

CDNsim: A Simulation Tool for Content Distribution Networks

STAMOS KONSTANTINOS

Department of Informatics, Aristotle University of Thessaloniki

GEORGE PALLIS

Department of Computer Science, University of Cyprus

ATHENA VAKALI

Department of Informatics, Aristotle University of Thessaloniki

DIMITRIOS KATSAROS

Department of Computer & Communication Engineering, University of Thessaly

ANTONIS SIDIROPOULOS

Department of Informatics, Aristotle University of Thessaloniki

and

YANNIS MANOLOPOULOS

Department of Informatics, Aristotle University of Thessaloniki

Content Distribution Networks (CDNs) have gained considerable attention in the past few years. As such, there is need for developing frameworks for carrying out CDN simulations. In this paper, we present a modeling and simulation framework for CDNs, called CDNsim. CDNsim has been designated to provide a realistic simulation for CDNs, simulating the surrogate servers, the TCP/IP protocol and the main CDN functions. The main advantages of this tool are its high performance, its extensibility and its user interface which is used to configure its parameters. CDNsim provides an automated environment for conducting experiments and extracting client, server and network statistics. The purpose of CDNsim is to be used as a testbed for CDN evaluation and experimentation. This is quite useful both for the research community (to experiment with new CDN data management techniques) and for CDN developers (to evaluate profits on prior certain CDN installations).

Categories and Subject Descriptors: I.6.5 [**Simulation and Modeling**]: Model Development; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms: Design, Experimentation, Measurement

Additional Key Words and Phrases: Content Distribution Network, trace-driven simulation, services, caching

Author's address: Department of Informatics, Aristotle University of Thessaloniki, 54124, Thessaloniki, Greece, e-mail: kstamos@csd.auth.gr

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2008 ACM 0098-3500/2008/1200-0001 \$5.00

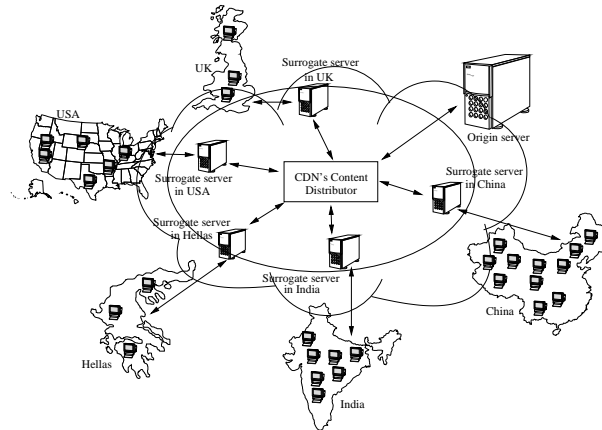


Fig. 1. A typical Content Distribution Network.

1. INTRODUCTION

Congested lines, obsolete backbones, multimedia content and increasing user population are all contributing to excessive Internet traffic. On a daily basis, users use the Internet for “resource-hungry” applications which involve content such as video, audio on-demand and distributed data. For instance, the Internet video site YouTube hits more than 100 million videos per day [YouTube]. Estimations of YouTube’s bandwidth go from 25TB/day to 200TB/day. At the same time, more and more applications (such as e-commerce, e-learning etc.) are relying on the Web but with high sensitivity to delays. A delay even a few milliseconds in a Web server content (e.g., the NASDAQ stock market) may be intolerable [Bent et al. 2004].

Content Distribution Networks (CDNs) [Vakali and Pallis 2003] have been proposed to meet such challenges by providing a secure, uniquely reliable, scalable and cost-effective mechanism for accelerating the delivery of the Web content. A CDN is an overlay network across Internet (an indicative CDN is depicted in Figure 1), which consists of a set of surrogate servers distributed around the world, routers and network elements. Surrogate servers are the key elements in a CDN, acting as proxy caches that serve directly cached content to clients. They store copies of identical content, such that clients requests are satisfied by the most appropriate site. Once a client requests for content on an origin server (managed by a CDN), his request is directed to the appropriate CDN surrogate server. Detailed information about CDN mechanisms are presented in [Rabinovich and Spatscheck 2002; Vakali and Pallis 2003].

CDNs play a key role in the Internet infrastructure since their high end-user performance and cost savings have urged many Web entrepreneurs to make contracts with CDNs [Market 2006]. Currently, CDNs invest on large-scale infrastructure (surrogate servers, network resources etc.), to provide high data quality and increased security for their clients. CDNs continuously become more competitive by offering novel services to the public. The development of a new service usually includes high investments. Therefore it is necessary to prototype, monitor and predict the behavior of a service in a controlled simulated environment, before and

after the release to the public.

A wide range of techniques [Chen et al. 2003; Kangasharju et al. 2002; Rabinovich and Spatscheck 2002; Venkataramani et al. 2002] has been developed, implemented and standardized for improving the performance of CDNs. However, most CDN providers do not take advantage of these techniques. This situation arises because the proposed ones have not been extensively evaluated by a detailed simulation testbed. Thus, the CDN administrators do not have a clear view about the costs/gains of these techniques in order to be enhanced by a CDN provider. The lack of efficient CDN simulation tools has been highlighted in several works [Bent et al. 2004; Clark et al. 2007; Chen et al. 2003; Wang et al. 2002]. Furthermore, academic CDNs [Pierre and Steen. 2006; CoDeeN ; CORAL], based on real testbeds like PlanetLab [Planetlab], are treated mostly as black boxes or require the volunteer involvement of many individuals. Therefore, the development of novel techniques in such environments is quite difficult or impossible.

Taking into account the high interest of CDNs [Pallis and Vakali 2006], it is crucial to develop a realistic simulation environment for CDNs. Specifically, the goal of this work is to present a reliable and memory-efficient tool which can simulate in great detail large-scale CDNs. Such a tool is essential for software researchers and practitioners, since it would become a useful testbed for evaluating and validating the performance of CDNs. The main paper's contributions are summarized in:

- developing an analytic simulation tool for CDNs, called CDNsim, taking into account the characteristics of Internet infrastructure. CDNsim has been designed to support research in broad-coverage CDN services. It is a parallel discrete event trace driven network simulation package that provides utilities and interfaces for content delivery on the Web. It has also the ability to simulate peer to peer (p2p) services as well as various internetwork configurations. CDNsim is scalable and robust in order to perform a wide range of CDN policies.
- providing a graphic user interface (window-based environment) for setting all the parameters of the simulation and automating the simulation executions.

To the best of our knowledge there is no other complete suite simulating a CDN. The challenge of this tool is to become an essential evaluation tool both for CDN scientific community providing a simulation testbed for current research and development activities in this area. In particular, CDNsim enables users - primarily researchers and software practitioners - to evaluate and validate new policies and services under a realistic CDN infrastructure.

The remainder of this paper is organized as follows: Section 2 reviews the related work. Section 3 presents the main features of CDNsim. Section 4 describes the architecture of the proposed CDN simulator. Section 5 presents some experimentation results of CDNsim. Section 6 presents the user interface of CDNsim. Section 7 presents two use cases of CDNsim. Section 8 discusses the value of CDNsim in practice and Section 9 concludes the paper.

2. RELATED WORK

CDNs have gained considerable attention in the past few years. The earlier recent research work on CDNs can be divided into the following four major categories:

- Establishing theoretical models:** Theoretical models can be used to efficiently solve the resource allocation and management problems in a CDN [Bektas and Ouveysi 2008]. In particular, mathematical models have been proposed in the literature to address several issues related to where to locate surrogate servers [Qiu et al. 2001], which content to outsource [Kangasharju et al. 2002], evaluating pricing models [Hosanagar et al. 2006] and request routing mechanisms [Oliveira and Pardalos 2005; Bektas et al. 2008]. Mathematical modeling techniques can also be used to gain insight to a variety of CDN problems arising in practice and to determine what mitigating actions can be taken. For instance, the authors of [Nguyen et al. 2005] use a Lagrangean-based solution algorithm based on a mathematical model to evaluate the effect of data clustering on the total revenue of a CDN provider using this algorithm. Moreover, theoretical models facilitate the solution of CDN problems by providing a generic framework on which efficient exact solution algorithms can be devised. These are also used as benchmarks to assess a variety of heuristic methods [Laoutaris et al. 2005]. However, all these models deal with the individual problems separately, without taking into account possible interplays between them. Therefore, while they provide valuable information, the need for simulations is not tackled where all those problems can be aggregated.
- Developing policies for CDN infrastructure:** Several issues are involved in CDNs since there are different decisions related to CDN framework setup, content distribution and management, and request management approaches. This category deals with identifying new policies for the above issues. We do not provide any details for such methods since it is beyond the scope of this paper. For the readers who are interested in this subject, a concrete paper is presented in [Pallis and Vakali 2006].
- Developing academic CDNs:** Instead of delegating the content delivery to a commercial CDN provider, the Web content servers participate in an academic CDN with low fees. Academic CDNs are real world systems and run in a wide area environment, the actual Internet topology. A well-known academic CDN, Globule [Pierre and Steen. 2006], is an open source CDN which is operated by end-users. The Web content servers participate in the Globule by adding a module to their Apache server. Another academic CDN is the CoralCDN [CORAL]. In order to use the CoralCDN, the Web content servers, which participate in this network, append .nyud.net:8080 to the hostname in a URL. Through DNS redirection, the clients with unmodified Web browsers are transparently redirected to nearby CORAL surrogate servers. Another well-known academic CDN is the CoDeeN [CoDeeN]. In order to use the CoDeeN, as previously, a prefix must be added to the hostname in a URL. Regarding the academic performance of CDNs, they offer less aggregate storage capacity than commercial CDNs and, require wide adoption of the system to bring substantial performance benefits to the end-users. However, the existing academic CDNs cannot be used as testbed platforms in order to evaluate the efficiency of novel CDN policies.
- Developing simulation testbed systems:** This category deals with developing a CDN simulation system, which will simulate a dedicated set of machines to reliably and efficiently distribute content to clients on behalf of the origin server.

Such a testbed runs locally on a single machine and contrary to the academic CDNs it is a simulated environment. In the following paragraphs, we present the existing CDN simulation systems.

The CDN providers are real time applications and they are not used for research purposes. Therefore, CDN simulators are valuable tools for researchers as well as for practitioners in order to develop and evaluate CDN policies. In addition, they are economical because they can carry out experiments without the actual hardware. They are also flexible because they can, for example, simulate a link with any bandwidth and propagation delay and a router with any queue size and queue management policy. Finally, the simulation results are reproducible and easy to analyze because the simulated network environment is free of other uncontrollable factors (e.g., other unwanted external traffic), which researchers may encounter when doing experiments on real networks. These factors may also be simulated in order to maximize the realism of the simulated model.

Most existing CDN simulation systems [Bent et al. 2004; Chen et al. 2003; Kangasharju et al. 2002; Wang et al. 2002] do not take into account several critical factors, such as the bottlenecks that are likely to occur in the network, the number of sessions that can serve each network element (e.g. router, surrogate server) etc. Thus, results may be misleading since they measure the number of traversed nodes (hops) without considering the TCP/IP network infrastructure. On the other hand, there is a wide range of network simulators [Fall]; however, they cannot effectively simulate the Internet infrastructure. For instance, the ns-2 simulator [NS] is a discrete event simulation framework commonly used to simulate the TCP/IP protocol, flow control and congestion control mechanisms. However, it requires a huge amount of memory to simulate large-scale internetwork infrastructures.

The insufficiency of existing network generators for simulating a CDN has also been indicated by [Wang et al. 2002] where the authors developed a CDN simulation environment integrating two existing simulations: the ns-2 network simulator (to simulate the Internet infrastructure) and the logsim simulator (to simulate the surrogate servers disks). However, as the authors report, this simulation model is not an efficient one for large-scale networks since it requires 2-6 GB of RAM, and generally takes 20-50 hours of wall-clock time. In order to restrict these high-memory requirements, the authors of [Kulkarni et al. 2003] proposed to simulate the cache disks by using memory-efficient data structures, called Bloom Filters. Results have shown that a prudent use of Bloom filters may achieve a considerable reduction in memory requirements of CDN simulations. However, the use of Bloom filters limits the ability to efficiently manage the storage space of surrogate servers by using cache replacement policies. Also, all the previously mentioned testbeds are not freely available to research community for conducting simulations.

Some researchers experiment their CDN policies on testbed platforms [Wang et al. 2004]. Such a platform is the PlanetLab [Planetlab]. Specifically, PlanetLab is a network of computers located at Universities and other Institutions around the world, forming a testbed for creating and deploying planetary-scale services - massive applications that span a significant part of the globe. In this context, both CoDeeN [CoDeeN] and CoralCDN [CORAL] are academic testbed CDNs built on top of PlanetLab. This testbed CDN consists of a network of high-performance

Testbed	TCP/IP	Memory usage	Experiments Reproducibility	Scalability	Availability	Execution Environment
CDN simulator (ns2 & logsim)	Yes	High	Yes	Medium	No	simulated - local
CDN simulator [Bent et al. 2004; Chen et al. 2003]	No (hop-based)	Medium	Yes	High	No	simulated - local
CDN simulator [Kulkarni et al. 2003]	No (hop-based)	Low (bloom filters)	Yes	High	No	simulated - local
CoDeeN	Yes (Planet-Lab)	Low	No	Medium	Restricted	real - wide
CoralCDN	Yes (Planet-Lab)	Low	No	Medium	Restricted	real - wide
Globule	Yes	Low	No	Low	Free - Open Source	real - wide
CDNsim	Yes	Low	Yes	High	Free - Open source	simulated - local

Table I. Testbed platforms for CDNs.

proxy servers. The proxy servers have been deployed on many PlanetLab nodes, which behave as surrogate servers. The major limitation of such a platform is that it can be used by the users where their institutions are member of the PlanetLab consortium. Another limitation is the fact that the experiments are not reproducible on PlanetLab platform [Oppenheimer et al. 2004; Spring et al. 2005], since it does not provide a controlled environment.

From the above discussion it is evident that there are no free CDN simulation suites available to research community. This is our primary motivation for designing CDNsim. A comparative table of the existing testbed platforms for CDNs is presented in Table I.

3. CDNSIM FEATURES

From the above discussion, it is obvious that there is a lack of a reliable and scalable CDN simulator, since both CDN simulators and testbed platforms have their own limitations. CDN simulators can only simulate real-world implementations with limited detail (e.g., using a static estimate for the network transfer time). The need for developing such a software tool has also been indicated in [Bent et al. 2004]. In general, the development of a complete CDN simulator, including associated application programs and network tools, is a time consuming task since typical network or cache simulators cannot be used in order to simulate a CDN.

The CDNsim has been developed to overcome the above problems. CDNsim is a public-source, modular and open-architecture parallel discrete event trace driven CDN simulation system which is based on OMNeT++ [Varga a] simulation environment and the INET framework. INET is an extension of OMNeT++ to provide network protocols like TCP/IP. The source code of CDNsim and its documentation are available from <http://oswinds.csd.auth.gr/~cdnsim>. In the appendix A, the fundamental concepts for the OMNeT++ are presented. CDNsim uses OMNeT++ only for the basic networking operations such as TCP/IP transmissions and for discrete event scheduling. The request routing, content distribution and management as well as all the CDN characteristics are simulated by CDNsim itself. In the following paragraphs, the main features of CDNsim are discussed (Table II).

CDN Framework Setup	
CDN Organization	Surrogate servers & network components
Servers	Origin servers & Surrogate servers
Relationships	Client \rightarrow Surrogate servers (inter-proxy communication) \rightarrow Origin server
Interaction Mechanisms	Network elements interaction; Inter-cache interaction
Content Service/Types	Static content; Streaming media; Services
Content Distribution & Management	
Surrogate Servers Placement	Any policy
Content Selection and Delivery	Any policy
Content Outsourcing	Cooperative push-based; uncooperative push-based; cooperative pull-based; uncooperative pull-based
Cache Organization	On demand; Periodic update
Request Management	
Request Routing Mechanisms	DNS-based request routing

Table II. Main features of CDNsim.

3.1 CDN Framework Setup

The main features of CDNsim framework setup can be categorized as follows:

- CDN Organization:** In CDNsim both the surrogate servers (which are placed at several places in the network) and the network components handle the distribution of specific content types (e.g. Web content, streaming media, and real time video). A similar approach is also followed by most of the commercial CDN providers such as AKAMAI and Limelight Networks for CDN organization.
- Servers:** In a CDN infrastructure, there are two types of servers: origin and surrogate servers. The origin server (also known as Web server content) stores the origin version of resources. A surrogate server holds a replica of a resource and acts as an authoritative reference for clients responses. The origin server communicates with the distributed surrogate servers to update the content stored in it. CDNsim may support in its infrastructure a large-scale number of both surrogate and origin servers.
- Relationships:** The complex distributed architecture of a CDN exhibits different relationships between its constituent components (clients, surrogate servers, origin servers, and other network elements). In CDNsim, a client communicates through the network's elements with surrogate and origin servers. The communication between a client and a surrogate server takes place in a transparent way, where each surrogate server serves clients requests from its local cache or acts as a gateway to another surrogate server or origin server. On the other hand, the

surrogate servers can be simultaneously accessed and shared by many clients.

- Interaction Mechanisms:** The interaction mechanisms are used for interaction among CDN components. Such interactions can be broadly classified into two types: interaction among network elements (e.g., routers) and interaction among surrogate servers. Regarding the networks element interaction, CDNSim implements an approach for signaling between servers and the network elements that forward traffic to them. This mechanism allows network elements to perform load balancing across a set of distributed servers and redirection to other servers. From a technical point of view, it uses TCP as the transport protocol, where each server establishes a TCP connection to the network elements using a well-known port number. Messages can then be sent bi-directionally between the server and network element. All the messages consist of a fixed-length header containing the total data length and a request followed by a reply or an acknowledgment. Regarding the interaction among surrogate servers, CDNSim implements a powerful mechanism to eliminate redundancy and make better use of Internet server and bandwidth resources. It supports peering between surrogate servers without taking place a request-response exchange. Using this mechanism, we accurately determine whether a particular surrogate server caches a given object. From a technical point of view, it is currently performed via HTTP or FTP.
- Content Service / Types:** CDNSim supports a wide variety of Web content including static content (HTML pages, images, documents, software patches, audio and/or video files), streaming media (live or on-demand) and services (e.g., e-commerce services).

3.2 Content Distribution and Management

Content distribution and management issues play a significant role for the CDN performance.

- Surrogate servers placement:** Determining the network locations for surrogate servers in a network topology (known as the Web server replica placement problem) is critical for content outsourcing performance and the overall content distribution process. CDN topology should be built such that the client-perceived performance is maximized and the infrastructure cost is minimized. Therefore, effective surrogate server placement reduces the number of surrogate servers needed and the size of content (replicated on them), in an effort to combine the high quality of services and low CDN prices. CDNSim may support a wide range of placement algorithms (Greedy, which incrementally places replicas, Hot Spot, which places replicas near the clients generating the greatest load, and Tree-based replicas) [Li et al. 1998; Qiu et al. 2001].
- Content selection and delivery:** The choice of the content that should be outsourced in order to meet clients needs is known as content selection problem. Considering the huge amount of Web data, the challenge of the content selection problem is to find a sophisticated management strategy for replication of Web content. CDNSim may support several Web data management policies [Katsaros et al. 2008; Sidiropoulos et al. 2008].
- Content outsourcing:** Under a CDN infrastructure with a given set of surrogate servers and a chosen content for delivery, it is crucial to decide which

content outsourcing practice to follow [Chen et al. 2003; Kangasharju et al. 2002; Pallis et al. 2005]. CDNsim supports four content outsourcing policies: cooperative push-based, uncooperative push-based, cooperative pull-based, and uncooperative pull-based. In cooperative push-based policy, the content is pushed (proactively) from the origin Web server to CDN surrogate servers. Initially, the content is prefetched (loaded in cache before it is accessed) to the surrogate servers and then, the surrogate servers cooperate in order to reduce the replication and update cost. In this scheme, CDNsim maintains a mapping between content and surrogate servers, and each request is directed to the closest surrogate server (that has the requested object), or otherwise, the request is directed to the origin server. In the uncooperative push-based scheme, the content is pushed (proactively) from the origin Web server to the surrogate servers. The requests can be satisfied either at a local surrogate server or at the origin Web server, but not at a nearby surrogate server due to the lack of informed request redirection. In cooperative pull-based approach, the clients requests are directed through DNS redirection to their closest surrogate server. The key in the cooperative pull-based approach is that the surrogate servers are cooperating with each other in case of cache misses. Finally, in uncooperative pull-based policy, the clients' requests are directed to their closest surrogate server. If there is a cache miss and the requested content is not found, the request is directed either to a peering surrogate server of the CDNsim or to the origin server. More specifically, the surrogate servers, which serve as caches, pull content from the origin server when a cache miss occurs.

- Cache organization:** In order to ensure content consistency and freshness, CDNsim is ready to support either on-demand or periodic update. In the on-demand update, the latest copy of a document is propagated to the surrogate server based on prior request for that content. In the periodic update, the CDNsim configures its origin Web servers content to provide instructions to caches about what content is cacheable, how long different content is to be considered fresh, and when to check back with the origin server for updated content. With this approach, caches are updated in a regular fashion. However, any developer could also build its own policy or use some heuristics to deploy organization specific caching policies [Laoutaris et al. 2005; Stamos et al. 2006].

3.3 Request Management

In a CDN there is a mechanism which redirects the clients' requests to the most appropriate surrogate server. This mechanism is responsible for routing the clients' requests to a specific surrogate server for the delivery of content. It has global awareness of the network topology and the surrogate servers content.

- Request routing mechanisms:** Request routing mechanisms inform the client about the selection of surrogate server, generated by the request-routing algorithms. CDNsim supports DNS-based request-routing mechanism. In this approach, the content distribution services rely on the modified DNS servers to perform the mapping between a surrogate server symbolic name and its numerical IP address. In DNS-based request-routing, a domain name has multiple IP addresses associated to it. When a client's request comes, the DNS server of the

CDNsim returns the IP addresses of servers holding the replica of the requested object. The client's DNS resolver probes to the surrogate servers and chooses the surrogate server with respect to the response times to these probes. The performance and effectiveness of DNS-based request-routing has been examined in a number of recent studies [Alzoubi et al. 2007]. The advantage of this approach is the transparency as the services are referred to by means of their DNS names, and not their IP addresses.

3.4 Security Vulnerability Issues

CDN systems may face attacks, such as botnets and puppetnets [Lam et al. 2006], that create flash crowd events [Ramamurthy et al. 2007], which could lead to Denial of Service [AkamaiReport 2008]. Specifically, flash crowds are sudden, unanticipated surges in traffic volume of request rates towards particular Web servers content. Such attacks occur quite often and present significant problems to Web server content owners. For instance, in commercial Web servers content, a flash crowd can lead to severe financial losses, as clients often resign from purchasing the goods and search for another, more accessible Web server content. We refrain from modelling each and every type of security vulnerability, since they all create surges of requests. Instead, we provide flexibility in modelling flash crowds. Thus, it is important to evaluate such attacks. More details about how flash crowds incorporated in CDNsim are described in section 7. Other attacks (e.g., routing attacks), which usually face most networking systems, are not common in CDN systems. CDNs provide monitoring services [Zhang et al. 2007] to detect such attacks.

4. CDNSIM ARCHITECTURE

In this section the architecture of CDNsim is presented. We define an abstract service oriented architecture and present how this architecture is used to implement the CDNsim features which have been discussed in the previous section.

4.1 Abstract Design

In a CDN topology we identify the following network nodes which are interconnected via network links: surrogate servers, origin servers, clients, routers and DNS redirection servers. Each network node provides a set of services which are available to other ones. Specifically, we identify the following two types of services:

- The client-server service.** It covers the case of internode communication / interaction. For instance, the client interacts with the DNS server to retrieve the IP address of the surrogate server to which it should send a request.
- The daemon service.** It runs on the system locally and does not fit the client-server approach. For instance, a surrogate server needs a service which periodically frees up cache space by removing unwanted objects from the local cache.

The nodes of the network topology have a common behavior; they all run a set of services. Therefore, they can be modeled by an abstract node of which all the network nodes are considered as subclasses. The abstract node provides a base for services hosting. The individual nodes are differentiated by implementing different services. Each service either exchanges some kind of information (i.e.

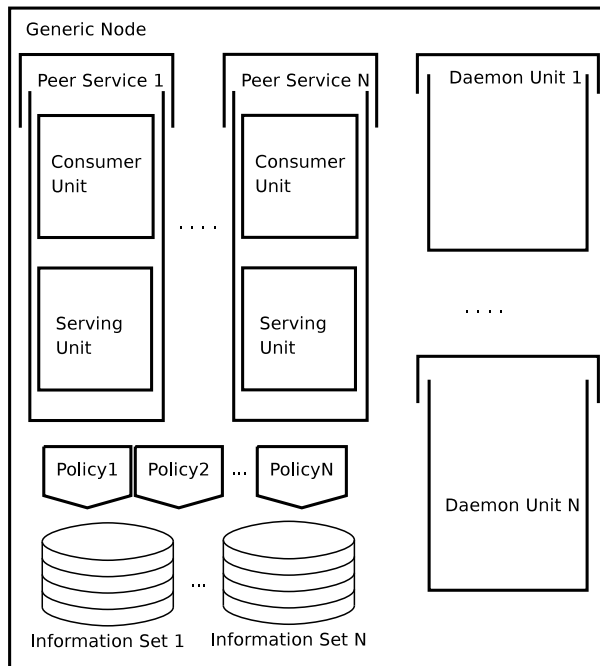


Fig. 2. Generic Node and internals.

videos) between nodes or affects a local repository of information (i.e. local cache). All the communications are performed by exchanging messages. Therefore, these observations lead us to formally define the following architectural components.

- **Information Unit.** It models in an abstract way any kind of information fragment that can be stored and transmitted. Using agglomerations of Information Units in an hierarchical form CDNsim may represent any type of content (e.g., video files, audio, packets of media streams, text, Web pages). In a client-server type service, Information Units are served to the client by the server.
- **Information Set.** This component acts as a storage manager of Information Units. A set of policies are offered as an interface to manipulate them.
- **Message.** Messages can be considered as envelops which contain information needed to be exchanged between components. The carried information may be Information Units or any other instructions (such as reply ports) that can be interpreted by the receiver of a Message.
- **Peer Service.** Peer Service is a component used to represent both service types (client-server and daemon). Regarding the client-server type we may identify three distinct expressions: a) client, b) server and c) mixed. The first case is when only the client side of a service is implemented (i.e. ftp client). The second case covers the server side implementation of a service (i.e ftp server). The last one covers the case where both (a, b) expressions coexist; imagine a p2p network (like eMule) where users both may download and serve files. This case exists in a CDN infrastructure where surrogate servers serve content but also download content

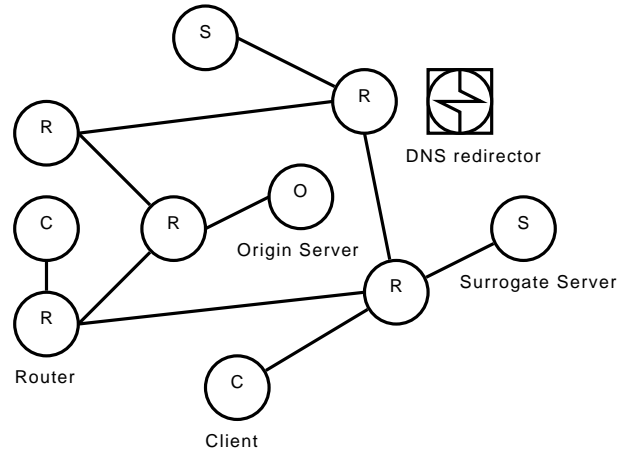


Fig. 3. The network topology.

from other servers. The *client* and *server* are implemented by two components which are called Consumer and Serving Unit respectively. Both Consumer and Serving Unit are wrapped by Peer Service. As far as the daemon service type is concerned, it is implemented by the Daemon Unit. Using Daemon Units, we may cause periodic events such as requests generation, bookkeeping procedures, cache cleaning and so on. Information Sets, optionally, can be shared (taking into account possible deadlocks) among many Peer Services.

- Generic Node.** This component can be extended to represent any network node such as surrogate servers, origin servers and clients. Figure 2 depicts its internal structure and the components hierarchy. It provides the appropriate environment for hosting services. It wraps all the Peer Services and Information Sets and forwards incoming Messages to the appropriate Peer Service according to the requested service type. Moreover, it provides the necessary network protocol interfaces for packets transmission, since the Generic Nodes are directly connected to the physical network through links. This design approach enables us to implement an extensible service-oriented CDN, where we can plug any number of new services inside a surrogate server.

4.2 *CDNsim* Implementation

CDNsim is implemented by extending the above architectural components. In this subsection we present the network topology modeling and the available CDN services.

4.2.1 *CDNsim* Network Topology Modeling. The network topology of *CDNsim* consists of a set of nodes (Generic Nodes) which are interconnected via network links as shown in Figure 3. Each node contains a compilation of services (details about these services are presented in the next subsection) where the internal structure of each node is described as follows:

- Client.** The client, depicted in Figure 4, is the initiator of requests to CDN.

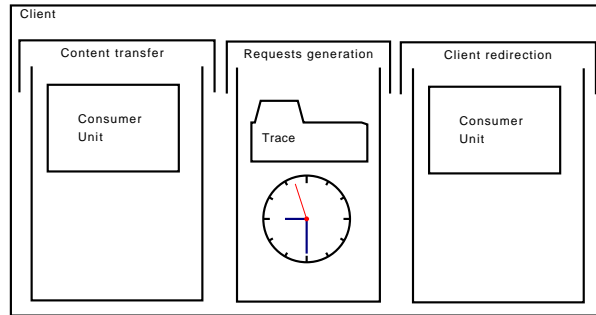


Fig. 4. The Client Generic Node.

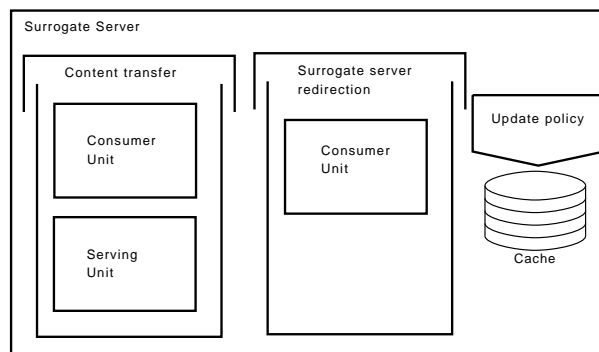


Fig. 5. The Surrogate Server Generic Node.

It contains the *requests generation service* (marked in the Figure with a wall clock) which preloads the clients' requests. The requests are performed and the serviced content is retrieved using the Consumer Unit of the *content transfer service*. Clients are assigned to surrogate servers in order to submit their requests using the *client redirection service*.

- **Surrogate server.** Figure 5 illustrates the surrogate server which contains the mixed *content transfer service* since it acts both as server and client. Additionally, it includes the *surrogate server redirection service* for detecting alternative servers to pull content and a local cache of finite capacity.
- **Origin server.** The origin server, shown in Figure 6, wraps the Serving Unit of the *content transfer service* and a cache which contains all the available content.
- **DNS redirector.** The DNS redirector, which is depicted in Figure 7, includes the Serving Units of all the *redirection services*.

The intermediate router nodes are provided by INET (they are excluded from the Generic Node architecture). When routers are configured, they are used as “black boxes”, which retransmit network packets according to the current network protocol. INET library includes options about retry timeouts, retry counts, delays concerning the Address Resolution Protocol (ARP), network datagram sizes etc. It should be noted that network nodes are not aware of the content of the packets. We

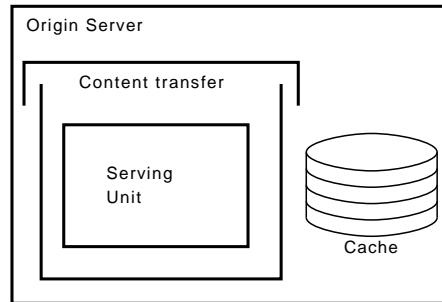


Fig. 6. The Origin Server Generic Node.

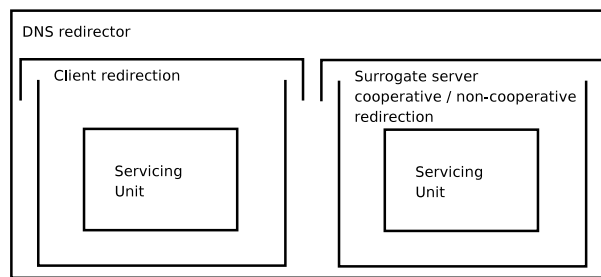


Fig. 7. The DNS Redirector Generic Node.

make use of the NED language which is suitable for modeling nodes hierarchy and connections building essentially a network topology. Thus, the surrogate servers and clients placement to the network is a matter of providing the desired configuration using NED. Detailed information about the NED language can be found in the OMNeT++ manual [Varga b]. The speed and propagation delay of each link can be modified (the user may use either real or artificial measurements [Sripanidkulchai et al. 2004]) and it is simulated by the INET framework. TCP / IP is the main protocol used for performing communications between services. TCP / IP and all lower level protocols (i.e. network layer) are provided by the INET framework. An in-depth presentation of TCP/IP in INET along with tests and benchmarks can be found in [Idserda 2004]. Although there are default options for the TCP/IP options, the advanced user may choose the appropriate TCP algorithm (TCPTahoe/TCPreno/TCPNoCongestionControl/DumbTCP), advertised window, maximum segment size, TTL and so on.

4.2.2 *CDNsim Services.* The CDNsim implementation supports the following CDN services:

- **Client redirection service.** This service manages the client redirection process. The clients are redirected to the closest surrogate server in terms of network distance. This distance metric is based on the Dijkstra algorithm [Dijkstra 1959], however it can be extended to include more sophisticated methodologies (such as MyXDNS [Alzoubi et al. 2007]) that include information about the load of each server. The client side of this service runs at the clients while the server side runs

at DNS redirector. Upon a request the client is redirected by the DNS redirector to the appropriate surrogate server by advertising the IP address and listening port.

- **Surrogate server (cooperative/non-cooperative) redirection service.** This service takes place upon a cache miss. Specifically, the surrogate servers use one of the two instances (cooperative/non-cooperative) of this service. If the cooperative service is activated, the surrogate servers are redirected through DNS redirector to the closest surrogate server that contains the requested object. On the other hand (the non-cooperative service is activated), the surrogate servers are directed through DNS redirector to the origin server.
- **Requests generation service.** This daemon service is running at the clients. At the beginning of the simulation it loads the corresponding trace file containing the requests to be made. The requests are sorted by timestamp and each one is scheduled appropriately. The scheduled requests are performed by the content transfer service of the client. CDNsim does not generate traffic according to some specific methodology or algorithm. The traffic patterns are defined in the trace file which can be real (i.e. Apache logs) or artificially generated by a third party tool such as Medisyn [Tang et al. 2003]. CDNsim executes the user specified traffic.
- **Content transfer service.** This service manages the requests for content (i.e. video, audio, text, HTML pages, etc). The clients perform requests to the CDN while surrogate servers attempt to satisfy them. This service implements a set of TCP applications responsible to upload and download files. The number of these applications set connections capacity of the service and thus the “strength” of a surrogate server. A generic interface is offered that can be tweaked in order to support services such as VOIP and streaming media. The CDNsim user could use TCP dumps that contain all the TCP traffic of streaming services between peers, in a form of a trace file. A TCP dump must be preprocessed in order to fit the CDNsim assumptions and Message transmission logic. Such TCP dumps are available at <http://content.lip6.fr/traces/trace/viewer/1>.

5. RESOURCES SCALING UNDER *CDNSIM*

This section presents a set of experiments conducted by CDNsim in order to address software bugs and performance issues. The primary goals of CDNsim implementation are to provide a bug free environment with reasonable memory footprint and execution time.

5.1 Software Bugs

Security vulnerabilities in a simulation system may result from software bugs. Thus, it is important to ensure that CDNsim is free of software bugs. Taking into account that CDNsim is a desktop application and does not make use of any real networking, we are focused on errors related to memory leaks and invalid memory accesses. To address these issues we evaluated CDNsim using Valgrind [Armour-Brown et al. ; Nethercote and Seward 2007], an open-source memory debugger. Valgrind is a binary-code dynamic checker which detects general memory-related bugs such as memory leaks, memory corruption and buffer overflow. Simulating

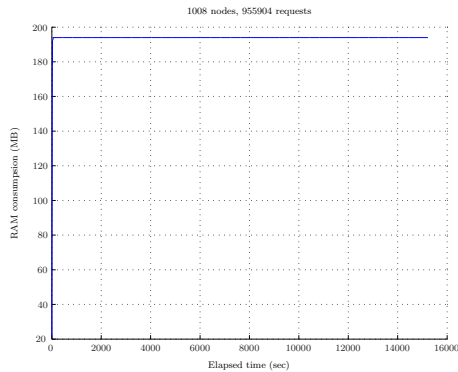


Fig. 8. Memory usage during simulation.

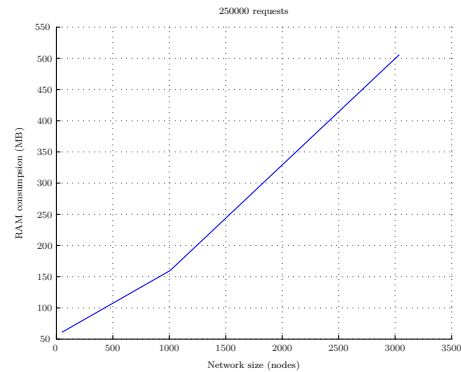


Fig. 9. Memory usage vs network size.

every single instruction of a program, it finds errors not only in a program but also in all supporting dynamically-linked libraries. All detected errors are reported. In CDNSim, the Valgrind report does not report any error.

Figure 8 demonstrates the memory variations during the execution of the simulation. The x-axis represents the elapsed CPU time measured in seconds whereas the y-axis represents the RAM consumption measured in MB. The network contains 1008 routers, 100 surrogate servers, and 955904 requests have been served. An expected increment of memory is observed at the beginning of the simulation while the simulation environment is being built and the network fills with network packets. After this short time period, the simulation memory footprint is stable to the end of the simulation. This is an indication that there are no memory leaks; otherwise we would observe an increasing memory curve as more and more memory chunks would remain in the system unfreed.

5.2 Performance Issues

Here, we investigate how CDNSim performance is affected by varying:

- the network size: CDNSim can be used to model large-scale network topologies. Figure 9 depicts how memory scales while increasing the network size. In our experiments, we used a Web site of 3000 objects and a request stream of 50000 requests. The x-axis represents the network size in nodes whereas the y-axis represents the RAM consumption measured in MB. Using 50, 1008 and 3037 routers we observe an almost linear increment in memory consumption as the number of nodes increases. We should note that for 3037 nodes we need about 500 MB of RAM, while using ns2 we built the same 3037 nodes TCP/IP network and we needed up to 2GB, which means 4 times more memory consumption than CDNSim.
- the network topology: Using the GT-ITM [Zegura et al. 1996] internetwork topology generator we generated various network topologies (pure random, Transit-stub and AS). The Transit-stub generates internetwork topologies composed of interconnected transit-stub domains. An AS-level Internet topology uses BGP routing data collected from a set of geographically-dispersed BGP peers. The

generated topologies are the backbone of the network topology where the surrogate servers, intermediate routers, clients and origin server are attached. The intermediate routers are simulated by using a module of the INET library that implements the necessary interfaces for Ethernet, MAC and the various OSI layers. This module includes options about retry timeouts, retry counts, delays concerning the Address Resolution Protocol (ARP), network datagram sizes etc. More details can be found in the OMNeT++ manual [Varga b]. A summary of the network parameters is recorded in Table III. In our experiments we used 1000000 requests in order to observe the long-term behavior of CDNsim. As expected, we observe an increasing memory consumption as the number of nodes (routers) increases. Regarding the CPU time, we observe only small fluctuations to the recorded values. This happens because the Internet topology exhibits small world properties introduced in [Watts and Strogatz 1998]. Specifically, studies have shown that the average number of hops between the nodes in the AS-level Internet topology is small and does not change as the network grows in size [Dhamdhere and Dovrolis 2008]. Another observation is that the dominant factor about the memory consumption between the different types of topology is the number of edges (network links). At this set it occurs that the pure random topology had the most edges following Transit-stub and AS.

- the number of users requests: The simulation of a CDN is highly compute-intensive. The compute-intensive nature arises from the need to simulate a large number of end-user requests and various inter-proxy and proxy-server interactions. In general, the larger the number of end-user requests and the scale of CDN, the more are the computational requirements. Figure 10 depicts the performance of CDNsim regarding CPU time in terms of increasing the number of requests. The x-axis represents the number of requests, whereas, the y-axis represents the CPU time measured in seconds. CDNsim is able to execute up to 1 million requests in a network of 1000 nodes in about 4 hours using a Pentium IV 3.2 GHz. This CPU time is satisfactory given the fact that the authors of [Wang et al. 2002] needed up to 50 hours for such a simulation. Moreover, by increasing the number of requests the execution time increases as well in an absolutely linear way indicating that CDNsim scales efficiently as the number of requests increases.
- the users demand models: To produce different users demand models we used a client request stream generator which captures the main characteristics of Web users behavior [Katsaros et al. 2008]. This generator produces synthetic workload using mathematical models and stochastic processes. Here, we produced different user demand models by varying the following parameters: Zipf slope of the popularity distribution, the Pareto heavy tail index of the sizes distribution, and the percentage of unique objects inside the request stream. These parameters have usually a significant impact in the workload generation. For the experiments, we used a topology of 1008 routers, 100 surrogate servers and the LRU (Least Recently Used) cache replacement policy in CDNsim surrogate servers. The summary of results is recorded in Table IV. For the percentage of distinct documents of total number of requests, we used the values 10%, 30% and 80%. As the percentage increases, we observe an increment in the CPU time. This is

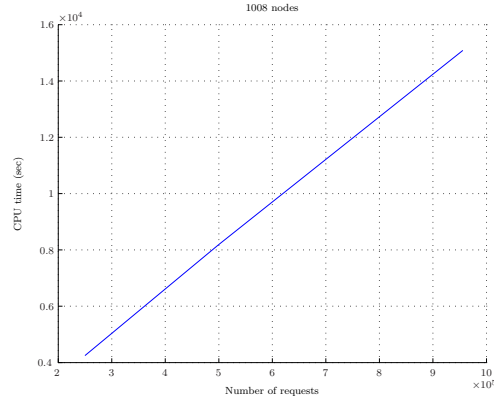


Fig. 10. CPU time vs number of requests.

expected since LRU policy suffers from high miss ratio. The cache misses lead to intersurrogate server traffic and, consequently, high CPU time. Moreover the memory increases as the number of objects increases. For the Zipf slope, we used a variety of values from 0.01 up to 1. In general, a steep slope favors the LRU; the best CPU time is observed at the maximum value of Zipf. On the contrary, for value 0.01 we have the most CPU time, while the intermediate values do not show any significant fluctuation. The memory consumption appears to be the same. Finally, the Pareto distribution does not seem to affect the memory consumption neither the CPU time. To sum up, the dominant factor that increases the memory usage in CDNsims is the number of objects, while the CPU time is affected by the performance of cache replacement policy.

- the redirection policies: CDNsims offers the following request redirection policies: cooperative closest server, cooperative server load balance, cooperative random server selection and non-cooperative. These policies can be combined with the options of having a pull or push based outsourcing scheme given the initial surrogate server content. Table V summarizes the memory usage and CPU time of the experiments conducted with all the redirection policies. We used 1000 routers, 100 surrogate servers, 50000 clients, 1000000 requests, a Web site of 3000 objects; the surrogate server caches are initially empty (pull based) running LRU. From the experiments it occurs that the observed values for memory consumption are quite similar for all the policies. However, this is not the case for the CPU time. The random server selection has the highest execution time, since the requests are shared without logic causing network traffic. The load balance policy is the second highest; at every request a search is performed for the least loaded server. The best time is observed to the closest surrogate policy, since all the inter-surrogate server distances are precalculated speeding up the search for the closest surrogate server. The non-cooperative policy does not include any search; it redirects every request to the origin server, upon a cache miss, leading to high traffic close to the origin server.

<i>Random</i>	<i>Memory (MB)</i>	<i>CPU time (sec)</i>
3037 routers, 23277 links	1895.93	25964
4037 routers, 41069 links	3154.98	25712
5037 routers, 63687 links	4578.79	25483
6037 routers, 91110 links	6308.37	26706
<i>AS</i>	<i>Memory (MB)</i>	<i>CPU time (sec)</i>
3037 routers, 4789 links	1046.44	32122
4037 routers, 6720 links	1653.16	33854
5037 routers, 8768 links	2331.57	35858
6037 routers, 10931 links	3142.44	36128
<i>Transit stub</i>	<i>Memory (MB)</i>	<i>CPU time (sec)</i>
3192 routers, 6776 links	1557.77	42666
4104 routers, 8638 links	2377.88	39459
5256 routers, 12062 links	3548.08	41701
6642 routers, 15169 links	5206.1	43059

Table III. Network topology flavors.

<i>requests</i>	<i>objects</i>	<i>Zipf</i>	<i>Pareto</i>	<i>Distinct objects %</i>	<i>Memory (MB)</i>	<i>CPU time (sec)</i>
985810	100000	0.75	1.2	10	1416.61	6509.07
990792	300000	0.75	1.2	30	3253.33	7871.14
920694	800000	0.75	1.2	80	7887.69	9032.77
970446	300000	0.01	1.2	30	3254.27	8920.26
965839	300000	0.25	1.2	30	3252.95	7862.26
954703	300000	0.5	1.2	30	3247.24	7414.79
990792	300000	0.75	1.2	30	3253.33	7871.14
950211	300000	1	1.2	30	3255.18	6787.9
990792	300000	0.75	1	30	3234.03	7659.38
990792	300000	0.75	1.2	30	3253.33	7871.14
990792	300000	0.75	2	30	3256.06	7144.15

Table IV. Request stream parameters.

<i>Redirection policy</i>	<i>Memory (MB)</i>	<i>CPU time (sec)</i>
Cooperative closest server	563	18323
Cooperative server load balance	556	26461
Cooperative random server selection	569	29885
Non-cooperative	557	19322

Table V. Redirection policies.

5.3 Summary

The simulation of a large-scale CDN is a memory and compute-intensive task. The memory-intensive nature arises from the need to simulate a disk cache at each surrogate server. The larger the number of objects the greater the memory requirements. This also happens to the network size. A larger network requires more memory. Finally, the memory usage is also depended on the number of requests, since they are loaded into memory to speed up their retrieval.

On the other hand, the main factors that dominate the CPU requirements can be summarized as follows:

- The event scheduling. The basic simulation element that advances the simulated time (not the wall clock time) in OMNeT++ and, consequently, in CDNSim is the *Message*. Messages are considered as events in simulation terminology. All the information passing, signaling and object transmission is performed in the form of a Message. OMNeT++ is responsible to schedule the Messages according to their timestamps and priorities. During a typical CDNSim simulation, several millions of Messages are generated. It is evident that the scheduler's complexity can easily be a bottleneck to the performance. OMNeT++ implements the binary heap structure for Future Event Set (FES) scheduling, which is a standard structure in discrete event simulation systems demonstrating the best performance in most cases.
- The client request scheduling. Each client maintains a queue of waiting requests to be performed. The requests are sorted by their time stamp, once at the beginning of the simulation. Every new request (i.e. retry) is added at the head of the queue. Therefore, the complexity of removing/adding a single request is $O(1)$. The overall time scales linearly as the number of requests increases.
- The request life cycle. By the time a new request is generated, it is being processed by many modules. The goal of CDNSim is to handle millions of requests efficiently. Therefore, each request is passed only by reference between modules, avoiding redundant copies (memory and CPU friendly). Each module is responsible to promote the request to the next processing module.
- Cache management. The surrogate server caches are usually a hot spot since millions of accesses are performed. The performance of the cache depends on the cache management algorithms implemented and the respective data structures. CDNSim uses priority queues for the major cache replacement algorithms (LRU, LFU, SIZE).

To sum up, our experiments have shown that CDNSim scales linearly in terms of network size and number of requests. Using a common commercial PC with 2 GB of RAM the user is able to run simulations with more than 5000 nodes and several millions of requests within a few hours.

6. CDNSIM: FROM THE USER'S PERSPECTIVE

CDNSim provides a set of utilities for preparing and executing the simulation environment. The simulation environment includes all the necessary input files, configurations, libraries and executables to perform a simulation. Furthermore, CDNSim provides utilities for extracting statistics results. Figure 11 depicts the phases that a user may follow in order to simulate a CDN. These phases are described in detail in the following subsections.

6.1 Phase 1 - Simulation Environments Preparation

During the first phase, the user is driven by a wizard to prepare the simulation environments to be executed. All these are bundled into a compressed archive, the so called *bottle*. Bottles are self-contained simulation environments ready to be executed. However, the procedure of creating manually input for the simulator is a cumbersome task since a lot of parameters and files are involved. The CDNSim wizard saves significant time by organizing the basic parameters and

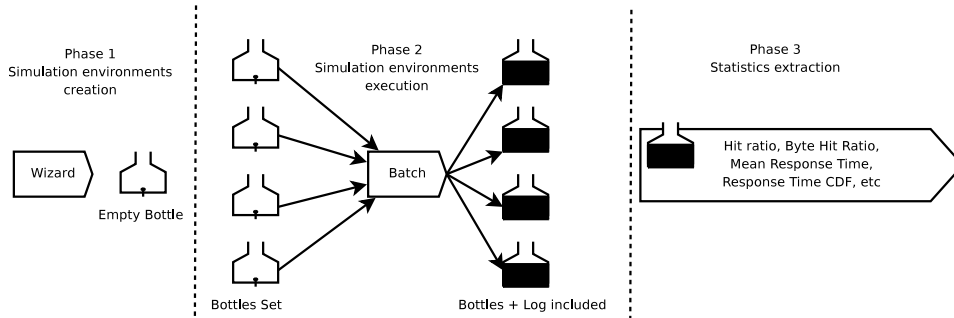


Fig. 11. CDNsim: From the user's perspective.

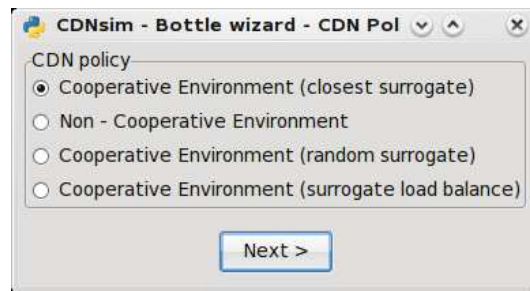


Fig. 12. CDN policy management.

by validating user input making the bottles creation an easy task. The wizard is written in Python (<http://www.python.org/>) and the GUI components in wxPython (<http://www.wxpython.org/>), a blending of the wxWidgets C++ class library (<http://wxwidgets.org/>) with the Python programming language. Figures 12-16 depict the CDNsim wizard running on Linux with KDE 4.0.1.

In the first step (Figure 12), the user selects whether the CDN surrogate servers will cooperate with others upon cache misses or not. There are four different flavors concerning the Cooperative policy namely “closest surrogate”, “random surrogate” and “surrogate load balance”. In the second step, depicted in Figure 13, the network topology is defined. The user provides a graph file describing the network backbone and the links speed in Mbps [Zegura et al. 1996]. The user may further tune the placement of the surrogate servers in the network by altering the corresponding network NED file. There are options concerning the number of connections for consuming services (outgoing) and the number of connections for serving (incoming). Moreover, the user may optionally define how the clients should behave upon a Denial of Service, meaning whether to retry or not. Network topology parameters can be further tuned including TCP / IP options by modifying the appropriate configuration files inside the bottle. The user is not limited to a specific network type since any network flavors can be used as input.

In the next step (Figure 14), the user should input a file that describes the objects' attributes and a file that represents the traffic generated by the clients during the simulation. The first one includes information such as the size of an object while

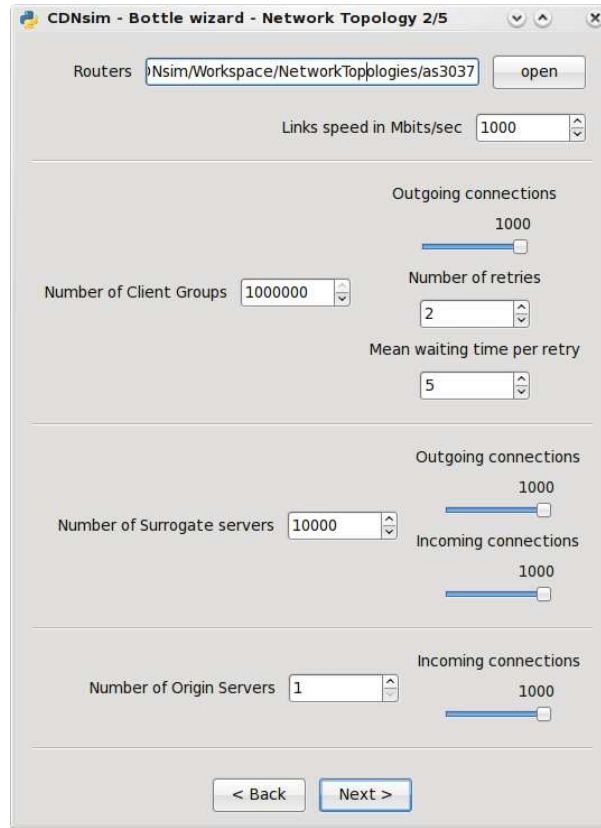


Fig. 13. Network topology management.

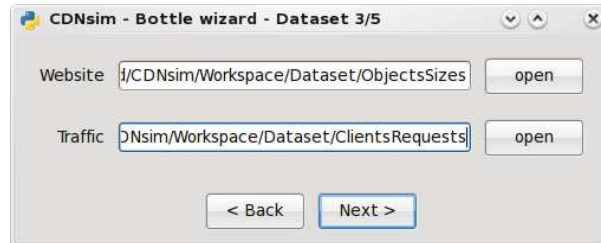


Fig. 14. Data sets.

the second one includes the timestamps of the clients' requests. These files can be either real (i.e. Apache logs and real web site) or artificially generated by any third party tool (e.g., Medisyn [Tang et al. 2003]).

In the fourth step, the surrogate servers cache attributes are configured. The user inputs a configuration file that describes the content, capacity and cache outsourcing policy (push/pull) of each surrogate server. This enables user to configure a CDN simulation setup having multiple outsourcing policies. As depicted in Figure 16,

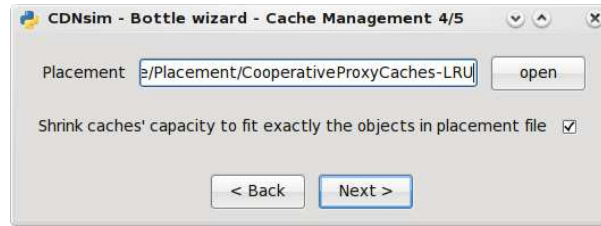


Fig. 15. Content management.

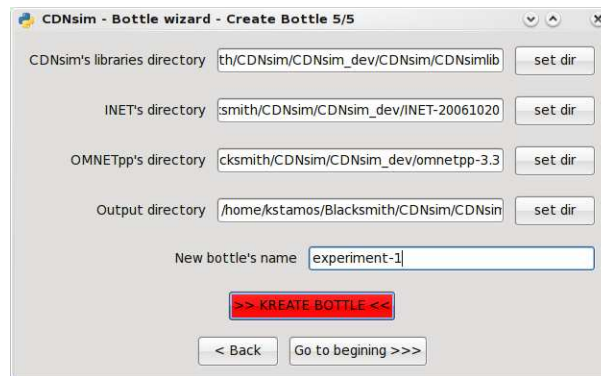


Fig. 16. Bottles creation.

the path of CDNsim libraries, OMNeT++ installation and INET installation are required. CDNsim libraries model all the CDN infrastructure and they are loaded at runtime by the INET's main executable along with the OMNeT++ libraries. Furthermore, the output directory must be set and a representative name for the bottle is created. By pressing the “CREATE BOTTLE” button all the input is verified and a bottle is created.

To sum up, these four steps can be followed many times in order to create a set of bottles. Advanced tuning of simulation can be performed by editing the bottles content.

6.2 Phase 2 - Simulation Environments Execution

Assuming that we have created a large amount of bottles, we need to execute each one of them to get the simulations results. This task is time consuming to be performed manually by the user. To increase the user's productivity we offer a shell script, which processes sequentially all the bottles with the following procedure: Each bottle is uncompressed, the respective simulation is executed and the bottle is re-compressed including the simulation output. Using this script the user is able to execute a large number of unattended simulations. All the script activities are kept in log files.

CDNsim offers also developers a way to inspect the simulator internals for debugging as well as an attractive way to present a model. This is supported by the graphical environment which is provided by OMNeT++. A sample is depicted

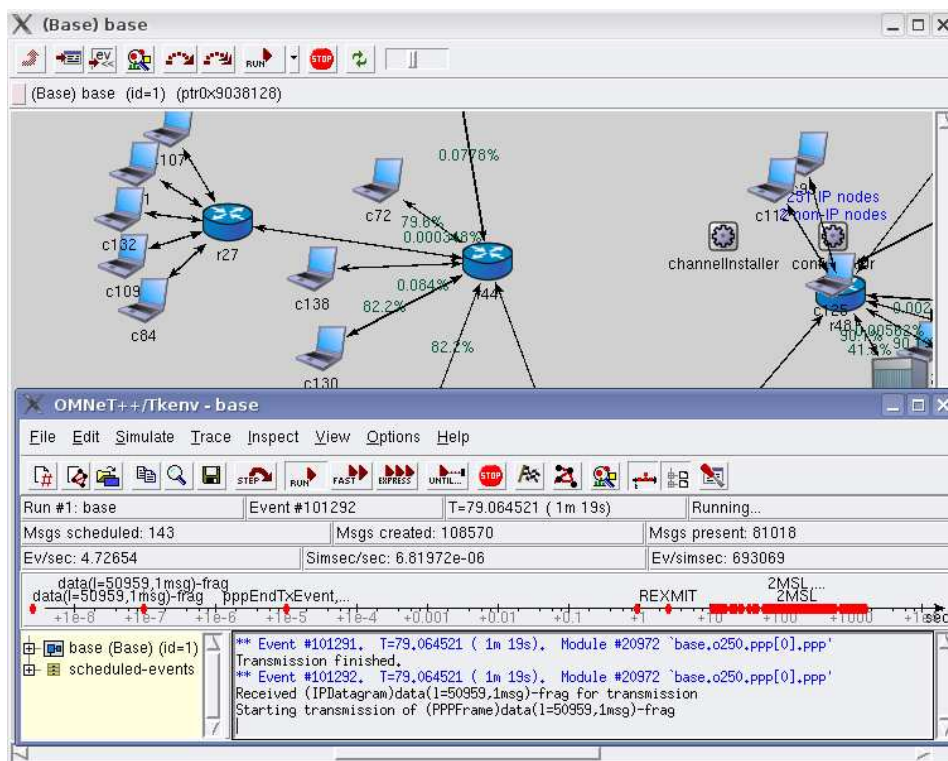


Fig. 17. Simulation visualization.

in Figure 17. The user is able to start, pause, speed up/down a simulation. The network is depicted as interconnected nodes of which the internal operations can be inspected. Network activity is animated; so the user is able to watch the packet transmissions and examine the content of each packet, bandwidth consumption, etc. Further information can be found in the OMNeT++ manual.

6.3 Phase 3 - Statistics Extraction

The final phase includes the statistics extraction. Once a bottle is executed, the simulation log is recorded. The log can be considered as a flat file containing raw information about the state of the simulator at various timestamps and events. The resulting log is parsed by the appropriate utility to produce the statistics, at the end of the simulation. Since we deal with raw data, one may produce statistics other than the defaults provided by the tool, or even apply data mining techniques. The default statistics are summarized in the following paragraphs.

Client side statistics.

These statistics refer to the clients activities, i.e. requests for objects. Those are:

- Number of satisfied requests. This is the total number of client-to-CDN requests that served successfully. Not all requests are satisfied due to Denial of Service, caused by increased surrogate servers load.

- Number of failed requests. This is the number of client-to-CDN requests that has not been served. To reduce this number, one may reduce the mean interarrival time of the requests, increase the surrogate servers connections and increase the network speed.
- Mean response time. This is a measurement that indicates how fast a client is satisfied. It is defined as $\frac{\sum_{i=0}^{N-1} t_i}{N}$, where N is the number of satisfied requests and t_i is the response time of i^{th} request. The response time starts at the timestamp when the request begins and ends at the timestamp when the connection is closed.
- Response time CDF. The Cumulative Distribution Function (CDF) denotes the probability of having response times lower or equal to a given response time. The goal of a CDN is to increase the probability of having response times around the lower bound of response times.
- Requests distribution through time. It is a representation of the requests response time through time. The x axis represents the requests sorted by their start timestamp. The y axis is the response time of the corresponding requests. This is especially useful since we may detect time related phenomena and easily observe problematic behaviors during a flash crowd event.
- Mean retries. Upon a Denial of Service the client should perform an action. Either the denied request is considered failed or a retry is performed according to the configuration (Figure 13). The mean retries is defined as $\frac{\sum_{i=0}^{C-1} r_i}{C}$, where C is the number of clients and r_i is the number of retries the i^{th} client performed. Values above zero suggest network performance issues.
- Mean waiting time. Between retries, the clients wait for a specified amount of time according to an exponential distribution configured by the wizard (Figure 13). This emulates a client that “hits“ the reload button of the web browser randomly when the connection timeouts. The Mean waiting time is defined as $\frac{\sum_{i=0}^{C-1} w_i}{C}$, where w_i is the total waiting time of the i^{th} client.

Server side statistics

These statistics are focused on the operations of the surrogate and origin servers servers. In summary:

- Hit ratio. It is the percentage of the client-to-CDN requests that resulted in a cache hit. High values indicate high quality content placement of the surrogate servers.
- Byte hit ratio. This is the hit ratio expressed in bytes, meaning that instead of requests we count the corresponding bytes of the requests. High values indicate optimized space usage and lower network traffic.
- Load. It refers to the percentage of the connections that are active serving clients on the average. Each network element has a finite connections capacity, i.e. the number of clients that can be served simultaneously. Values close to 0.9 indicate an unstable system.
- Origin requests percentage. It refers to the percentage of the satisfied requests that redirected to the origin server. Low values indicate good CDN performance and accurate content selection.

Network statistics

All network operations run on top of TCP/IP. Therefore, several measurements can be extracted concerning the network topology. Specifically:

- Handshake time. This is the time required for a connection to be opened. It includes the classic three way handshake. During a flash crowd event the values are significantly higher.
- Bit error rate. CDNSim is able to simulate transmissions including packet errors. This adds to the realism of the model and it can be recorded for statistical analysis.
- Network delay. It is the amount of time the arrival of the message is delayed when it travels through a channel. Propagation delay is specified in seconds.
- Data rate. The data rate is specified in bits/second, and it is used for transmission delay calculation. The sending time of the message normally corresponds to the transmission of the first bit, and the arrival time of the message corresponds to the reception of the last bit.
- Net utility. It is a value that expresses the relation between the number of bytes of the served content against the number of bytes of the pulled content (from origin or other surrogate servers) [Mortazavi and Kesidis 2006]. It is bounded to the range $[0, 1]$ and provides an indication about the CDN performance. High net utility values indicate good content outsourcing policy and improved mean response times for the clients. Further details are available in the next section.

7. CDNSIM: USE CASES

7.1 Flash Crowd Event

In this subsection we demonstrated a flash crowd event simulated by CDNSim along with the respective results and model behavior. In general, during a flash crowd event, significantly higher response times are expected. CDNSim captured this behavior correctly. In this context we built an AS network topology consisting of 100 surrogate servers, 1 origin server, 3037 routers and 39847 clients. The network links were 1 Gbps. The trace file contained 286758 requests. More specifically, the trace file was equally split into three consecutive parts, namely epochs. The first epoch (pre flash crowd event) contained requests having mean interarrival time of 0.6 seconds. The second epoch (flash crowd event) contained requests having mean interarrival time of $\frac{0.6}{100}$ seconds, effectively causing a massive wave of requests to the CDN. The third epoch (post flash crowd event) reflected the relaxation phase right after the flash crowd event. The mean interarrival time was 0.6 sec.

Figure 18 depicts the evolution of a flash crowd event through time. The x axis represents the requests sorted by their timestamp (the time a request has been submitted to CDN), while the y axis is the respective response time. As time progresses, during the flash crowd event (epoch 2) the observed response times are getting higher and unstable. A peak is reached at the middle of the flash crowd event. The system becomes less loaded reaching stability again as entering the third epoch, showing normal activity like the first epoch. The CDF of the response time is illustrated in Figure 19. The x axis refers to the response time, while the y axis to the probability of having lower response time than a given value in the

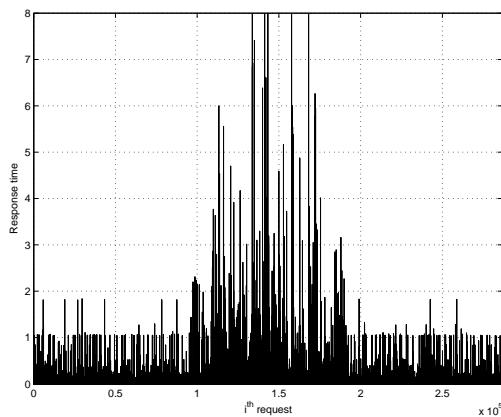


Fig. 18. Flash crowd event through time.

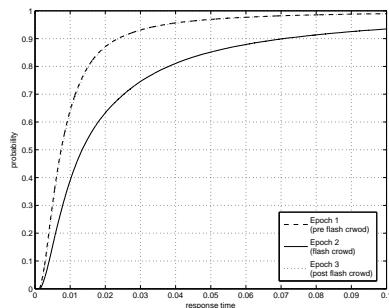


Fig. 19. Response time CDF.

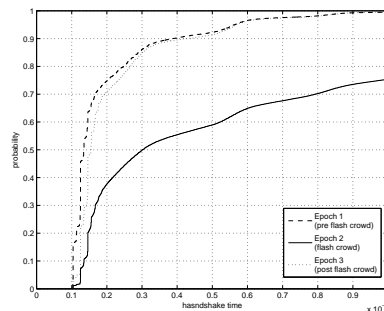


Fig. 20. Handshake time CDF.

x axis. The continuous line shows the pathological behavior of the network as the probability of having low response time is quite lower than epochs 1,3.

Another characteristic of the flash crowd events is the increased time for connection establishment. This measurement is available through the TCP sockets used by the clients in order to connect with the CDN. Figure 20 is the illustration of the handshake times CDF. The x axis refers to the handshake time, while the y axis to the probability of having lower handshake time than a given value in the x axis. As expected, the probability of a quick connection establishment is significantly lower during epoch 2. CDNsim managed to capture this network behavior.

7.2 CDN pricing

In this subsection, we present a scenario where a CDN has a contract with a Web content provider. The CDN outsources content on behalf of content provider and charges according to a usage based pricing function. The ultimate goal is to identify the final cost for the content provider under a CDN infrastructure. In this context, we are mainly focused on capturing the CDN usage via a *net utility* measure. Then,

the respective net utility of the CDN can be easily translated into a price for the offered services.

The monetary cost of the Web content provider is defined in [Hosanagar et al. 2006] using equation (1) below, where: U_{CDN} is the final cost of Web content provider under a CDN infrastructure, $V(X)$ is the benefit of the content provider by responding to the whole request volume X , $\tau(N)$ is the benefit per request from faster content delivery through a geographically distributed set of N CDN surrogate servers, C_o is cost of outsourcing content delivery, $P(u)$ is the usage-based pricing function, and u is the CDN net utility.

$$U_{CDN} = V(X) + \tau(N) \times X - C_o - P(u) \quad (1)$$

Regarding the usage-based pricing function, we should define the CDN net utility. We quantify a net utility u_i of a CDN surrogate server i by using equation (2) below, see, e.g., [Mortazavi and Kesidis 2006] for a similar utility for a p2p system. The intuition of this metric is that a surrogate server is considered to be useful (high net utility) if it uploads content more than it downloads, and vice versa. The parameter ξ is the ratio of the uploaded bytes to the downloaded bytes. The resulting net utility ranges to $[0, 1]$. The value $u_i = 1$ is achieved if the surrogate server uploads only content ($\xi = \infty$). On the contrary, the value $u_i = 0$ is achieved if the surrogate server downloads only content ($\xi = 0$). In the case of equal upload and download ($\xi = 1$), the resulting value is $u_i = 0.5$. Finally, the CDN net utility u can be expressed as the mean of the individual utilities of each surrogate server.

$$u_i = \frac{2}{\pi} \times \arctan(\xi) \quad (2)$$

In order to observe the net utility in action, we setup a simulation with 100 surrogate servers, 1008 routers, a Web content provider with 300000 pages and 1000000 requests. The redirection policy was set to the cooperative closest surrogate server and the caches were, initially, all empty running LRU. Figure 21 records the net utility of a surrogate server through time. The x axis represents the order of the requests sorted by their time of arrival, while the y is the respective net utility using equation 2. As expected, there is a warm-up phase at the beginning of the curve where the cache starts to fill with objects. The initial net utility is 0.5 as no content has been uploaded or downloaded and at a short time period it is below 0.5 reflecting the initial content pull to fill the cache. As the time progresses, the net utility is improved since the cache hit ratio is improved leading to less downloads. The curve reaches a plateau at 0.77 giving indications about the performance limits given the current configuration and traffic. The CDN net utility is the mean of all the measurements of all surrogate servers and in our case is 0.5859 with standard deviation 0.0340.

8. CDNSIM IN PRACTICE

CDNsim can be used for evaluating the performance of a CDN infrastructure. Its simulation environment allows researchers and software practitioners to develop state-of-the-art policies as well as address new research pathways in the area of

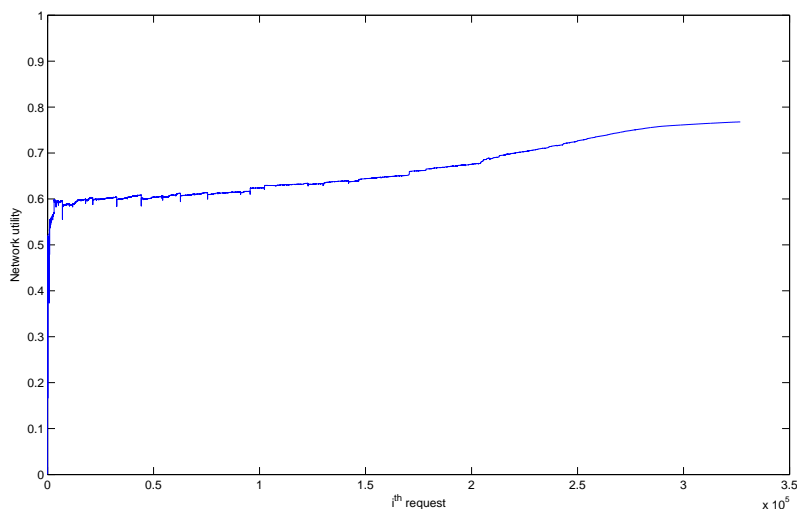


Fig. 21. Net utility of a surrogate server through time.

CDNs. Moreover, the analysts of a real CDN provider are able to inspect a content service during its pre/post-service release life cycle.

8.1 *CDNsim* Validation

According to [Sargent 2005], in order to validate a simulated model, it should be compared with another reference model either real or simulated (provided that it is validated). In the context of CDNs, several measures should be considered for benchmarking, such as mean response time, cache hit ratio and byte hit ratio. Advanced network monitoring should also be performed including TCP/IP traffic, DNS redirection, and internode interaction.

Considering that there is not any CDN simulation model in the literature (more details in section 2) which shares common characteristics with *CDNsim* (e.g., supports TCP/IP networking), no validation can be performed. Another approach is to validate *CDNsim* with the existing academic CDNs (CoDeeN, CoralCDN, Globule). As we referred in the related work section, CoDeeN and CoralCDN infrastructures are built upon Planetlab. However, these testbeds cannot be used for validating *CDNsim*, since we do not have full knowledge of Planetlab network topology. Specifically, there cannot be a precise bandwidth measurement and network topology structure mapping for the Planetlab infrastructure. This problem is an open research issue [Lee et al. 2005] and it is out of the scope of this paper. Thus, both CoDeeN and CoralCDN are treated as “black boxes”; we are not aware of the DNS redirections, cache hits and misses, or even details such as the maximum connections per node etc. Globule is also treated as “black box” since it has been implemented as a third-party module for the Apache HTTP server. Globule requires individuals to install voluntarily the module to their machines. Moreover, we are still not able to extract a precise network topology out of the

Globule CDN. Therefore, no comparison with CDNSim can be performed due to insufficient knowledge of the reference system.

Thus, CDNSim has been validated by the OMNeT++ community [Varga a] since CDNSim has been built upon the OMNeT++ framework. Actually, it has been announced by the official site of OMNeT++. CDNSim reliability is also reflected by the fact that it has been a growing number of publications ¹ that used CDNSim in their experimentation. Therefore, we believe that CDNSim has reached a level of maturity that enables researchers to use it for production.

8.2 From the Perspective of Researchers

CDNSim has been used to the following CDN research issues providing new insights and perspectives:

- **Content Delivery Practices:** Several issues are involved in CDNs since there are different decisions related to where to locate surrogate servers, which content to outsource, and which practice to use for (selected content) outsourcing. It is obvious that each decision for these issues results in different costs and constrains for CDN providers. In this framework, CDNSim has been used to evaluate a wide range of policies [Pallis et al. 2005; Pallis et al. 2006; Sidiropoulos et al. 2008]. Furthermore, CDNSim has been used in order to explore the benefits of caching in a CDN infrastructure [Stamos et al. 2006]. In such an approach, the surrogate servers may act simultaneously both as proxy servers and content replicators.
- **Pricing of CDN Services:** Pricing of CDN services is a challenging problem faced by managers in CDN providers. Deployment of new services, such as *Edgesuite*, are accompanied with open questions regarding pricing and service adoption. The authors of [Hosanagar et al. 2006] developed an analytic model to analyze optimal pricing policies for CDNs. This model extracts useful conclusions for the infrastructure of CDNs. In [Hosanagar et al. 2006], CDNSim has been used in order to prove that the above conclusions can be validated under a realistic simulation environment.
- **Peering of CDNs:** Peering of CDNs is gaining popularity among researchers of the scientific community. Several approaches are being conducted for finding ways for peering CDNs. However, several critical issues (i.e., When to peer? How to peer? etc.) should be addressed. In [Pathan 2007], the authors present a novel architecture of a Virtual Organization (VO) based model for forming peering CDNs. CDNSim will be used to demonstrate the behavior and effectiveness of the developed policies to ensure effective peering among CDNs. It can also be utilized to evaluate the best practices and new techniques for load measurement, request redirection and content replication in the proposed framework for peering CDNs. According to the authors, CDNSim is suitable to simulate the peering CDN framework under realistic traffic, workload and replication conditions.

CDNSim might also offer new perspectives for researchers in order to evaluate and validate their proposed approaches. Some indicative applications where CDNSim would be used as a simulation testbed could be the following:

¹A list of the publications where CDNSim is involved can be found in <http://oswinds.csd.auth.gr/~cdnsim/#Publications>

- Security in CDNs:** The rapid growth of business transactions conducted on the Internet has drawn much attention to the problem of data security in CDNs [Yao et al. 2007]. In this context, secure content delivery protocols should be proposed in order to maintain content integrity (the delivered content which is modified by unauthorized entities should not be accepted) and confidentiality (the delivered contents cannot be viewed by unauthorized entities, including unauthorized proxies and other users besides the requester) in CDNs. The high extensibility of CDNsim allows researchers to adapt the proposed protocols (e.g., iDeliver [Yao et al. 2007]) under its infrastructure.
- CDNs on the sensor Web:** Content delivery on the sensor Web is a topic of emerging interest and importance in the academic and industrial communities [Balazinska et al. 2007]. In general, the sensor Web is a distributed sensing system in which information is globally shared and used by wired and wireless platforms. Considering that the CDN infrastructure may be either wired or wireless, CDNs or CDN-like overlay networks will play a key role in the evolution of large-scale sensor network deployments. Specifically, the worldwide sensor Web requires a distributed data management infrastructure, such as a CDN, since sensors are geographically distributed and produce data at high rates. Sensor data will be stored near its source, and data processing and filtering will be pushed to the edges. Thus, such overlay network structures may facilitate and optimize the management and delivery of static or streaming content over sensor Web. In addition, such architecture will reduce bandwidth requirements, enable parallel processing of sensor feeds, and finally achieve a delicate balance among the load of sensors.
- Mobile CDNs:** The recent advances in mobile content networking (e.g. GSM/3G, WiFi, etc.) enable the wireless network infrastructures to support bandwidth-intensive services such as streaming media, mobile TV etc. Taking into account that mobile user appetites for information is expected to keep growing, we need innovative techniques that can improve information dissemination. In this context, mobile CDNs are deployed within the range of a wireless network (e.g. cellular network, WiFi) and offer high quality services for delivering dynamic data and rich multimedia content to mobile devices [Eriksson et al. 2008; Louloulides et al. 2008]. Specifically, the network infrastructure in mobile CDNs is de-composed in the two following components: a) the wired network infrastructure and b) the wireless network infrastructure. The former is an infrastructure responsible for the wired environment of CDN; it provides the communication links which connect origin servers with surrogate servers and surrogate servers with network elements (e.g. switches, routers, 3G/GSM enabled base stations (BS), Wi-Fi enabled Access Points (AP)). On the other hand, the wireless network infrastructure is responsible for enabling communication and information dissemination among static and mobile users in the wireless environment of mobile CDN. CDNsim can be used as a testbed for simulating the wired network infrastructure of a mobile CDN. CDNsim can also be extended through the development of new add-on modules that will allow the support of mobile CDNs. For instance, there are various mobile, ad-hoc and sensor simulation frameworks based on OMNeT++.

—**P2P, GRID and Agent Technologies in CDNs:** Since CDNs are complex large-scale distributed systems, their development may be supported by the new emerging technologies of P2P, GRID and Agents, which respectively offer dynamism, robustness, and intelligence [Fortino and Russo 2007]. The integration of such technologies is a challenging issue which is being undertaken in several Web application domains, such as distributed information retrieval and data mining [Luo et al. 2007], large-scale service-oriented systems for the semantic Web [Li et al. 2004], and provision of multimedia services [Bruneo et al. 2005]. The successful exploitation and integration of these paradigms and technologies under a CDN infrastructure would provide an efficient way to cope with the aforementioned issues and would contribute significantly to the development of more efficient CDNs. The CDNs_{sim} architecture can easily enhance the aforementioned emerging technologies.

8.3 From the Perspective of Software Practitioners

CDN providers are interested in maximizing the benefit of their network infrastructure. To achieve this, the software practitioners design proprietary algorithms that manage the content effectively. The natural derivative of such activity is the creation of a new product. In the context of CDNs, the product usually is a new content delivery service, like streaming video, large files delivery, etc. Although each service² may differ from the others in terms of functionality, a common set of periods in the life time of every service can be identified, where CDNs_{sim} can be of use :

- before service release:* This period includes the development process of the service before its release to the users. CDNs_{sim} could be of use at the early development stages. It can be used to design and implement prototypes giving shape to the initial product ideas. Once the prototyping is done, it can be used to perform an *in vitro* evaluation of the performance and behavior under various network configurations and traffic patterns. CDNs_{sim} could significantly reduce the infrastructure investments during the stages of testing and prototyping until a certain level of maturity is reached. Then, evaluation is performed at the real CDN infrastructure. A real world example of the concept of prototyping and testing that could potentially be performed by CDNs_{sim} is the recent High Definition Video streaming by AKAMAI. Another tool that can be used to predict the performance and the execution time of a distributed application is P2PPerf [Ernst-Desmulier et al. 2007].
- after service release:* The service, by the time of its release to the wider public, should have passed a set of testing suites. Additionally, there is a set of documented conclusions about its behavior and performance. However, as the product is being used under untested circumstances, the behavior may divert from the initial conclusions. CDNs_{sim} may be used to reproduce a problematic or unexpected situation aiding the analysts to explain *why* an observed behavior is reached. Therefore, CDNs_{sim} could be used for continuous evaluation without disrupting the deployment of the service. Since the environment where a service

²Using the term *service* we refer to any content delivery service.

runs is not static, CDNsim might act as a preventing mechanism of unwanted situations before they happen. For instance, the necessity of behavior prediction and disaster prevention is apparent before a worldwide broadcast of soccer world championship by Limelight Networks [LimeLight].

—*service evolution in time*: Eventually, a service will reach a certain level of maturity, stability, and correctness. However, the “habitat” of service (network configurations, typical user populations, current technologies) is constantly evolving. A representative example is the increment of fast Internet connections and the fact that IPv6 will become a necessity since the available IP addresses are reducing. CDNsim could be used to perform a *what-if* analysis. How does the service scale with larger user populations? Can the service and the existing infrastructure keep up with much faster connections currently not available? These questions could be addressed by setting up the respective network configurations in CDNsim. Failing to predict the long-term evolution could result in loss of clients by not investing on upgraded infrastructure in time.

9. CONCLUSION

With the emergence of the Web, the leading use of the Internet has become the content delivery. In this context, CDNs have appeared to provide a delicate balance between costs (for Web content providers) and quality of services (for Web customers). Considering that CDNs are in their early development, a vast number of research efforts has mainly focused on developing Web data management policies on infrastructure of CDNs. However, there has not yet been developed a reliable, efficient and scalable simulation system which will simulate in great detail the CDN infrastructure. Thus, this work comes to cover this gap, presenting an efficient simulation tool for CDNs.

To sum up, CDNsim opens new perspectives to the research community since it is the first simulation tool for CDNs. Specifically, CDNsim has been designated to provide a realistic simulation for CDNs, simulating the surrogate servers, the TCP/IP protocol and the main CDN functions. The main advantages of this tool are its high performance, its extensibility and its user interface which is used to configure its parameters.

10. ACKNOWLEDGMENTS

The authors appreciate and thank the anonymous reviewers and CDNsim users for their valuable comments and suggestions, which have considerably contributed in improving CDNsim.

REFERENCES

- AKAMAIREPORT. 2008. The state of the internet, volume 1, number 2, 2nd quarter 2008. <http://www.akamai.com/stateoftheinternet/>.
- ALZOUBI, H. A., RABINOVICH, M., AND SPATSCHECK, O. 2007. MyXDNS: a request routing DNS server with decoupled server selection. In *Proceedings of the 16th international conference on World Wide Web*. 351 – 360.
- ARMOUR-BROWN, C., FITZHARDINGE, J., HUGHES, T., NETHERCOTE, N., MACKERRAS, P., MUELLER, D., SEWARD, J., ASSCHE, B. V., WALSH, R., AND WEIDENDORFER, J. Valgrind instrumentation framework for building dynamic analysis tools. <http://valgrind.org/>.

- BALAZINSKA, M., DESHPANDE, A., FRANKLIN, M. J., GIBBONS, P. B., GRAY, J., HANSEN, M., LIEBHOLD, M., NATH, S., SZALAY, A., AND TAO, V. 2007. Data Management in the Worldwide Sensor Web. *IEEE Pervasive Computing* 6, 2 (April), 30–40.
- BEKTAS, T., CORDEAU, J.-F., ERKUT, E., AND LAPORTE, G. 2008. Exact algorithms for the joint object placement and request routing problem in content distribution networks. *Computers and Operations Research* 35, 12, 3860–3884.
- BEKTAS, T. AND OUVEYSI, I. 2008. *Lecture Notes in Electrical Engineering*. Vol. 9. Springer Berlin Heidelberg, Chapter Mathematical Models for Resource Management and Allocation in CDNs, 225–250.
- BENT, L., MICHAEL, R., GEOFFREY, V. M., AND ZHEN, X. 2004. Characterization of a large Web site population with implications for content delivery. In *Proceedings of the 13th international conference on World Wide Web*. 522 – 533.
- BRUNEO, D., ZAIA, A., AND PULIAFITO, A. 2005. Agent-based middleware to access multimedia services in a Grid environment. *Multiagent and Grid Systems* 1, 1 (January), 41 – 59.
- CHEN, Y., QIU, L., CHEN, W., NGUYEN, L., AND KATZ, R. H. 2003. Efficient and adaptive Web replication using content clustering. *IEEE Selected Areas in Communications* 21, 6 (August), 979 – 994.
- CLARK, D., SHENKER, S., AND FALK, A. 2007. Global Environment for Network Innovations. Technical report, Geni.
- CoDEEN. CoDeeN : A CDN for PlanetLab. <http://codeen.cs.princeton.edu>.
- CORAL. CORAL CDN. <http://www.coralcdn.org>.
- DHAMDHARE, A. AND DOVROLIS, C. 2008. Ten years in the evolution of the internet ecosystem. In *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, New York, NY, USA, 183–196.
- DIJKSTRA, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1, 1 (December), 269–271.
- ERIKSSON, J., BALAKRISHNAN, H., AND MADDEN, S. 2008. Cabernet: Vehicular Content Delivery Using WiFi. In *MobiCom 2008: Proceedings of the 14th ACM international conference on Mobile Computing and Networking*.
- ERNST-DESMULIER, J.-B., BOURGEOIS, J., AND SPIES, F. 2007. P2PPerf: a framework for simulating and optimizing peer-to-peer-distributed computing applications. *Concurrency and Computation: Practice and Experience* 20, 6 (July), 693 – 712.
- FALL, K. Network Simulators. <http://www-nrg.ee.lbl.gov/kfall/netsims.html>.
- FORTINO, G. AND RUSSO, W. 2007. Using P2P, GRID and Agent technologies for the development of content distribution networks. *Future Generation Computer Systems* 24, 3 (March), 180–190.
- HOSANAGAR, K., CHUANG, J., KRISHNAN, R., AND SMITH, M. 2006. Service adoption and pricing of Content Delivery Network (CDN) services. Tech. rep., Social Science Research Network.
- IDSERDA, J. 2004. TCP/IP modelling in OMNeT++. Tech. rep., University of Twente, The Netherlands.
- KANGASHARJU, J., ROBERTS, J., AND ROSS, K. W. 2002. Object replication strategies in content distribution networks. *Computer Communications* 25, 4 (March), 367 – 383.
- KATSAROS, D., PALLIS, G., STAMOS, K., VAKALI, A., SIDIROPOULOS, A., AND MANOLOPOULOS, Y. 2008. CDNs Content Outsourcing via Generalized Communities. *IEEE Transactions on Knowledge and Data Engineering*.
- KULKARNI, P., SHENOY, P. J., AND GONG, W. 2003. Scalable techniques for memory-efficient CDN simulations. In *Proceedings of the 12th International World Wide Web Conference*. 609 – 618.
- LAM, V. T., ANTONATOS, S., AKRITIDIS, P., AND ANAGNOSTAKIS, K. G. 2006. Puppetnets: misusing web browsers as a distributed attack infrastructure. In *ACM Conference on Computer and Communications Security (2006-12-11)*, A. Juels, R. N. Wright, and S. D. C. di Vimercati, Eds. ACM, 221–234.
- LAOUTARIS, N., ZISSIMOPOULOS, V., AND STAVRAKAKIS, I. 2005. On the Optimization of Storage Capacity Allocation for Content Distribution. *Computer Networks* 47, 409–428.
- Transactions on Modeling and Computer Simulation, Vol. V, No. N, November 2008.

- LEE, S., SHARMA, P., BANERJEE, S., BASU, S., AND FONSECA, R. 2005. Measuring Bandwidth between PlanetLab Nodes. Technical report, HP Labs.
- LI, B., DENG, X., GOLIN, M. J., AND SOHRABY, K. 1998. On the optimal placement of web proxies in the internet: The linear topology. In *HPN '98: Proceedings of the IFIP TC-6 Eighth International Conference on High Performance Networking*. Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 485–495.
- LI, M., VAN SANTEN, P., WALKER, D. W., RANA, O. F., AND BAKER, M. A. 2004. SGrid: a service-oriented model for the Semantic grid. *Future Generation Computer Systems* 20, 1 (January), 7 – 18.
- LIMELIGHT. limelightnet. <http://www.limelightcompany.com/>.
- LOULLOUDES, N., PALLIS, G., AND DIKAIKAKOS, M. D. 2008. Information Dissemination in Mobile CDNs. *Content Delivery Networks: Principles and Paradigms*, editors R. Buyya et. al., Springer.
- LUO, J., WANG, M., HU, J., AND SHI, Z. 2007. Distributed data mining on agent grid: issues, platform and development toolkit. *Future Generation Computer Systems* 23, 1 (January), 61–68.
- MARKET. 2006. Content Delivery Networks, Market Strategies and Forecasts (2001-2006). Tech. rep., AccuStream iMedia Research.
- MORTAZAVI, B. AND KESIDIS, G. 2006. Model and simulation study of a peer-to-peer game with a reputation-based incentive mechanism. In *Information Theory and Applications Workshop*.
- NETHERCOTE, N. AND SEWARD, J. 2007. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*. ACM, New York, NY, USA, 89–100.
- NGUYEN, T. V., SAFAEI, F., BOUSTEAD, P., AND CHOU, C. T. 2005. Provisioning overlay distribution networks. *Comput. Netw.* 49, 1, 103–118.
- NS. The Network Simulator - ns-2. <http://www.isi.edu/nsnam/ns>.
- OLIVEIRA, C. A. S. AND PARDALOS, P. M. 2005. A survey of combinatorial optimization problems in multicast routing. *Comput. Oper. Res.* 32, 8, 1953–1981.
- OPPENHEIMER, D., ALBRECHT, J., PATTERSON, D., AND VAHDAT, A. 2004. Distributed resource discovery on PlanetLab with SWORD. In *Proceedings of the 1st Workshop on Real, Large Distributed Systems*.
- PALLIS, G., STAMOS, K., VAKALI, A., KATSAROS, D., SIDIROPOULOS, A., AND MANOLOPOULOS, Y. 2006. Replication based on objects load under a content distribution network. In Proceedings of the 2nd International Workshop on Challenges in Web Information Retrieval and Integration. *Proceedings of the 2nd International Workshop on Challenges in Web Information Retrieval and Integration*.
- PALLIS, G. AND VAKALI, A. 2006. Insights and perspectives for content delivery networks. *Communications of the ACM* 49, 1 (January), 101–106.
- PALLIS, G., VAKALI, A., STAMOS, K., SIDIROPOULOS, A., KATSAROS, D., AND MANOLOPOULOS, Y. 2005. A Latency-based object placement approach in content distribution networks. In *Proceedings of the 3rd Latin American Web Congress*.
- PATHAN, A.-M. K. 2007. Coordinated management and peering of content delivery networks. Technical draft, Grid Computing and Distributed Systems Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Australia.
- PIERRE, G. AND STEEN., M. 2006. Globule: a collaborative content delivery network. *IEEE Communications Magazine* 44, 8 (August), 127–133.
- PLANETLAB. Planetlab. <http://www.planet-lab.org/>.
- QIU, L., PADMANABHAN, N. V., AND VOELKER, M. G. 2001. On the Placement of Web Server Replicas. Tech. rep., Ithaca, NY, USA.
- RABINOVICH, M. AND SPATSHECK, O. 2002. *Web caching and replication*. Addison Wesley.
- RAMAMURTHY, P., SEKAR, V., AKELLA, A., KRISHNAMURTHY, B., AND SHAIKH, A. 2007. Using mini-flash crowds to infer resource constraints in remote web servers. In *INM '07: Proceedings of the 2007 SIGCOMM workshop on Internet network management*. ACM, New York, NY, USA, 250–255.

- SARGENT, R. G. 2005. Verification and validation of simulation models. In *Proceedings of the 37th Conference on Winter Simulation*. 130–143.
- SIDIROPOULOS, A., PALLIS, G., KATSAROS, D., STAMOS, K., VAKALI, A., AND MANOLOPOULOS, Y. accepted 2008. Prefetching in Content Distribution Networks via Web Communities Identification and Outsourcing. *World Wide Web Journal* 11, 1 (March), 39–70.
- SPRING, N., PETERSON, L., BAVIER, A., AND PAI, V. 2005. Using PlanetLab for network research: myths, realities, and best practices. In *Proceedings of the 2nd Workshop on Real, Large Distributed Systems*. 17–24.
- SRIPANIDKULCHAI, K., GANJAM, A., MAGGS, B., AND ZHANG, H. 2004. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proceedings of the ACM SIGCOMM 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*.
- STAMOS, K., PALLIS, G., THOMOS, C., AND VAKALI, A. 2006. A similarity-based approach for integrated Web caching and content replication in CDNs. In *Proceedings of 10th International Database Engineering and Applications Symposium*. 239–242.
- TANG, W., FU, Y., AND CHERKASOVA, L. 2003. MediSyn: A Synthetic Streaming Media Service Workload Generator. In *Proceedings of 13th International Workshop on Network and Operating System Support for Digital Audio and Video*. 12 – 21.
- VAKALI, A. AND PALLIS, G. 2003. Content delivery networks: Status and trends. *IEEE Internet Computing* 7, 6 (November), 68–74.
- VARGA, A. OMNeT++. <http://www.omnetpp.org/>.
- VARGA, A. OMNeT++ object-oriented discrete event simulation system user manual. <http://www.omnetpp.org/doc/manual/usman.html/>.
- VENKATARAMANI, A., YALAGANDULA, P., KOKKU, R., SHARIF, S., AND DAHLIN, M. 2002. The Potential costs and benefits of long term prefetching for content distribution. *Computer Communications* 25, 4 (June), 367 – 375.
- WANG, L., PAI, V., AND PETERSON, L. 2002. The effectiveness of request redirection on CDN robustness. In *Proceedings of the 5th Symposium on Operating System Design and Implementation*. 345 – 360.
- WANG, L., PARK, K., PANG, R., PAI, V., AND PETERSON, L. 2004. Reliability security in the CoDeeN content distribution network. In *Proceedings of the USENIX Annual Technical Conference*.
- WATTS, D. J. AND STROGATZ, S. H. 1998. Collective dynamics of 'small-world' networks. *Nature* 393, 6684 (June), 440–442.
- YAO, D., KOGLIN, Y., BERTINO, E., AND TAMASSIA, R. 2007. Decentralized authorization and data security in Web content delivery. In *Proceedings of the ACM Symposium on Applied Computing. Proceedings of the ACM Symposium on Applied Computing*, 1654–1661.
- YOUTUBE. Youtube. <http://www.youtube.com/>.
- ZEGURA, E. W., CALVERT, K. L., AND BHATTACHARJEE, S. 1996. How to model an internetwork. In *Proceedings of the Conference on Computer Communications, 15th Annual Joint Conference of the IEEE Computer and Communications Societies, Networking the Next Generation Proceedings*. 594 – 602.
- ZHANG, Y., ZHANG, Z., MAO, Z. M., HU, C., AND MAGGS, B. M. 2007. On the impact of route monitor selection. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, New York, NY, USA, 215–220.

APPENDIX: The OMNeT++ Framework

Objective Modular Network Test-bed in C++ (OMNeT++) is a public-source, component-based, modular simulation framework. It has been used to simulate communication networks and other distributed systems. The OMNeT++ model is a collection of hierarchically nested modules. The top-level module is called System Module or Network. This module contains one or more sub-modules each of which

could contain other sub-modules. The modules can be nested to any depth and hence it is possible to capture complex system models in OMNeT++.

Modules are distinguished as being either simple or compound. A simple module is associated with a C++ file that supplies the desired behaviors that encapsulate algorithms. Simple modules form the lowest level of the module hierarchy. Users implement simple modules in C++ using the OMNeT++ simulation class library. Compound modules are aggregates of simple modules and are not directly associated with a C++ file that supplies behaviors.

Modules communicate by exchanging messages. Each message may be a complex data structure. Messages may be exchanged directly between simple modules (based on their unique ID) or via a series of gates and connections. Messages represent frames or packets in a computer network. The local simulation time advances when a module receives messages from another module or from itself. Self-messages are used by a module to schedule events at a later time. The structure and interface of the modules are specified using a network description language. They implement the underlying behaviors of simple modules. Simulation executions are easily configured via initialization files, which track the events generated and ensure that messages are delivered to the right modules at the right time.

To take the advantage of the above features of OMNeT++ we have chosen it as the framework for the CDNsim. Its salient features include:

- It allows the design of modular simulation models, which can be combined and reused in a flexible way. This allows the modeling of various client types and network elements in CDNsim.
- It composes models with any granular hierarchy. This enables a detailed modeling of the various network elements, such as surrogate server caches and services.
- The object-oriented approach of OMNeT++ allows the flexible extension of the base classes provided in the simulation kernel. Following the same approach in CDNsim, a generic architecture is defined and all customized CDN elements are subclasses.
- Model components are compiled and linked with the simulation library, and one of the user interface libraries to form an executable program. One user interface library is optimized for command line and batch-oriented execution, while the other employs a graphical user interface (GUI) that can be used to trace and debug the simulation. This enables CDNsim to be used for mass experimentation, and careful step by step system monitoring.
- It offers an extensive simulation library that includes support for input/output, statistics, data collection, graphical presentation of simulation data, random number generators and data structures.
- OMNeT++ simulation kernel uses C++ which makes it possible to be embedded in larger applications.
- OMNeT++ models are built with the NED language and omnetpp.ini and do not use scripts which makes it easier for various simulations to be configured.
- INET, which is an extension of OMNeT++, offers a large suite of network protocols such as TCP/IP. Thus, we are able to design a CDN simulation environment as an overlay network on top of an Internet topology, just like the actual CDNs.