Proceedings of the 2011 Winter Simulation Conference
S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

A CASE FOR VIRTUALIZATION OF CONTENT DELIVERY NETWORKS

André Moreira
Josilene Moreira
Djamel Sadok
Arthur Callado
Moisés Rodrigues
Márcio Neves

Federal University of Pernambuco
Av. Prof. Moraes Rego, 1235
Recife, 50670-901, BRAZIL

Victor Souza
Per P. Karlsson

CDN and Cloud Computing Lab
Ericsson Research
Torshamnsgatan 39A, P.O. BOX 1153
Kista, 164 22, SWEDEN

ABSTRACT

Content Delivery Networks have gained a popular role among application service providers (ASPs) and infrastructural companies. A CDN is an overlay network that gives more control of asset delivery by strategically placing servers closer to the end-user, reducing response time and network congestion. Many strategies have been proposed to deal with aspects inherent to the CDN distribution model. Though mostly very effective, a traditional CDN approach of statically positioned elements often fails to meet quality of experience (QoE) requirements when network conditions suddenly change. In this paper, we introduce the idea of CDN virtualization. The goal is to allow programmatically modification in CDN infrastructure designed for video distribution, adapting it to new operating conditions. We developed a complete simulator focused on CDN overlay network characteristics where we implemented several approaches for each of the CDN elements. Our results show a decrease of 20% in startup delay and network usage.

1 INTRODUCTION

Content Delivery Networks have gained a popular role among application service providers (ASPs) and recently telecom operators such as AT&T. A CDN is an overlay network that gives more control of asset delivery while monitoring network load. It strategically places servers closer to the user, reducing response time and network congestion (Khan and Buyya 2007). When a client makes a request for content, the CDN redirects it to an optimally located mirrored server that should perform transparent and cost effective delivery to the end user. Today, there are a large number of successful commercial CDNs such as Akamai and Limelight, and other non-commercial ones, such as CoDeeN (Wang et al. 2004) and CoralCDN (Freedman et al. 2004) providing services including video distribution.

Many strategies have been proposed to deal with aspects inherent to the CDN distribution model, for example: the placement of surrogates and replicas, redirection mechanisms, cache replacement strategies and accounting. Though mostly very effective, they often fail to meet the high throughput demands of flash crowd events, for example. Furthermore, due to the distributed nature of CDNs and their varying demands, it remains difficult to predict an optimal configuration, for example, a suitable placement of caches given the network topology. This leads to spikes of poor performance, despite an overall good performance. To circumvent such shortcomings, some approaches have been proposed, for example (Chenyu et al., 2006) and P2P assisted CDNs (Sidiropoulos et al. 2008; Padmanabhan and Sripanidkulchai 2002; Lyer et al. 2002). This work introduces and evaluates the benefits of using a Virtualized CDN.

CDN resource virtualization allows to programmatically create, remove or change location of the CDN components, adapting them to new operating conditions. Also, it enables an isolation of the virtual structure, allowing multiple instances of CDNs in a multi-provider scenario.

This work evaluates a virtualized CDN designed for video distribution. We developed a complete simulator focused on CDN overlay network characteristics where we implemented several approaches for each of the CDN elements, supported with strategies for the placement of surrogate servers, request-routing strategies and content outsourcing. By evaluating scenarios specifically created including complex flash-crowd events, generated by ProWGen (Busari and Wiliamson 2002), we show the actual benefits of Virtualized CDNs.

2 RELATED WORK

Coordination is a requirement for the deployment of some new technologies and protocols (for example: differentiated services, IP multicast, secure routing and IPv6). The cost of implementing such a technology is usually not backed up by revenue increase since the service will only work when coordinated among networks. Based on this problem, (Feamster et al. 2007) propose a connectivity model where infrastructure providers are separated from service providers. In this view, the coordination of infrastructure upgrade is made easier since it will not require several entities to agree simultaneously with the change. This decoupling allows each provider to implement its own network architecture and even different architectures for different services, resulting in a virtualization of the network infrastructure.

Using the same approach of virtualization through the separation between service and infrastructure providers, (Zhu et al. 2008) propose a three-layer network model called Cabernet. The infrastructure layer is connected to a service layer through the intermediation of a connectivity layer. This connectivity layer is responsible for providing a single view of the network (multi-provider infrastructure) for the service layer and also for guaranteeing the quality of service requested. Zhu et al provide a discussion on the use of this model for IPTV delivery using the traditional but not widely deployed multicast in the network layer as a means of video distribution, which is much more effective than performing multicast in the application layer.

Another approach for virtualization, proposed by (Carapinha and Jiménez 2009), is the formalization of the basic components of network virtualization as links (physical and virtual ones), nodes (substrate and virtual) and networks. This allows for the creation of fully virtualized network on top of a substrate. The virtual network provider will find the appropriate means for establishing the service requested by the service provider and allocate resources from the infrastructure provider. Carapinha and Jiménez provide a tentative list of which parameters are required for the creation of a link (endpoints and traffic characteristics), for nodes (ID, physical location and CPU/Memory/Storage capacities) and for networks (ID, start time, duration and levels of reliability and preemption).

One approach for the use of network virtualization without altering the business model is the use of Netlets (Völker et al. 2009), which are programmable containers for network protocol stacks. The container is composed of a data plane and a control plane. This allows the rapid reorganization of overlay networks, thus allowing virtualization and fast deployment of novel networks.

An architecture for CDN sharing, called peering, is presented in (Buyya et al. 2006). In this paper, authors cite known proprietary CDN networks and deliver an open, service-oriented architecture that allows inter-CDN communication and Service Level Agreement (SLA) negotiation. However, the paper does not present protocol solutions or implementation results.

Finally, (Plagemann et al. 2006) advocate that the basic operations of a CDN, such as content creation, retrieval, modification and (re)location should be part of the underlying infrastructure, then called Content Networks. Authors assert that the field lacks a thorough common terminology and try to provide one.

To the best of our knowledge, no work detailing Quality of Experience (QoE) parameters under the virtualization of a CDN network was published to date.

3 THE CASE FOR VIRTUALIZATION

Virtualization and particularly OS virtualization has been around since the early days of computing as witnessed by those who developed software for or used mainframe computers in the 70's. More recently, VMWare emerged as a popular technology to offer different OSs over personal computers.

Server level virtualization is an expanding market. Companies such as HostingAce offer Virtual Private Servers (VPS) as a solution for webmasters, designers, developers, and business owners. In the telecom arena, virtual circuits have been seen as logical connections with possibly varying capacities while sharing the same physical circuits. Similar logically isolated tunnels have been used to offer the separation of traffic between concurrent users and increase security. Such concept has been at the heart of communication technologies such as Virtual Private Networks (VPNs), Virtual LANs (VLANs) and MPLS tunneling. A VPN user is then under the false impression that its network is isolated from others and that it has all the network resources for itself. In a way, virtualization is all about concurrent but safe sharing.

Though historically proven, virtualization has not been used on a large scale when it comes to services, mostly due to the research versus production network duality (for a proposal for smoother virtualization experiments in the Internet using PlanetLab, see (Anderson et al. 2005)). But this hopefully is all about to change with the advent of CDNs and the Cloud Computing paradigm. There are some benefits for providing virtualization specific to the application of Content Distribution Networks:

1. The outsourcing of the physical infrastructure should work in the benefit of the CDN providers as they do not have to deal with network failures and management. They nonetheless, need to manage their services in order to ensure customer satisfaction, know when they need to ask for resources or release idle ones.

2. There would be some benefits also if one considers that various service providers may interact with each other and negotiate new services or resources dynamically. One may think of a content provider asking for a feed from another one to cover a new event in a remote place where it has no presence.

3. A CDN provider may negotiate with more than a physical provider and combine their networks to build a single virtual network. This gives the CDN provider independence in choosing its transport provider and the possibility to reach new customers previously not covered.

To summarize, virtualization is a mechanism or a design concept that may be used at different system levels. This flexibility makes virtualization a powerful and hard functionality to fully manage, develop adequately and explore its potentials.

4 CDN ARCHITECTURES

A CDN is a highly complex overlay network composed by a set of elements that acts in a coordinated way to achieve targeted efficiency and QoE for the end-user. Application-specific servers and caches at several places in the network handle the distribution of specific content types (e.g.Web content, streaming media, news information, and real time video) (Khan and Buyya 2007). The basic components of a CDN are:

a) **Origin Server**: represents the actual content owner. It publishes the content in the CDN, by distributing the objects to the surrogate servers. A technical issue that arises here is how the content outsourcing will be made. Content outsourcing is of three types: cooperative push-based, non-cooperative pull-based and cooperative pull-based (Khan and Buyya 2007).

b) **Surrogate**: surrogate or replica servers are caches that store copies of the origin server content. Each surrogate server has a database that contains a list of all available streaming sessions, objects stored in the surrogate and information from the monitor and for the management of CDN. Placement of surrogates in a CDN is closely related to the content delivery process. It is a key issue of a CDN. To solve it, some theoretical approaches have modeled the problem as the "center

placement problem": for the placement of a given number of centers, the question is then to minimize the maximum distance between a node and the nearest center. Some variants of this problem are the facility location problem, the use of k-hierarchically well-separated trees and the minimum K-center problem (Jamin et al. 2000). Due to the computational complexity of these algorithms, some heuristics such as Greedy replica placement (Krishnan, Raz and Shavitt 2000), Hotspot (Qiu et al. 2001), Tree-based (Li et al. 1999) and Topology-informed placement (Jamin et al. 2001) were developed. They take into account existing information from a CDN, such as workload patterns and the network topology.

c) **Redirector**. The Redirector or request-routing system performs the task of estimating the most adequate surrogate server for each different request and client. The most adequate surrogate may not be the "closest" one. Hence, a request-routing system uses a set of metrics such as network proximity, client perceived latency, distance, and replica server load in an attempt to direct users to the closest surrogate that can best serve the request. Simpler redirectors can be implemented by modified DNS or through HTTP rewrite, for example.

d) **Monitor**. Its role is to probe the network and collect data that supports the tasks performed by the redirector and the content manager.

e) **Content Manager**. The content manager controls how the replica objects are stored in each surrogate server. It provides this information to the Redirector, to get each client served by the most suitable surrogate. Also, the information managed by the **Content Locator** is stored in a database (content DB). An important technical design issue is the caching algorithm. It can be performed centralized in the content manager, distributed among the surrogate servers or in a hybrid approach. The caching algorithm must deal with the popularity of the content and somehow predict the changes on the popularity to prepare the CDN for flash crowd events. It also must ensure that at least a single copy of any of its objects must remain present in the CDN at anytime.

f) **Accounting**. CDN providers charge their customers according to the content delivered (i.e., traffic) by their surrogate servers to the clients.

In a virtualized model, the surrogate servers may now be virtual machines instantiated in a Cloud computing environment for example. Thus, new surrogates can be created and added to the CDN infrastructure as operating conditions change. But, most importantly, the surrogates can be repositioned, adapting the infrastructure to new user demands and resource usage states. Thus, during a flash crowd event, for example, it should be possible to replace surrogates to keep them closer to the source of the event, decreasing the startup delay experienced by clients. To accomplish this and other management level decisions and changes, a new component is added: the Virtual CDN Manager (see Figure 1).

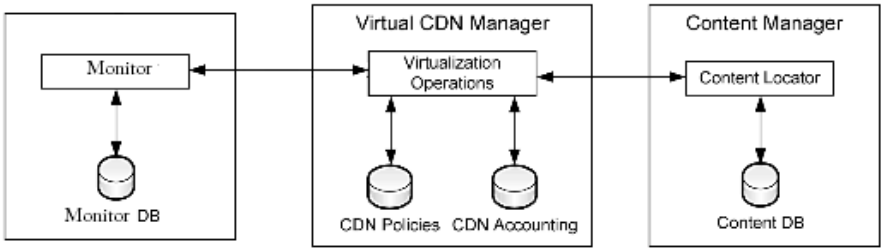

Figure 1: The Virtual CDN Manager

Such manager is responsible for detecting significant changes in network conditions and triggering the necessary CDN resources adaptation. Thus, it may interface with the host physical infrastructure, such as Cloud, in order to allow the dynamic allocation and release of resources (i.e. surrogates). The Virtual CDN Manager receives information from the CDN resource Monitor including important network traffic

metrics, and based on its policies, may therefore decide whether and how to perform adaptation. For instance, if it detects a change in traffic behavior, for example, the presence of a hotspot in the virtual topology that is suffering from a sudden increase in requests, it may move some surrogates around to avoid increase in startup delay or network usage. Such policies may include information such as the granularity of change, rate of monitoring or the desired CDN goals such as: increase availability, decrease startup delay, network usage or inter-AS traffic. Also, there may be restrictions on aspects such as the maximum number of allowed configuration changes that can be made within a given time frame, some surrogates are fixed and cannot be migrated or removed, etc.

The virtual manager is based on the monitoring and processing of information provided by the monitor (probes) to constantly check changes in network condition. This is accomplished by a **utility function** that receives the probes as input to build the network state (**pattern**). This utility function takes into consideration the desired policies and goals of the CDN and weighs the data provided by the monitor. The system periodically calculates the result of the utility function and passes it as input to a **trigger function**. The trigger function, in turn, is responsible for detecting whether the changes over time are significant, if so, the system must perform a pre-established adaptation.

This is achieved by comparing the recent system probes to a baseline that reflects the last system state. If it is above a certain **threshold**, the system would take a decision to adapt. The number of probes (per observation window) in order to be considered as a baseline is a system parameter. They may be weighed according to the following rule: the first probe after an adaptation has a better representation of the baseline system state, thus it must have a higher influence on the baseline configuration. Subsequent probes that form the baseline have their influence decreased by a factor (weight). The same rule applies to the probe window. Figure 2 shows an example of the behavior of the weights.

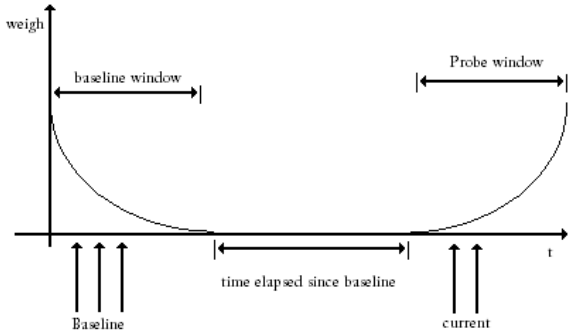

Figure 2: Weight of probe windows

Both baseline and probe window are monitored by the trigger. If there is a significant change in the CDN pattern or behavior, then an adaptation (i.e. replacement of surrogates) is performed. To summarize, the virtualization adopts the algorithm shown below:

```
a. Collect CDN information from DB Monitor (probes)
b. Detect the pattern (utility function from a set of information) and fix it as
baseline
c. Continuously monitor same information and compare it to baseline
   - Information in a set is weighed according to a window function (weigh) and a
window size
   - Windows size and function may differ in baseline and probing
d. If probe value abruptly changes (trigger function) in relation to baseline
   - Adapt topology (replacement)
   - Fix new baseline
```

5 SIMULATION RESULTS

To evaluate some of our CDN resource management ideas, we searched the literature for available tools. There is a notorious lack of tools supporting CDN performance analysis. In fact, to the best of our knowledge, there is a single available CDN simulation environment, CDNSim (Stavrou et al., 2004) that simulates a complete CDN infrastructure. CDNSim is implemented in the C programming language and supports several techniques for server placement, server cooperation, cache organization and request routing management. On the other hand, our experience using it has proved that it is difficult to extend, mainly if considering our requirements for testing the impact of infrastructural changes and new resource management algorithms. As a result, we were faced with the task of implementing a CDN simulator from scratch.

5.1 Simulation tool

Our Simulator was completely implemented in Java 6 and uses the Object-Oriented paradigm to represent the components and interactions of a CDN and network structure. It uses the Desmo-J (Lachler and Page 1999) library that provides a framework to discrete-event modeling and simulation, licensed under the Apache License. The Desmo-J facilitates simulation programming by abstracting the tasks inherent to discrete-event modeling. Also, Desmo-J provides some useful components to help the design of discrete-event simulation, for example, probability distributions and a result reporting system. Our Simulator extended the reporting system to include more detailed information. It presents results in a time series basis, allowing the user to observe specific behaviors that change over time or according to a specific event, for instance, a flash crowd event on a CDN. In addition, these results are plotted with the help of JFreeChart library (Gilbert et al. 2011) and presented in graphics automatically while the simulation is still executing.

5.1.1 Simulator basic functionalities

The structure of the Simulator can be divided into three parts, the core, the network infrastructure and the CDN overlay. Figure 3 shows our simulator architecture.

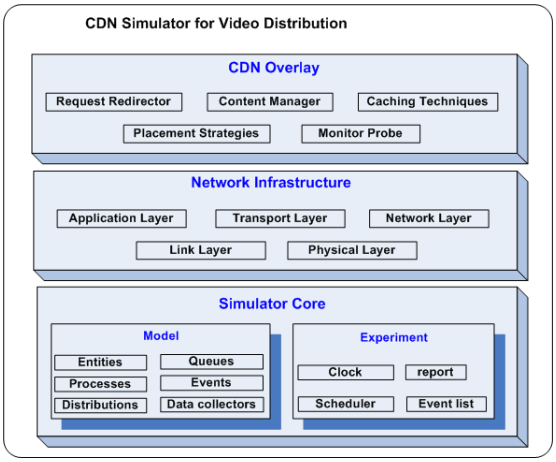

Figure 3: Simulator architecture

The core contains the main components of discrete-event design and it is mainly handled by the Desmo-J API. The second part represents the network infrastructure. It simulates the operation of an entire network by reading topology configuration files (generated by a topology generating tool such as INET, for instance) and re-creating a topology which is defined by a set of nodes and links. The network configuration file provides information on how the nodes are linked as well as the parameters for the links, such as delay, type of the queue, buffer size and speed. On each node, there is a network stack composed

by a network layer that performs routing; the transport layer that actually implements transport protocols including TCP and UDP; and the application layer that handles the processes that run on nodes.

The adopted routing mechanism is very simple. Packets are routed according to a router table provided in a configuration file or created at runtime based on the shortest path algorithm between nodes. Each of these layers can also be extended to add new functionalities such as a more advanced routing mechanism or a different implementation of a transport protocol.

Our simulator takes advantage of these network level features as these aspects are really simulated, i.e., packets are actually routed through the routers, and characteristics as link capacity, delay and congestion are truly captured by the simulator. For example, in a flash-crowd scenario the link saturation really occurs and then we are able to measure packet losses. Similarly, for video distribution, we provide a realistic measurement of the start-up delay.

An important module within our Simulator lies at the application layer. It is designed to facilitate the implementation of new applications by emulating a Java network programming environment. Applications that run on top of nodes can be created by inheriting an application process class and implementing the run method. This super class provides some helpful functionalities such as sockets, that are very similar to Java Sockets. Thus, other network overlay, besides CDN, can also be implemented and simulated. This has been very useful for our development as we were to evaluate separately a number of CDN types such as pure CDNs, Peer-to-Peer CDNs and Hybrid CDNs. The result is a considerably plug and play CDN dependent software overlay that makes use of a common simulation framework hence saving development time effort and increasing reuse.

The CDN overlay is designed for video distribution. It is composed by basic CDN components: the origin server, surrogates, a request redirector, a content distribution manager, a monitor and clients. All of these elements are created as application processes that run on top of the network nodes. They are coordinated by a main component that operates the CDN according to interchangeable strategies and algorithms.

To extend the simulator, new strategies can be created by simply implementing a specific interface. For instance, our simulator has an interface for the placement strategy of surrogates. Such strategy is executed at the beginning of a simulation and its main task is to place all surrogates on network topology. The surrogates, in turn, must use a specific caching replacement technique and a specific cache size. All other components are instantiated in the same manner, and all their associated parameters must be specified in the CDN´s configuration file. Thus, different CDN configurations can be simulated by creating new strategies and then adjusting the configuration file accordingly.

5.1.2 CDN Operation

Simulation starts when a client requests an object. The coordinator instantiates the client at the appropriate time and activates its thread. Firstly, the client contacts the request redirector to discover which surrogate will serve its request. Then, the selected surrogate starts streaming the video object to the client. Finally, when download is finished, the coordinator allows the client to release memory. The definition of requests, video objects and where the client will be placed in network are specified by configuration files and the distributions therein. With the exception of the requests file, they can also be established at runtime, according to a specific probability distribution.

5.1.3 Virtualization

The virtualization model implemented is very simple. The window size for both baseline and window is 1, thus the weight based function is simplified to an identity function. The target of the system is to decrease network usage, so the main metric monitored by the monitor is the traffic at the nodes. The adapta-

tion is performed by replacing the surrogates over the available nodes when a significant network state (pattern) change is detected.

The replacement is achieved in the same way as for the placement: using one of the placement algorithms implemented in our simulator. In some previous experiments, when we compared those, we verified that the HotSpot with radius 1 (number of hops) has a similar performance to that of the Greedy algorithm, but at a lower computational cost. Thus, we used the Hotspot strategy in our simulations. The Hotspot algorithm attempts to place replicas near the clients generating the greatest load within a vicinity, which is defined as a circle centered on a node and some radius. The placement algorithm uses the information regarding the last requests seen to decide where to setup a surrogate. However, old requests information, i.e. those before a stipulated amount of time, is discarded to avoid interference from an old behavior that no longer represents the system state.

All the information regarding traffic in the system provided by the monitor is mapped into a vector. This represents the load on each link being monitored. The vector is then normalized and the utility function is defined as the dot product between the vectors from baseline and current probes. If the dot product is bellow an established configurable threshold, the replacement is triggered. The dot product can be understood as similarity between baseline and current vectors. The closer it is to one, the less change there was in traffic behavior as shown in Figure 4.

5.2 Scenario and Trace Data Description

For all the following scenarios, we adopted a network topology with 500 nodes and links of 1000 Mbit/s. We placed the origin server at a center node, according to min-K center placement strategy (Jamin et al. 2000). The topology has 10 autonomous systems and was generated with the IGen Tool depicted in Figure 5. We used a non-cooperative pull-based outsourcing strategy as described in (Sidiropoulos et al. 2008). The video objects have a constant bitrate of 300 Kbits/s and follow a Zipf-like size (or popularity) distribution. The monitor collects information every 1000 seconds. To avoid interference from old requests on new replacement decisions, as described in last section, we only use the requests from the last 1000 or 2000 seconds (two scenarios), hence shortening the system´s memory. Also, experiments were performed with 6 and 10 Caches whereas the comparison threshold was set to 0.9.

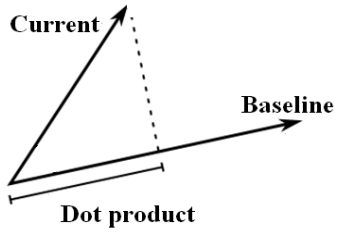

Figure 4: Dot product between baseline and current vectors

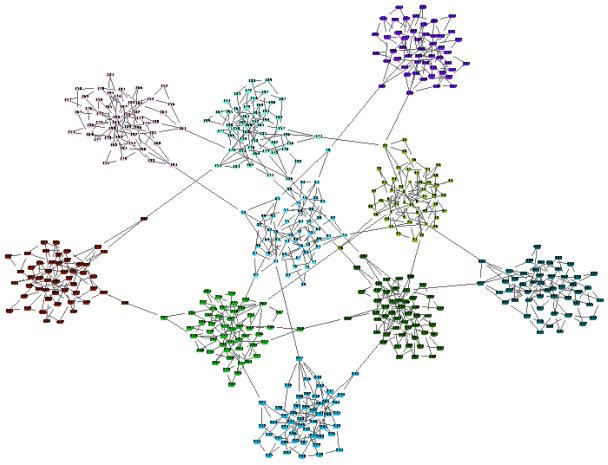

Figure 5: Topology used in simulations

For comparison purposes, we also implemented a virtualized model where adaptations are always performed at fixed time. This amount of time is a parameter called virtualization time. We call this scenario static virtualization as opposed to dynamic virtualization described above. Also, we have a scenario with no replacements taking place. In this scenario, the input for the placement algorithm (initial place-

ment) is the actual entire request file (i.e., it considers the requests that will take place during the simulations, just as if it had *Omniscience*), which means that the cache will always have the file requested. Thus, it works as a reasonable prediction of future (since all videos requested will already be available in the surrogate) and is therefore used as a baseline for comparing both static and dynamic virtualization, both of which perform placements based on recent history. It is not, however, a perfect prediction since the location of the surrogates will not be perfect throughout the simulation (it may not be considered a *perfect* scenario since the surrogates do not change location and therefore may not "move" towards the vicinity of the clients). Consider this scenario a case where replacements are performed only once (in the beginning) and the window is the whole simulation time.

Each scenario has one million requests distributed in 24 hours. Most requests arrive at a rate of 5 req/s from all nodes. However, in two time intervals, between 5 and 8 hours and between 12 and 15 hours, the rate suddenly increases to near 50 req/s and to almost 70 req/s respectively. Those are flash crowd events, modeled the same way as in (Chenyu et al. 2006). Differently from the rest of the simulation, the requests of flash crowd moments come from a specific AS. Thus, they concentrated in two of the 10 ASes, one at each time interval. This should force the system to adapt itself by moving surrogates closer to that AS at the flash crowd, and then, when it finishes, distributing them in the whole topology subsequently.

6 RESULTS

Figure 6 (b) shows the behavior of the mean startup delay during the simulation in a typical dynamic virtualization scenario. As it can be seen, during flash crowd events the startup delay understandably decreases. This happens because the majority of clients, which are located on the same AS (source of the flash crowd event) can reach the surrogate more rapidly as they are closer (due to surrogate relocation), at the expense of the ones outside that AS. Thus, the other clients, the ones located outside of the source, will have a higher startup delay. This is expected because during the flash crowd, most of the surrogates will be closer to the AS where it is originated, then, decreasing the mean startup delay. As the number of clients with less startup delay is much bigger, the mean is smaller for this metric. Note that the number of surrogates is not changed throughout the simulation time.

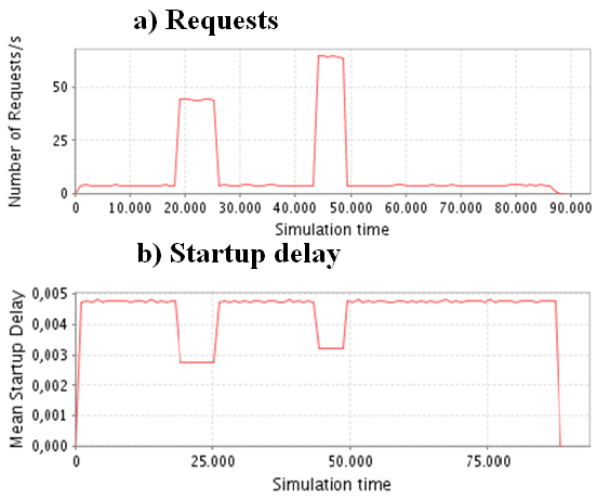

Figure 6: Average request count and startup delay

Table 1: Comparison among strategies

	None	**Static Virtualization**	**Dynamic Virtualization**
Monitor	N/A	1000 s	1000 s
Replacement	N/A	Each 2000 s	According to trigger function Considering the last 2000 s in requests for placement algorithm
# of replacements	1	44	15
Transferred bytes	8869304 MB	7201875 MB	6997881 MB

The results are very similar for network usage (the target). As, in these scenarios, there is no influence of external traffic in the network, the network usage is closely related to the observed startup delay. This only happens due to the low level of congestion in the network, and more requests means greater availability of content through peer-to-peer connections.

The replacement of surrogates in a CDN is a costly task. It involves reallocation of resources in the cloud. Thus, the virtualization time parameter must be a balance between a faster adaptation, which is related to more replacements, and the cost of the replacement, which is high. Concerning this issue, dynamic virtualization could achieve very good results. Table 1 shows the total amount of traffic in the simulation and the number of replacements. Dynamic virtualization performed 15 replacements while static virtualization performed as many as 44 on the same scenario. So, it has a better result with fewer replacements. In other words, it performs the adaptation at the right time, saving costs resulting from resource reallocation such as in a Cloud. Figure 7 shows all-network bytes transferred results for the scenarios with six caches and all types of virtualization. Figure 8 shows results with ten caches.

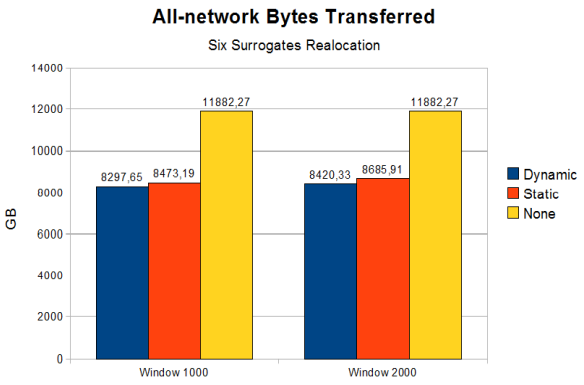

Figure 7: Comparison among virtualization strategies (6 caches)

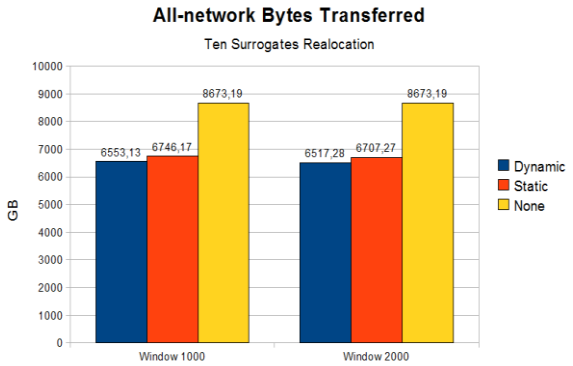

Figure 8: Comparison among virtualization strategies (10 caches)

According to Figure 7 and Figure 8, both dynamic and static virtualizations achieve very similar results (20% decrease) in network utilization. Since static virtualization induces considerably more reallocations, dynamic virtualization is by far the most recommended method.

For startup delay, results are shown in Figure 9 and Figure 10. As it can be seen, the results are closely related to network bytes transferred since there is no influence of external traffic. Also, it achieved a decrease of about 20% (best case).

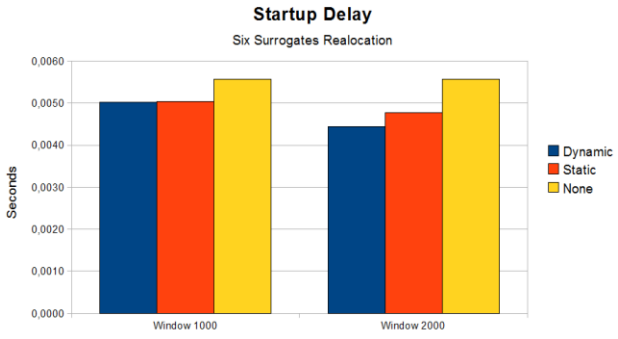

Figure 9: Startup delay results (6 caches)

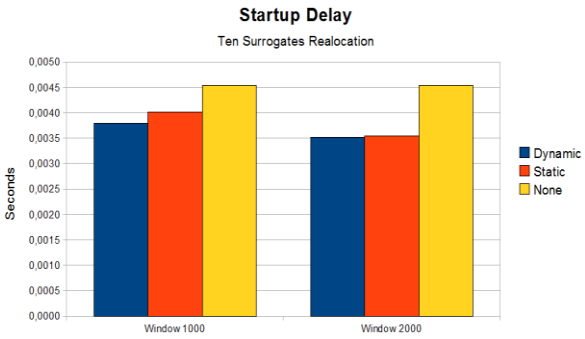

Figure 10: Startup delay results (10 caches)

7 CONCLUSIONS

This paper presented a proposal of virtualized CDN as a promising approach. It was designed to adapt the CDN infrastructure to the conditions of the network, mainly during some flash crowd events. The adapta-

tion is performed by the reallocation of resources in a Cloud computing environment. The adaptations can be either static, i.e. when adaptations occur at fixed time intervals, or dynamic, when performed according to a monitored trigger function. The necessary simulations were conducted with a CDN Simulator implemented in Java.

The utility function and triggering algorithm based on the dot product of the current and baseline usage data of the network have shown to be effective at adapting the topology of the virtualized CDN. Further studies including different methods will be performed. Moreover, the impact of creation of new surrogate servers need to be further evaluated.

The simulator offers an efficient and easy-to-extend environment to evaluate the performance of CDN strategies in realistic network conditions. It supports realistic measurements, simulating the main properties of the network and the complete behavior of the most relevant CDN components. It is an important tool for the research community. We intend to make it publicly available to the open-source community soon. In a near future, we plan to implement and evaluate more strategies based on existing literature and to perform more simulations with flash-crowd events as well in networks with high-levels of congestion.

REFERENCES

Anderson, T., L. Peterson, S. Shenker and J. Turner. 2005. "Overcoming the Internet impasse through virtualization," *Computer*.

Busari, M. and C. Williamson. 2002. "Prowgen: A synthetic workload generation tool for simulation evaluation of web proxy caches," *Computer Networks*, vol. 38, pp. 779-794.

Buyya, R., A.-M. K. Pathan, J. Broberg and Z. Tari. 2006. "A Case for Peering of Content Delivery Networks," *IEEE Distributed Systems Online*, 7 (10).

Carapinha, J. and J. Jiménez. 2009. "Network virtualization: a view from the bottom." In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*.

Chenyu, P., A. Merdan, B.H. Mohammad, S. Toshihiko and Y. Norihiko. 2006. "FCAN: Flash Crowds Alleviation Network Using Adaptive P2P Overlay of Cache Proxies," *IEICE Trans. Commun*.

Feamster, N., L. Gao and J. Rexford. 2007. "How to Lease the Internet in Your Spare Time," *ACM SIGCOMM Computer Communication Review*.

Freedman, M. J., E. Freudenthal and D. Mazières. 2004. "Democratizing Content Publication with Coral." In *Proceedings of 1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, San Francisco, CA, March 2004.

Gilbert, D. and T. Morgner. JFreeChart. Available at: http://www. jfree. org/jfreechart/index.html

Jamin, S., C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. 2000. "On the placement of Internet instrumentation." In *Proc. of IEEE INFOCOM*, Tel-Aviv, Israel, pp. 295–304.

Jamin, S., C. Jin, A.R. Kure, D. Raz and Y. Shavitt. 2001. "Constrained Mirror Placement on the Internet." In *Proceedings of IEEE INFOCOM*, Anchorage, Alaska, USA.

Khan, A. and R. Buyya. 2007. *A Taxonomy and Survey of Content Delivery Networks*, Technical Report, Australia.

Krishnan, P., D. Raz and Y. Shavitt. 2000. "The Cache Location Problem," *IEEE/ACM Transaction on Networking*, 8 (5).

Lechler, T. and B. Page. 1999. "DESMO-J: An Object Oriented Discrete Simulation Framework in Java". In *Proceedings of the 11th European Simulation Symposium*., pp. 46-50.

Li, B., M.J. Golin, G.F. Italiano, D. Xin and K. Sohraby. 1999. "On the Optimal Placement of Web Proxies in the Internet," In *Proceedings of IEEE INFOCOM*, NY, USA, pp. 1282-1290.

Lyer, S., A. Rowstron and P. Druschel. 2002. "Squirrel: A decentralized peer-to-peer web cache," In *Proc. 21th ACM Symposium on Principles of Distributed Computing*.

Padmanabhan, V.N. and K. Sripanidkulchai. 2002. "The case for cooperative networking." In *Proc. 1st International Workshop on Peer-to-Peer Systems*, pp.178–190, Cambridge, MA, USA.

Plagemann, T., V. Goebel, A. Mauthe, L. Mathy, T. Turletti and G. Urvoy-Keller. 2006. "From content distribution networks to content networks - issues and challenges," *Computer Communications*, 29(5).

Qiu, L., V.N. Padmanabhan and G.M. Voelker. 2001. "On the Placement of Web Server Replicas." In *Proceedings of IEEE INFOCOM*, Anchorage, Alaska, USA, pp. 1587-1596.

Sidiropoulos, A., G. Pallis, D. Katsaros, K. Stamos, A. Vakali and Y. Manolopoulos. 2008. "Prefetching in content distribution networks via Web communities identification and outsourcing." *World Wide Web Journal*, 11(1):39–70.

Stavrou, A., D. Rubenstein and S. Sahu. 2004. "A lightweight, robust P2P system to handle flash crowds," *IEEE J. Sel. Areas Commun.*, 22(1), pp.6–17.

Völker, L., D. Martin, I. El Khayaty, C. Werle and M. Zitterbart. 2009. "A Node Architecture for 1000 Future Networks," *Workshop of the IEEE International Conference on Communications*.

Wang, L., K.S. Park, R. Pang, V.S. Pai and L. Peterson. 2004. "Reliability and Security in CoDeeN Content Distribution Network." In *Proceedings of the USENIX 2004 Annual Technical Conference*, Boston, MA, USA.

Zhu, Y., R. Zhang-Shen, S. Rangarajan and J. Rexford. 2008. "Cabernet: connectivity architecture for better network services." In *Proceedings of the ACM CoNEXT Conference*.

AUTHOR BIOGRAPHIES

ANDRÉ L.C. MOREIRA is a PhD candidate in computer science in the Federal University of Pernambuco. His email address is andre@gprt.ufpe.br.

JOSILENE A. MOREIRA is an adjunct professor in the Federal University of Paraíba and a senior researcher in the Federal University of Pernambuco. Her email address is josilene@gprt.ufpe.br.

DJAMEL F. H. SADOK is an associate professor and the head of research in computer networks in the Federal University of Pernambuco. He is also a reviewer of several journals in computer networks. His email address is jamel@gprt.ufpe.br.

ARTHUR DE C. CALLADO is an adjunct professor in the Federal University of Ceará and a senior researcher in the Federal University of Pernambuco. His email address is arthur@ufc.br.

MOISÉS B.E. RODRIGUES is a MSc researcher in the Federal University of Pernambuco. His email address is moises@gprt.ufpe.br.

MÁRCIO M. Neves is a MSc student and researcher in the is Federal University of Pernambuco. His email address is mmn2@gprt.ufpe.br.

VICTOR SOUZA is a senior researcher in cloud computing in Ericsson Research, Sweden. His email address is victor.souza@ericsson.com.

PER P. KARLSSON is the head of research in cloud computing in Ericsson Research, Sweden. His email address is per.p.karlsson@ericsson.com.