) = \frac{1}{2}\left(\frac{d(c)}{\Delta T} p(s,c) - 1\right). \quad (7)$$

The number of completed streams interrupted by the previous execution of the algorithm can be determined in a similar way, with the difference that the fraction of each completed stream is now determined as

$$f'(i) = \frac{E'[t_i]}{d} = \frac{i}{\lambda'_{s,c} d(c)} \qquad (8)$$

where $E'[t_i]$ is the expected time of arrival of the i–th request in the previous interval $\Delta T'$. In this case $\lambda'_{s,c}$ is obtained as the ratio $\lambda'_{s,c} = p'(s,c)/\Delta T'$.

ConTEL 2011, ISBN: 978-953-184-152-8

Applying (8) in the sum of fractions for $q'(s,c)=p'(s,c)-n'_c(s,c)$ we obtain

$$n'_p(s,c)=\sum_{i=1}^{q'(s,c)} f'(i)=\frac{1}{2}\left(\frac{d(c)}{\Delta T'}p'(s,c)+1\right). \quad (9)$$

If we substitute (4), (7) and (9) in (3) and then the later in (1), expressing the streaming duration as a ratio between the content size and its rate $d(c)=s(c)/r_s(c)$, we get the average demand for the content items on the streaming servers as

$$r(s,c)=\frac{s^2(c)p'(s,c)}{2\Delta T'\Delta T r_s(c)}+\left(1-\frac{s(c)}{2\Delta T r_s(c)}\right)\frac{s(c)p(s,c)}{\Delta T}. \quad (10)$$

The values $r(s,c)$ for all the replicas stored on the server and the current load of the server $u(s)$, help us determine the percentage of the total stream rate that belongs to each of the replicas. We use this value to calculate the replication metric $m_r(c)$ and the deletion metric $m_d(c)$ for determining the number of new replicas or the number of replicas that have to be removed.

The replication metric $m_r(c)$ is defined as the average amount of load per replica that is generated for serving the requests for content item c. It is calculated according to the following expression

$$m_r(c)=\frac{1}{\sum_{c\in C}a(s,c)}\sum_{s\in S}\frac{r(s,c)u^2(s)U(s)}{\sum_{c\in C}r(s,c)}. \quad (11)$$

In the expression above, the calculated rate per replica server is multiplied by the value of the streaming rate of the server in order to give more weight to the replicas that are placed on more loaded servers. The determination of the number of replicas is related to a threshold stream rate u_o which determines the maximum amount of overload that can be supported by a server in the system from a single replica. Whenever the overload metric reaches this value, new replicas are generated. The value of u_o determines the level of replication in the system. Lower values imply higher sensitivity of the system to the overload traffic which will result in more replicas.

The deletion metric $m_d(c)$ is a measure of useful load that the replicas of a content item c could produce on the servers, but they are in fact occupying memory storage without generating streaming traffic because of its low popularity. Since this metric is intended to serve for deletion of replicas, it takes into consideration the occupied storage space on every server. It is calculated according to the following expression

$$m_d(c)=\sum_{s=1}^{S}\left(\frac{r(s,c)}{\sum_{c=1}^{C}r(s,c)}-\frac{1}{\sum_{c=1}^{C}a(s,c)}\right)m(s)u(s)U(s). \quad (12)$$

In this expression, the locally unpopular replicas of the contents contribute with a negative value in the sum i.e. they have negative contribution to the optimal usage of the server and therefore should be considered for deletion. Therefore, the replicas of the content items that have the most negative value of the deletion should be removed so that the space they occupy is used for storing more popular replicas. The number of replicas that will be removed is determined by dividing the deletion metric by a parameter that determines the maximum allowed underload streaming rate u_u per single replica.

IV. REDISTRIBUTION ALGORITHM

The goal of the redistribution algorithm is to reorganize the system by moving the existing replicas closer to the clients and creating new copies of the contents with increased popularity in order to optimize the network utilization according to the users' behavior. It is run whenever the system detects increase of the streaming utilization of the servers. The redistribution algorithm is executed in three phases: marking replicas for deletion; movement and replication; and unmarking and deletion.

The first phase of the algorithm reserves free storage space for placing future replicas. It marks for deletion the replicas of those content items which have excessive number of copies for their popularity. Therefore, for each content item the algorithm first chooses the number of replicas that could be potentially deleted and then chooses the servers which will have to remove the replica. The algorithm marks for deletion the replicas that are placed on servers lower in the hierarchy, thus reserving more storage space on the servers that are closer to the clients for more popular replicas. This phase provides only "virtual" free space, because although it updates the available storage space, it does not remove the replicas, but only assigns them for potential removal. The final decision for physical removal of the replicas is taken after the replication and movement phase.

In the second phase (Fig. 5), the algorithm attempts to move the existing replicas of a content item as low in the hierarchy as possible and to place new replicas in the levels that remain. It first sorts the content items according to their global popularity so that it can consider the most requested content items first. Then, for each content item (lines 3-16) starting from the upmost level that contains a replica, it selects as an origin server the first server that is close or above its streaming threshold. Afterwards, the algorithm looks for servers in the lowest levels that are underloaded and have enough storage space to host the replica from the upper server. One important condition that the replica has to fulfill is that its global popularity is higher than the minimum global popularity of the chosen level.

After the movement, the algorithm calculates the number of replicas and then searches for the best servers where they can be placed (lines 17-34). It starts from the lowest level in the hierarchy and makes a list of candidate servers which could potentially store a replica. Apart from being underloaded and having sufficient storage space, the candidate servers must belong to a level with minimum global popularity lower than the global popularity of the content item i.e. the content item must be popular enough in order to be placed on the considered level. When the list of candidate servers is completed, the server that is least loaded is chosen as a destination server for the new replica. If there is no server in the considered level that

fulfills the condition, the algorithm goes one level above, until a destination server is found. The process continues until there are no pending replicas left or there are no available resources for storing the pending replicas.

```
1:sortGlobalPopDesc(C)
2:for each c ∈ C
3:    levelUp = maxLevel(c)
4:    levelDown = 1
5:    while levelUp > 0 and levelUp > levelDown
6:        for each s ∈ levelUp
7:            if u(s) > μT(s) and a(s,c)
8:                and exists server d on levelDown such that
9:                  not a(s,c) and u(d) < T(d) and f(d) + s(c) < M(d)
10:                       and P(c) > P_Lmin(levelDown)
11:                 move c from s to d
12:                 u(s) = u(s) − δ(s), u(d) = u(d) + δ(s)
13:                 f(s) = f(s) − s(c), f(d) = f(d) + s(c)
14:           else
15:               levelDown = levelDown + 1
16:   end while
17:   levelUp = levelUp − 1
18:   replicas = round(m_r(c) / u_o)
19:   levelDown = 1
20:   empty candidateServ
21:   while replicas > 0 and level < L
22:       for each s ∈ level
23:           if not a(s,c) and u(s) < T(s) and f(s) + s(c) ≤ M(s)
24:                   and P(c) > P_Lmin(levelDown)
25:               add s in candidateServ
26:       end for
27:       if candidateServ not empty
28:           bestServ = minLoad(candidateServ)
29:           place replica on bestServ
30:           u(s) = u(s) + δ(s), f(s) = f(s) + s(c)
31:           replicas = replicas − 1
32:       else
33:           level = level + 1
34:   end while
35:end for
```

Figure 5. Movement and replication phase

In the movement and replication process, the increase or decrease of the streaming utilization of the servers as a result of the change in distribution is updated by a value predicted according to the current state of the servers and the streaming demand of the content item.

With the proposed strategy, the replicas of the most popular content items are always pushed towards the lowest level, giving a priority to the more popular contents to occupy the lower serves so that when their resources are fully used, the only possibility for placement of replicas of the less popular contents are the servers higher in the hierarchy.

Once the new distribution is determined, the algorithm chooses origin servers that will deliver the replicas to the destination servers. For every destination server assigned to host a replica, the algorithm chooses as an origin server the closest one above in the hierarchy that is not overloaded. The algorithm also takes into consideration the overhead traffic that will be produced for delivery of the replica. It changes the streaming utilization of the origin server by the delivery rate which is always higher than the streaming rate, so that once a portion of it is loaded on the destination server, the requests for that replica can be immediately served.

In the last phase, the algorithm attempts to keep some of the replicas that were marked for deletion so that all the storage space of the server is completely utilized and there is no unnecessary removal of replicas. For every server it unmarks the most requested replicas among the previously marked replicas until there is physical memory space left. All the replicas that cannot be unmarked are permanently deleted from the servers. When this phase is over, the ACM server creates commands and sends them to the streaming servers so that the new distribution is deployed.

V. REDIRECTION STRATEGY

The objective of the redirection strategy is to redirect each request to the appropriate server so that the streaming traffic is concentrated in the lower levels of the network and the traffic between the servers in a same level is equally distributed. Upon a request, the SS server calculates a redirection metric for each server s that contains a replica of the content item c and chooses the one that minimizes the metric. The value of the redirection metric $m_{red}(s,c)$ is calculated as

$$m_{red}(s,c) = \frac{p(s,c)}{1+\sum_{s\in S} a(s,c)\,p(s,c)}(1-\beta(L-l(s)))u(s) \quad (13)$$

where β is the level likelihood factor with values within the interval [0,1/L) which defines the preference of the SS server to redirect the requests to a certain level of the hierarchy. If all the servers have the same value of $u(s)$, the servers in the first level (l=1) will always have smaller value of the metric and will minimize the replication metric. In order to balance the load generated by the replica of the same contents between the servers, the metric includes the percentage of served streams of a single replica by a server relative to the total number of served streams for that content item. Whenever the SS server assigns a server for streaming, it increases its streaming utilization $u(s)$ by the value $r_s(c)/R(s)$. It also predicts a certain reduction of the streaming rate as a result of completion of some of the on-going streams.

VI. SIMULATION RESULTS

In this section we present the experimental results for the proposed model. We developed the model in the discrete event simulator OMNeT++ [14], using the implementation of the network protocols defined in the INET library [15]. We used a network of S=11 streaming servers. Following the architecture of a network for IPTV services [10], we classify the servers according to their vicinity to the clients as edge, branch and central servers. The network contains 6 streaming servers, 3 branch servers and 2 central servers with streaming capacities $U(s)$ of 300, 250 and 200 Mbps and trigger threshold $T(s)$ of 0.9, 0.85 and 0.8. There are 1000 clients that request streaming services from the servers. The CR server hosts C=200 files with average size of $s(c)$=50 MB and streaming rate of $r_s(c)$=2.7 Mbps. All the servers have the same storage space and have the capacity to store an average of 1/6 of the total number of contents. The value of the level likelihood factor is β=0.25.Based on the results obtained from the research on multimedia contents popularity [16][17], our simulations implement a popularity model which obeys a generalized Zipf like distribution, obtained by applying a Zipf-k transformation to the basic form of Zipf distribution. The transformation gives a curved shape to the linear log-log representation of the Zipf distribution with intensity defined by the parameters k_x and k_y, which in our simulation have value $k_x = k_y$ =7 [16].

ConTEL 2011, ISBN: 978-953-184-152-8

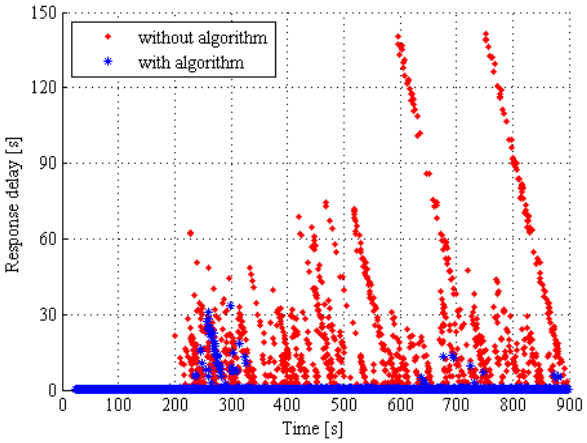

Figure 6. Service response time for the clients

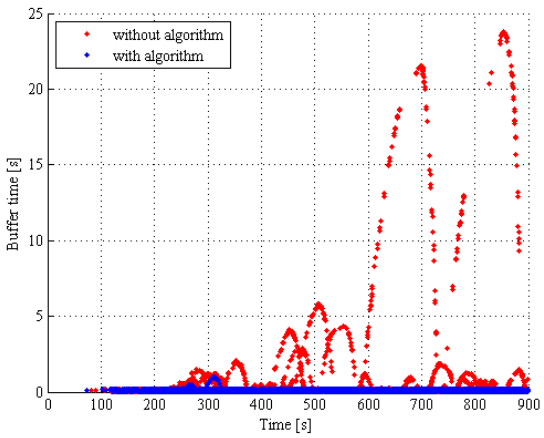

Figure 7. Buffer time

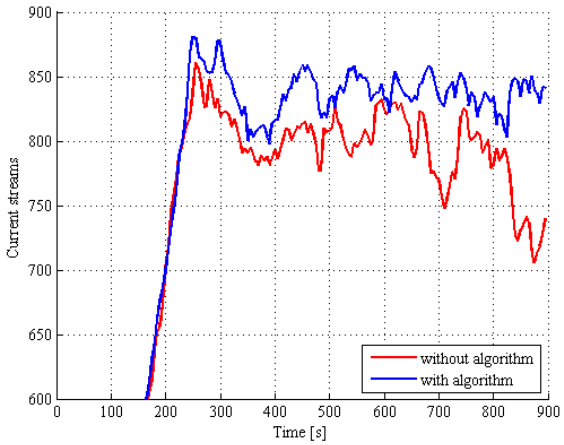

Figure 8. Number of currently served streams

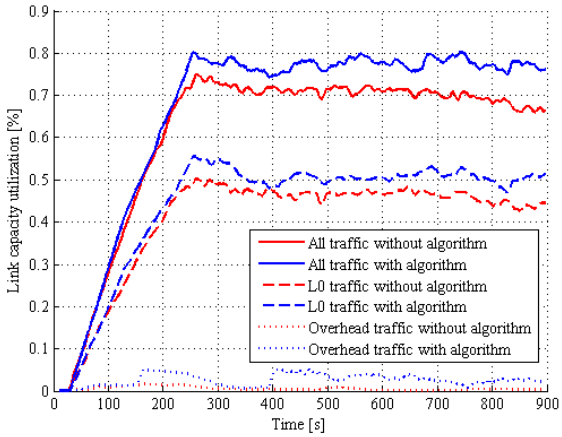

Figure 9. Link capacity utilization

Each client generates a request for a content item within an interval determined according to an exponential distribution with mean value of 30 s. When the streaming of the required content is completed, the client obtains the time of the next request in the same manner.

In order to see the advantages of the employment of the redistribution algorithm, we run the simulation under two different scenarios. In the first scenario we let the algorithm run with a minimum inter-execution interval ΔT_{min}=120 s and in the second scenario we run the simulation without the redistribution algorithm. The files are initially randomly distributed among the servers. Initially there is no traffic in the network and the time of the first request for service of the clients is uniformly distributed between 0 and 250 s. The simulations were run within a period of T=900 s.

Fig. 6 shows the comparison of the time that a client has to wait from the moment it requested the content item until the beginning of the streaming. Each waiting time is presented in the moment when the client has actually made the first request. From the figure it can be noticed that the waiting times follow patterns with linear descending character. This happens because some of the servers become overloaded and therefore reject every request until some of the current streams are finished. Each "line" refers to one overloaded server and its duration is exactly the same as the duration while the server is being overloaded. Obviously, the client that is first rejected has to wait longer time than any other client that makes a request later. The improvement of the system's performance is evident from the comparison: when the algorithm is used, in most of the cases, the clients get immediate service. There can be noted a delay of service at the end of the initial request interval, which is result of the random distribution of the replicas (not all the content items have a replica on the servers) and the rapid saturation of some servers before the algorithm is run. Shortly after the algorithm is run for a first time in the system, the contents are redistributed according to the request pattern and therefore, there is considerably reduced waiting time in the rest of the simulation. In the case when the algorithm is not used, most of the clients have to wait considerable time to obtain the requested stream.

Another measure that we consider for evaluating the quality of service is the time the clients have to wait for buffering enough data for the video to be uninterruptedly played, measured after the streaming of the videos is completed. This quantity is shown on Fig. 7. It can be noted that the highest values of the buffering time are in the intervals when the servers are overload. The algorithm utilization again proves to be of a great advantage, since the buffer time is less than 1 s when the network is most congested, whereas clients have to wait up to 25 s in the cases when the contents are not being redistributed. If this

time is added to the service response time, then it is very obvious that such a service is unacceptable.

Fig. 8 shows the number of clients which are simultaneously served by the streaming servers. It can be seen that in both the cases, the system reaches almost the same maximum number of simultaneous sessions, but unlike the case when the algorithm is used, the absence of redistribution cannot maintain that value. The result of this behavior is less clients served, which in moments differentiate in more than 100 sessions.

Fig. 9 shows several aspects of the link capacity utilization of all the servers in the system. One of the aspects is the overall generated traffic both for streaming and distribution purposes. From the figure we can see that when the algorithm is run, there is considerably more traffic generated in the system compared to the case when the algorithm is not used. One reason for this is that more clients are served because of the balanced load in the network, but another reason is that in this case there is overhead traffic that is generated due to the distribution of the replicas among the streaming servers. Although there is extra traffic in the network, the experiments show that its quantity is less than 5% of the overall traffic. As the algorithm is never executed in the second case, there is only an insignificant amount of overhead traffic at the beginning of the simulation due to the redistribution of the content items that initially have no replica on the servers. The figure also shows that there is higher percent of the overall generated traffic concentrated in the first level, which is one of the main objectives of the algorithm.

VII. CONCLUSIONS

In the presented work we propose a hierarchical system for optimal streaming and distribution of VoD contents in managed networks. The system implements a redistribution algorithm that uses the current demand of the content items and the state of the network to take distribution decisions that optimize the network utilization and improve the quality of service received by the clients. The system additionally implements a redirection strategy which keeps the servers balanced and the traffic to the edges of the network. We also propose an efficient estimation method for determining the streaming demand for the replicas in the systems that reduces the management traffic in the network and the time necessary to obtain the required rate for every replica of the content items in the system.

After the experimental results we prove that the proposed system reaches the defined objectives for improved quality of service and optimal network utilization. It redistributes the content items according to the request pattern and thus, starting from a random distribution of the content items, it achieves a fast convergence to an optimal distribution. The advantages of the optimal distribution and the efficient redirection are numerous: the time a client has to wait for a service is reduced and immediate service is achieved; there is almost uninterrupted streaming which eliminates the necessity of large buffers at client side; the traffic is concentrated to the edges of the network thus providing less congested network and a better utilization of the network resources; and more clients are simultaneously served. The price that has to be paid for these improvements is the overhead traffic generated for distribution of the replicas among the streaming servers. However, this traffic occupies only insignificant part of the overall traffic in the network and is always limited to shortest possible distances between the servers.

REFERENCES

[1] P. Rodriguez, C. Spanner, and E.W. Biersack, "Analysis of Web caching architectures: hierarchical and distributed caching," *IEEE/ACM Trans. Networking*, vol. 9, n. 4, pp. 404-418, 2001.

[2] R. Buyya, M. Pathan, and A. Vakali, *Content Delivery Networks*, ser. Lecture Notes In Electrical Engineering, Berlin, Germany: Springer-Verlag, vol. 9, 2008, pp. 418.

[3] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replicas," in *Proc. IEEE INFOCOM*, 2001, pp.1587-1596.

[4] W. Fu, N. Xiao, and X. Lu, "A Quantitative Survey on QoS-Aware Replica Placement," in *Porc. GCC*, Oct. 2008, pp. 281-286.

[5] F. Lo Presti, C. Petrioli, and C. Vicari, "Distributed Dynamic Replica Placement and Request Redirection in Content Delivery Networks," in *Proc. MASCOTS*, 2007, pp. 366-373.

[6] A. J. Cahill and C. J. Sreenan, "An Efficient CDN Placement Algorithm for the Delivery of High-Quality TV Content." in *Proc. EuroIMSA*, Feb. 2005.

[7] A. Hamra, E.W. Biersack and G. Urvoy-Keller, "Architectural choices for video-on-demand systems," in *Web content caching and distribution*, Norwell, MA: Kluwer Academic Publishers, 2004, pp.129-138.

[8] J. Liu and J. Xu, "Proxy caching for media streaming over the Internet," *IEEE Commun. Mag.*, vol. 42, no. 8, pp. 88- 94, Aug. 2004

[9] W. Simpson and H. Greenfield, *IPTV and Internet Video,* Burlington, MA: Focal Press, 2nd ed, 2009

[10] T. Stockhammer and J. Heiles, "DVB-IPTV Content Download Services-IPTV services anytime and anywhere," in *Proc. IEEE ConTEL*, 2009, pp. 413–420.

[11] M. Verhoeyen, D. De Vleeschauwer and D. Robinson, "Content storage architectures for boosted IPTV service," *Bell Labs Technical Journal*, vol. 13, no. 3, pp. 29-43, 2008.

[12] X. Zhou and C. Xu, "Efficient algorithms of video replication and placement on a cluster of streaming servers*," Journal of Network and Computer Applications*, vol. 30, no. 1, 2007, pp. 551-540, 2007.

[13] J. Dukes and J. Jones, "Dynamic RePacking: a content replication policy for clustered multimedia servers," in *Proc. Microsoft Research Summer Workshop*, 2002.

[14] R. Hornig and A. Varga, "An overview of the OMNeT++ simulation environment," in *Proc. ICST*, 2008.

[15] A. Varga. (2011, Apr 15). *INET Framework for OMNeT++* [Online]. Available:http://inet.omnetpp.org

[16] W. Tang *et al.* "Medisyn: A synthetic streaming media service workload generator," in *Proc. NOSSDAV*, 2003, pp. 12–21.

[17] G.Paneda *et al. "*Popularity analysis of a video-on-demand service with a great variety of content types. Influence of the subject and video characteristics," *Int. J. Adv. Media Commun*., vol. 9, n. 4, pp. 369-385, 2007.