An Extended *ns-2* for Validation of Load Balancing Algorithms in Content Delivery Networks

Francesco Cece
Dipartimento di Informatica e Sistemistica
Federico II University of Napoli, Italy
ing.frankcece@gmail.com

Valerio Formicola
Dipartimento di Informatica e Sistemistica
Federico II University of Napoli, Italy
valerio.formicola@unina.it

Francesco Oliviero
Dipartimento di Informatica e Sistemistica
Federico II University of Napoli, Italy
folivier@unina.it

Simon Pietro Romano
Dipartimento di Informatica e Sistemistica
Federico II University of Napoli, Italy
spromano@unina.it

ABSTRACT

This paper deals with the design, the development and the usage guidelines of a novel Content Delivery Network library for the ns-2 simulator. Such library allows evaluating new application-level load balancing approaches, with special regard to distributed content web servers. It includes some typical load balancing algorithms proposed in the literature and it can be extended to support new solutions. The proposed tool extends the ns-2 simulator with new HTTP data types and new application components which are in charge of data treatment. Moreover a new agent has been added to allow the simulation of data transferring. The library has been designed to work in a non-hierarchical and peer to peer cooperation environment. Several examples of testing scenarios are proposed in the paper.

Keywords

Content Delivery Networks, Network Simulator, Load Balancing

1. INTRODUCTION

A Content Delivery Network (CDN) represents a solution to effectively provide contents to users by adopting a distributed overlay of servers. By replicating content on several servers a CDN is capable to partially solve congestion issues due to high client request rates, thus reducing latency at the same time increasing content availability. Usually, a CDN consists of an original server (called back-end server) containing new data to be distributed, together with one or more distribution servers, called surrogate servers. In some typical scenarios there is a server called redirector, which dynamically redirects client requests based on selected policies.

A critical component of a CDN architecture is the request routing mechanism. It allows to direct users requests for a content to the appropriate server, based on a specified set of parameters. The proximity principle, by means of which a request is always served by the server closest to the client, can sometime fail. Indeed, the routing process associated with a request might take into account several parameters (like traffic load, bandwidth and servers computational capabilities) in order to provide the best performance in terms of time of service, delay, etc.

In spite of his growing popularity, unfortunately only few solutions have been proposed in the last years for testing the CDN infrastructure. No suitable tools, in particular, have been designed for both implementation and simulation of novel solutions for load balancing.

In this paper we present an ns-2 extension for CDNs which allows the comparative evaluation of innovative mechanisms for load balancing based on a request redirection paradigm. New algorithms can be easily integrated in the simulator for testing purposes.

2. REQUEST ROUTING IN CONTENT DELIVERY NETWORKS

Content Delivery Networks were born to improve accessibility, while maintaining correctness: this is achieved through content replication. They involve an orchestrated combination of heterogeneous techniques, like content delivery, request routing, information spreading and accounting.

Depending on the network layers and mechanisms involved in the process, generally request routing techniques can be classified in DNS request routing, transport-layer request routing, and application-layer request routing [1]. In a DNS based approach, a specialized DNS server is able to provide a request balancing mechanism based on well-defined policies and metrics [2] [5] [7]. With transport-layer request routing, a layer 4 switch usually inspects information contained in the request header in order to select the most appropriate surrogate server. With application-layer request routing, the task of selecting the surrogate server is typically carried out by a layer 7 application, or by the contacted web-server itself. In particular, in the presence of a web-server routing mechanism the server can decide to either serve or redirect a client request to a remote node. Differently form the previous mechanism, which usually needs a centralized element, a web-server routing solution is usually designed in a distributed fashion. URL rewriting and HTTP redirection are typical solutions based on this approach. In this paper we will focus our attention on the application-layer request routing mechanism.

Request routing can usually be classified as either static or dynamic, depending on the policy adopted for server selection [3]. Static algorithms select the server without relying on any information about the status of the system at decision time. The simplest static algorithm is the Random (Rand) balancing mechanism. In such policy the incoming requests are distributed to the servers in the network with a uniform probability. Another well-known static solution is the Round Robin (RR) algorithm. This algorithm selects a different server for each incoming request in a cyclic mode. Dynamic load balancing strategies represent a valid alternative to static algorithms. Such approaches make use of information coming either from the network or from the servers in order to improve the request assignment process. For example, the Least-Loaded (LL) algorithm is a well-known dynamic strategy for load balancing. It assigns the incoming client request to the currently least-loaded server. Such approach is adopted in several commercial solutions. Unfortunately, it tends to rapidly saturate the least-loaded server until a new message is propagated [6]. Alternative solutions can rely on Response Time to select the server: the request is assigned to the server that shows the fastest response time [4]. We cite also the 2 Random Choices algorithm (2RC), which randomly chooses two servers and assigns a request to the least loaded one [8].

3. STATE OF THE ART IN CDN SIMULATORS

For the evaluation of the content delivery infrastructures some simulators have been proposed in the latest years, most of which refer to a web-services scenario. In the following of the section we will briefly introduce some of them.

To the best of our knowledge, CDNsim [1] is the only simulator for canonical content delivery networks. It has been created for modeling Cooperative/Non-Cooperative push/pull based content management policies in CDNs. The simulator provides: (i) a utility for converting Apache log files into CDNsim trace files, so to reproduce realistic request profiles, (ii) the possibility to set the LRU cache replacement policy, (iii) a static cache policy, and (iv) integration with TCP/IP networking. This simulator has been designed based on a hierarchical architecture with surrogate and origin servers. It also implements simple load balancing approaches, namely the random and the "least loaded" mechanisms. Unfortunately CDNsim does not provide the possibility to introduce new load balancing solutions and new overlay protocols. Moreover, peer-to-peer CDN scenarios cannot be evaluated.

Simulators for overlay peer-to-peer networks have been proposed in the last years. OverSim[2] is an OMNeT++[3] extension used to reproduce several kinds of overlay networks; it has been specifically developed for supporting P2P application experiments. The modular architecture of OverSim allows to simulate a full protocol stack, and it supports both structured and unstructured overlay architectures like Chord, Kademlia, Koorde, Broose and GIA. The OverSim code can be easily extended with its own architecture by implementing several common procedures for interaction with the simulator core engine.

Distributed web-services and web-caching can also be evaluated with ad hoc tools. The HttpTools[4] simulator is again an extension of OMNeT++ and provides modules to reproduce web-hosts behavior. Such modules adopt the underlying ONNeT++ architecture to exchange information. HttpTools reproduces fine-grained interactions among HTTP servers and clients. A centralized component, the HTTPController, enables a client to find a specific server. It also distributes randomly the requests among the servers based on either uniform or zipf distributions.

The network simulator ns-2[5] can also simulate HTTP traffic and web cache scenarios. The HTTP traffic generator module just simulates HTTP packet size profiles: this means that HTTP traffic is "virtual", since it does not actually simulate the transfer of application level data, but just the size and the time of such transfers. With the ns-2 Web-cache environment it is possible to reproduce real HTTP data. It implements HTTP client, server and cache applications as well as several common HTTP methods. Since the TCP protocol in ns-2 has not been realized to transfer real data, a new TCP-like protocol has been implemented for the Web-cache environment. Unfortunately neither error recovery nor flow/congestion control and packet segmentation have been realized.

As far as we know, the existing simulators do not include any instruments for evaluating new solutions for both static and dynamic request routing. In the following of this paper we will introduce a new extension to ns-2 which allows to easily implement and test new algorithms for request balancing among the CDN nodes.

4. A NEW NS-2 EXTENSION FOR CDN SIMULATION

Due to the importance of the request routing mechanisms and the implications that these issues have on the overall performance of a content distributed infrastructure, we consider fundamental the development of a tool which allows to easily evaluate new algorithms for load balancing. We propose a new library for extending the ns-2 with CDN simulation capabilities. In the following of this section we first introduce the main components of the new ns-2 module and their functionality. Then, we describe the integration of such module in the simulator framework. Finally, we discuss the implementation of a novel load balancing algorithm.

[1]CDNsim - `http://oswinds.csd.auth.gr/~cdnsim/`

[2]OverSim - `http://www.oversim.org/`

[3]OMNeT - `http://www.omnetpp.org/`

[4]HttpTools - `http://code.google.com/p/omnet-httptools/`

[5]NS2 - `http://www.isi.edu/nsnam/ns/`

4.1 Architecture Overview

The main novelties of our CDN architecture introduced in ns-2 are: (i) new HTTP client and HTTP server which act as CDN nodes, (ii) a new mechanism for introducing load balancing algorithms at the servers. The client and server can generate standard HTTP messages, like `GET` and `REDIRECT`. In our architecture the server also includes load balancing capabilities, and for this reason it can work as an entity in a peer-to-peer overlay network of CDN servers. The servers are inter-connected through the underlay network infrastructure. In our extension we neglected the problem of consistency of the data hosted by the servers. Therefore no mechanism for updating the contents among the servers' network has been provided; we rather supposed that all the servers are hosting a coherent copy of the data.

The client sends a `GET` message to the "closest" server, containing a request for a specific resource. Every content is identified by a number and it requires some server processing time, which translates into a delay time for serving the request. The client can behave in two ways: it may send either an infinite or a finite number of `GET` messages at regular intervals without waiting for any server reply, or it may send `GET` messages and then wait for the associated reply before sending a new request. In the first case the module allows to use a stochastic distribution for the sending interval; in the "waiting mode" the module adopts a proper queue for buffering the outgoing requests. The main novelty in the HTTP client is its capability to handle explicit `REDIRECT` messages. Indeed, based on a specific balancing criterium, a server can redirect the client's request to the proper server which is able to serve it. The client receiving such REDIRECT message can issue a further request towards the new server. In this simple description we can observe that the behavior of our HTTP client takes into account both the request arrival rate and the server's computation time. The module also provides utilities allowing the client to act not just as a single HTTP client, but rather as a "clients' cloud" with different requests' inter-arrival distributions.

The CDN server is in charge of both serving incoming requests from clients, and managing the content distribution with the other servers. From an implementation point of view, the server adopts a simple FIFO queue for requests buffering. When a new client request message is received, the server enqueues it, and it serves the first request in the queue. Any request needs a specific elaboration time; for implementing a realistic server scenario a configurable processing rate is provided for simulations. After elaborating a request, the server replies to the client with the `CONTENT` message. The implemented server module also acts as a member of a peer-to-peer CDN network. This requires that every server maintains a list of its partners. Such list contains information about the neighbors which can be contacted for the implementation of a distributed load balancing algorithm. For example, in case of dynamic balancing mechanisms it could contain servers queue lengths. According with the "state-aware" mechanisms, such information should be updated periodically. For this reason we provide the server with a proper protocol for the exchanging of server status information. Figure 1 depicts the sequence diagram of a dynamic load balancing algorithm: it shows the service status exchange, as well as the redirection procedure.

In summary, several messages are exchanged between client and server, which have been implemented in the proposed ns-2 extension for CDNs. As stated above, together with the well-known client-server messages GET, CONTENT, and REDIRECT, we introduce the `CDN` message which contains information about the server load status and which can be considered as a control message in the peer-to-peer network.

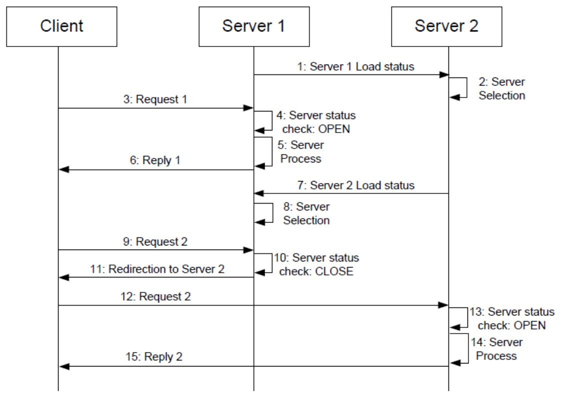


Figure 1: Load Balancing Sequence Diagram

4.2 Integration in the ns-2 framework

Ns-2 is an object-oriented, discrete, event-driven network simulator written in C++ and OTcl[6]. Primarily used for simulating local and wide area networks, ns-2 includes different kinds of event schedulers, network components object libraries, network setup module libraries. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++. Through an OTcl linking it is possible to control C++ objects by means of an OTcl script. Anyway ns-2 also allows to develop simulation objects entirely in the OTcl language.

Our CDN extension has completely been developed in C++, and we have provided linking functions for controlling C++ objects through OTcl script. To integrate our extension in the ns-2 framework, we introduced several C++ classes (Figure 2): (i) a *CdnData* class for data messages, (ii) a *CdnAgent* class for sending and receiving messages at nodes, (iii) a *CdnClient* application class for generating requests and for processing redirect messages, (iv) a *CdnServer* application class for processing incoming requests, for executing load balancing algorithms and for sending redirect messages to clients, (v) auxiliary classes to manage clients requests queue at the server and a couple of lists to manage peers directories and request indicators. Moreover, a new packet header has been added to the original code, so to properly identify the incoming packet application source type.

4.2.1 CdnData class

The *CdnData* class is the application level data exchanged both between the client and the server and among the servers. It is obtained as an *AppData* subclass and is made of an application type descriptor (inherited from *AppData* and set to `HTTP`), a sender identifier, a page requested identifier, a server identifier for redirection, the HTTP message methods, and a server status information.

[6]OTcl - `http://sourceforge.net/projects/otcl-tclcl/`

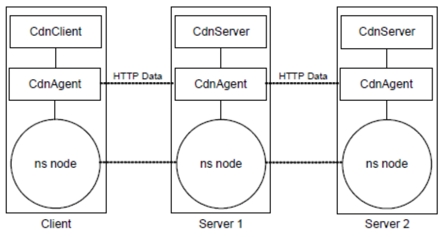


Figure 2: The CDN Framework in *ns-2*

4.2.2 CdnAgent class

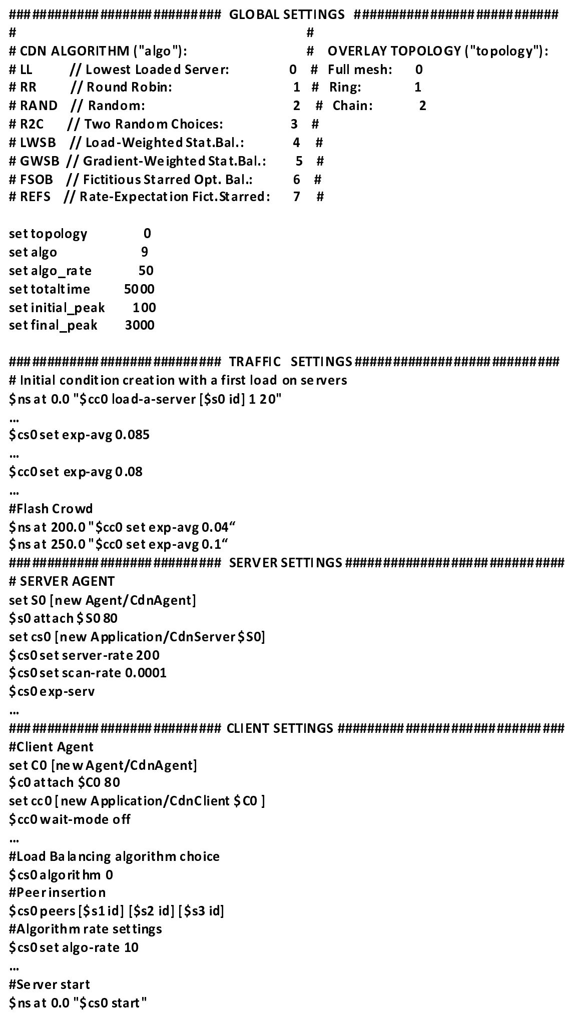
A fully ns-2 compliant extension requires the development of an agent module capable to send application level data, receive such data from the upper layer and pass them to the right application. For this reason we have implemented a new agent class, the *CdnAgent*, which can be "attached" through OTcl commands to an existing node. This class inherits from the *Agent* class. For this reason it includes a pointer for the reference to an application class object, which is fundamental for the implementation of class methods. Indeed, based on the specific application exploiting the agent, the `sendto` method properly creates the packet to send, while the `recv` method can call the right application data processing method. The *CdnAgent* module is also capable to send packets to different destinations at simulation time, differently from the standard ns-2 agents, which require knowledge of the destination at configuration time. This feature is fundamental for message redirections, because every client has to generate new `GET` messages towards an unpredictable destination server.

4.2.3 CdnClient class

As any other ns-2 application, the *CdnClient* class, which implements the client of the CDN architecture, has to be attached to an agent. For this reason it must include an agent pointer, initialized in the configuration script to the *CdnAgent*. The client sends `GET` requests to the servers and properly handles `REDIRECT` messages. The server to contact and the page to require are set at configuration time with specific commands. A client can send either an infinite or a finite number of `GET` messages at either regular or random intervals. The most important *CdnClient* method is called `processData` and handles both `CONTENT` messages and `REDIRECT` messages.

4.2.4 CdnServer class

The *CdnServer* class has the same properties of the *CdnClient*, but is more extended in functionality. Firstly, its `processData` method is in charge of also buffering client requests in a local queue. Furthermore, the *CdnServer* also has to handle `CDN` messages from other peers in the CDN. For this reason it is provided with a specific method called `processCdnData`. Finally, the server includes a specific function, called `cdnProcess`, which implements the load balancing algorithm exploited in the network. Every server has two timers: the former is used to periodically control the requests queue in order to pop the first request to serve; the latter is used to manage the periodical status information exchange. Each server has several status variables and flags. For example, the `cdnp2p` flag is useful for activating the distributed balancing mechanisms. The *CdnServer* also includes a `cdnPeerList` containing the CDN peers. Finally, several methods have been implemented to enable performance analysis of the server.



```
############################## GLOBAL SETTINGS ##############################
#                                                    #
# CDN ALGORITHM ("algo"):                            #   OVERLAY TOPOLOGY ("topology"):
# LL       // Lowest Loaded Server:              0   #  Full mesh:    0
# RR       // Round Robin:                       1   #  Ring:         1
# RAND   // Random:                              2   #  Chain:        2
# R2C     // Two Random Choices:                 3   #
# LWSB   // Load-Weighted Stat.Bal.:             4   #
# GWSB  // Gradient-Weighted Stat.Bal.:          5   #
# FSOB   // Fictitious Starred Opt. Bal.:        6   #
# REFS   // Rate-Expectation Fict.Starred:       7   #

set topology         0
set algo             9
set algo_rate       50
set totaltime     5000
set initial_peak   100
set final_peak    3000

############################## TRAFFIC   SETTINGS ##############################
# Initial condition creation with a first load on servers
$ns at 0.0 "$cc0 load-a-server [$s0 id] 1 20"
...
$cs0 set exp-avg 0.085
...
$cc0 set exp-avg 0.08
...
#Flash Crowd
$ns at 200.0 "$cc0 set exp-avg 0.04“
$ns at 250.0 "$cc0 set exp-avg 0.1“
############################## SERVER SETTINGS ##############################
# SERVER AGENT
set S0 [new Agent/CdnAgent]
$s0 attach $S0 80
set cs0 [new Application/CdnServer $S0]
$cs0 set server-rate 200
$cs0 set scan-rate 0.0001
$cs0 exp-serv
...
############################## CLIENT SETTINGS ##############################
#Client Agent
set C0 [new Agent/CdnAgent]
$c0 attach $C0 80
set cc0 [new Application/CdnClient $C0 ]
$cc0 wait-mode off
...
#Load Balancing algorithm choice
$cs0 algorithm 0
#Peer insertion
$cs0 peers [$s1 id] [$s2 id] [$s3 id]
#Algorithm rate settings
$cs0 set algo-rate 10
...
#Server start
$ns at 0.0 "$cs0 start"
```

Figure 3: Menu Script

4.2.5 Simulation Commands

In order to setup and control the simulation environment we extended the standard ns-2 OTcl commands with new ones specifically conceived for a CDN scenario. Figure 3 reports examples of OTcl scripts. In particular, it shows the creation of a new client and a new server node, together with their settings.

4.3 Queue and Traffic Characterization

In order to evaluate the performance of the different techniques, we implemented the ns-2 modules in such a way that we can set the client request rate and the server service time according to a stochastic distribution. Every server has been implemented based on a standard queueing model; in particular we adopted D/D/1 and M/M/1 models for deterministic and exponential distribution processes, respectively. Furthermore, we associate each server with an unlimited length for the buffer. Another important feature of our simulator is the possibility to set up an initial number of pending requests for each server. Furthermore, it is possible to change the request rate with a specific command during the simulation, in order to reproduce unusual traffic behaviors like the well-known flash-crowd phenomenon (Figure 3).

4.4 Implementation of new Balancing Algorithms

The main novelty of our solution is the possibility to easily extend the CDN infrastructure with new algorithms for request balancing. In order to guarantee such capability we have organized the server code in such a way as to isolate the main mechanisms for the balancing process. The server implementation, indeed, provides the following methods: (i) the `send_cdn_data()` which extracts server status information and sends such data to the server's neighbors, (ii) the `process_cdn_data()` method which collects and elaborates the status information coming from neighbors, (iii) the `choose_server()` method which selects, for each incoming request, the most suitable server for the redirection process. Each method includes a *switch structure* which, based on the algorithm specified in the OTcl script, allows to select the right balancing mechanism. This guarantees that new algorithms can be easily added to the framework by including a new *case* in the *switch structure*.

4.5 Metrics for Performance Evaluation

In order to compare different algorithms, we have included in the client and server implementation some variables for performance evaluation. The main client parameters are: the *Mean Response Time* (i.e. the *RTT*, Round Trip Time), which includes the propagation time on the network links, the intermediate routers queue waiting time, the servers queue waiting time and eventually the service time; the *Response Time Standard Deviation*, which provides information about both the worst and the best latency situation.

For server analysis we provide instruments for periodical *Queue Length* monitoring. Furthermore the server "balancing attitude" of an algorithm is evaluated through a parameter called *Unbalancing Index*, which is obtained by computing firstly, for every sampled time, the variance of the values of the queue length on the total number of the servers and then averaging it on the number of samples. Another important parameter is the *Multiple Redirection Overhead*, which is the ratio of the total number of redirected `GET` messages to the total number of `GET` messages generated by the clients. The above measurements are realized by an ad-hoc OTcl procedure.

5. A PERFORMANCE EVALUATION EXAMPLE

In this section we want to provide a simple example of how to perform a comparison of several load balancing algorithms with our ns-2 extension. We have implemented three mechanisms: Random (Rand), Least Loaded (LL), and Two Random Choices (2RC).

We have modeled each server as an M/M/1 queue with service rate μ. The frequency of requests generated by the clients is a Poisson process with rate λ.

In Figure 4 we show preliminary simulation results related to the LL algorithm in a simple 3-node ring topology. Every server is set with a different initial load and the same request rate. According with the expected results, we observe the balancing of the queue length over time.

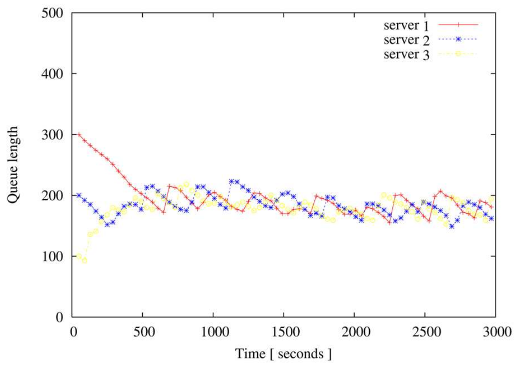


Figure 4: Queues length in a 3 node ring topology

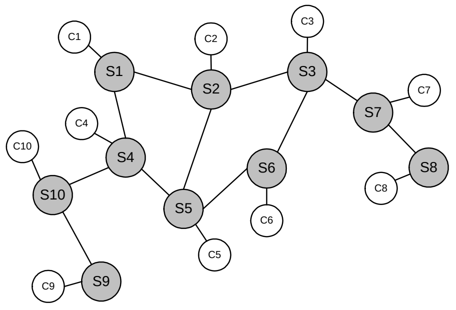


Figure 5: Simulation Topology

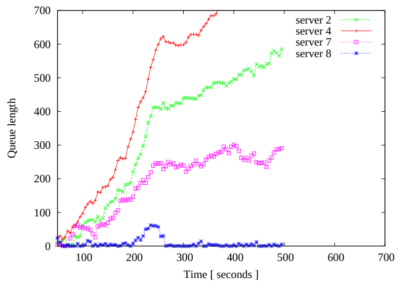
Table 1: Servers Parameters

	1	2	3	4	5	6	7	8	9	10
Q.L.	20	10	5	25	20	10	5	25	10	20
λ_i [req/s]	11	12	13	15	7	9	8	5	6	10
μ_i [req/s]	10	7	5	6	12	8	9	13	15	11

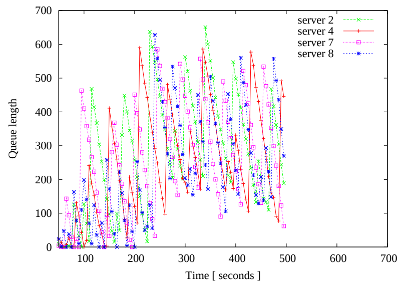
A more complex scenario (Figure 5) has also been considered in order to evaluate the soundness of our simulations. We have set different values for the request arrival rate, service rate, and initial load at each server, as reported in Table 1.

In order to consider a more realistic scenario we have also simulated a flash-crowd event by changing the arrival rate at server 7 to $\lambda_7 = 100\ reqs/s$ in the interval between $t_0 = 200s$ and $t_1 = 250s$.

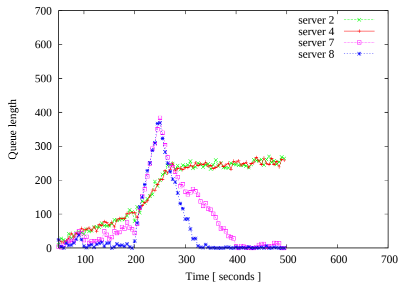
In Figure 6 we show the queue behaviors over time for Rand, LL, and 2RC algorithms, respectively (for readability reasons we have reported only four nodes). These graphs match the theoretical expectation for the queues length and they represent a quick way to compare the effectiveness of the balancing mechanisms. For the Random solution (6-a), due to the absence of a balancing mechanism, the queues at servers 2 and 4 diverge since their request rates are bigger than service rates. As expected, server 8 is unloaded since the incoming request rate is lower than the service rate. For server 7 the queue length has an intermediate behavior



(a) Rand



(b) LL



(c) 2RC

Figure 6: Queue Behaviors

Table 2: Unbalancing Index

	Normal	Flash-crowd
Random	344.10	337.06
Least Loaded	123.00	126.37
2 Random Choice	34.37	60.74

Table 3: Performance Evaluation with Flash-crowd

	$Avg_{RT}[s]$	$\sigma_{RT}[s]$
Rand	68.03	207.29
LL	23.94	14.82
2RC	12.97	8.16

since arrival and service rates are comparable. For the LL algorithm (6-b) the queue length follows the attended oscillations due to the time interval required for updating the status of the servers' load. For the 2RC algorithm (6-c) we observe that the queue lengths at servers 2 and 4 have a similar trend, due to the comparable traffic rates; server 7 is subject to a traffic peak, which propagates to server 8, which is close to it; the behavior of such servers is quite similar due to their isolation in a stub of the topology. In this case the flash crowd effects are quite visible on the involved elements.

On the other hand, from tables 2 and 3 we observe that the Random algorithm has no attitude in load balancing (see the unbalancing index), while the Least Loaded algorithm produces a better balancing, though with both a great excursion in the instantaneous queue length and a huge *Response Time*. The 2RC algorithm shows the best performance globally with respect to both servers queue stability and clients' RTTs.

6. CONCLUSIONS

In this paper we proposed a novel ns-2 extension for the evaluation of load balancing algorithms in content delivery networks. We developed and implemented new C++ components representing the client, the server and the transmitted data. New OTcl commands for controlling client and server have also been provided. Finally, procedures for the evaluation of the performance of the load balancing mechanisms have been included in our software.

In the future we are going to exploit such instrument as a support means for the definition of original solutions for optimized request balancing.

This ns-2 extension also represents a valid starting point for the implementation of new instruments for optimization problems in distributed systems. In particular we are confident that the capability of our library to work in non-hierarchical environments assures the possibility to design new simulation tools for the analysis of peer-to-peer overlay routing mechanisms.

7. ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 225553 (INSPIRE Project).

8. REFERENCES

[1] A. Barbir, B. Cain, and R. Nair. Rfc 3568 - known content network (cn) request-routing mechanisms. Internet draft, IETF, July 2003.

[2] T. Brisco. Rfc 1794 - dns support for load balancing. Internet draft, IETF, April 1995.

[3] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):263–311, June 2002.

[4] R. L. Carter and M. E. Crovella. Server selection using dynamic path characterization in wide-area networks. In *Proceedings of IEEE INFOCOM '97*, volume 3, pages 1014–1021, April 1997.

[5] M. Colajanni, P. S. Yu, and D. M. Dias. Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(6):585–600, June 1998.

[6] M. Dahlin. Interpreting stale load information. *IEEE Transactions on Parallel and Distributed Systems*, 11(10):1033–1047, October 2000.

[7] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available web server. In *Proceedings of IEEE Computer Conference*, pages 85–92, Febraury 1996.

[8] M. D. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, October 2001.