Resource Discovery and Request-Redirection for Dynamic Load Sharing in Multi-Provider Peering Content Delivery Networks

Mukaddim Pathan and Rajkumar Buyya

Grid Computing and **D**istributed **S**ystems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Parkville, VIC 3010, Australia
{apathan, raj}@csse.unimelb.edu.au

Abstract: A constellation of Content Delivery Networks (CDNs), termed as *peering CDNs*, endeavors to guarantee adequate delivery performance when the incoming request load is overwhelming for a single provider alone. Each user is served by an optimal Web server in terms of network cost, even under heavy load conditions. Before it could be comprehended, appropriate resource discovery and request-redirection mechanisms, coupled with an optimal server selection strategy, should be in place to perform the distribution of highly skewed loads. In this paper, we devise an effective load distribution strategy by adopting distributed resource discovery and dynamic request-redirection mechanisms, taking traffic load and network proximity into account. The load distribution strategy reacts to overload conditions, at a time instance, in any primary CDN server(s) and instantly distributes loads to the target servers, minimizing network cost and observing practical constraints. In this context, we exercise an asynchronous resource discovery protocol, reminiscent of the public/subscribe notion, and formulate the resulting redirection scheme. Extensive simulation analyses demonstrate the novelty of our approach. In particular, we show that our approach is effective to handle high load skews by preserving locality, and thus achieve service "responsiveness". We also perform a sensitivity analysis to reveal that our redirection scheme outperforms other alternatives to handle peak loads.

1. Introduction

Content Delivery Networks (CDNs) [6][19] evolved as a solution of Internet service degradations such as congestions and bottlenecks due to the large end-user demands posed on Web access services. To operate effectively, often a CDN is required to break down system silos to increase utilization rates, either through over-provisioning its capacity or harnessing external resources on demand. The requirements for providing high quality service through global coverage can be fulfilled through a constellation of CDNs, termed as 'peering CDNs' [20]. Such collaboration, leveraging existing infrastructures, is not only important from a reachability perspective but also from quality and performance perspective. Peering between CDNs can be observed for a short or long-term duration to handle workload variations.

The success of peering and the effectiveness of operations for content delivery in a peering CDNs system depend on its ability to perform *resource discovery*, *server selection* and *dynamic request-redirection* under degenerated load conditions (e.g. flash crowds). The resource discovery process specifies how external resources offered by disparate CDNs are discovered. An effective server selection strategy determines the optimally underloaded edge server(s) that is best suited to serve user requests. The server selection phase typically chooses the "nearest" optimal server to the requesting user. A dynamic request-redirection mechanism assists in directing user requests to the target edge server(s), so as to alleviate imbalanced load situations. These phases may be interleaved to collectively perform load distribution by reacting to overload conditions in a multi-provider peering CDNs system and thus endeavor to achieve scalability.

Many previous research [7][8][10][11][16][17][25][26][30] have focused on devising resource discovery and redirection algorithms for distributed Web servers, overlay networks, Internet, large-scale Grids, and Peer-to-Peer (P2P)-based systems. However, they can not be directly applied for load balancing in peering CDNs, due to the necessity for handling dynamic circumstances, thus requiring up-to-date information about widely-distributed resources. In addition, providers should learn about available resources quickly, without using an inordinate amount of communication, and the resource discovery and redirection algorithms may be used repeatedly to obtain updated resource status information. There are also other challenges, which include virtualization of multiple providers and offloading requests from the overloaded provider to its underloaded peers, based on cost, performance and load. In such a cooperative multi-provider environment, requests are directed to sets of servers deployed across multiple CDNs as opposed to individual servers belonging to a single entity. Therefore, resource discovery and request-redirections must occur over distributed sets of servers spanning multiple CDNs, without having complete state information.

In this paper, we present *distributed* resource discovery and *dynamic* request-redirection algorithms for an effective load distribution strategy. Our aims are: i) to perform dynamic load distribution under traffic surges by redirecting excess requests to optimally underloaded Web server(s), thus binding users to optimal replicas (*timeliness*); (ii) to exhibit acceptable throughput under overload conditions (e.g. during *flash crowds*); iii) to scale to distributed inter-CDN resources scattered across the globe (*dynamic lookup*); and iv) to maintain administrative control over local resources and their states (*resource encapsulation*).

Specifically, the communication protocol to aid resource discovery conservatively implements the public/subscribe paradigm. The use of the public/subscribe notion endeavors to perceive—scalability and full decoupling from other system operations; a possibly "offline" approach due to the asynchronous nature of resource discovery; and indirect addressing for load balancing. At the heart of the load distribution strategy lies the redirection scheme that takes traffic

load and network proximity into account. In our approach, load indices of distributed inter-CDN servers are obtained through an *asynchronous feedback mechanism* and network proximity is measured using a *pinger logic* with low messaging overhead. A simulation model, capturing key system components, is developed to evaluate the performance of our approach. Experiment results reveal that an acceptable level of throughput can be achieved, even under heavy load, and the proposed redirection scheme outperforms other alternatives. The main contributions of this paper are:

- An asynchronous resource discovery algorithm without any central coordinating authority to identify resources from disparate CDNs.
- A load and proximity-aware request-redirection algorithm that reacts to overloaded server conditions in multi-provider peering CDNs by steering excess user requests to optimally underloaded servers.
- A comparison-based simulation analysis to evaluate the performance and perceived benefits of our approach, and a sensitivity analysis of the proposed redirection scheme using critical system parameters.

The rest of the paper is structured as follows. In Section 2, a brief description of the peering CDNs is provided. It is followed by the proposed resource discovery and request-redirection algorithms. Simulation methodology is described in Section 4 and results are presented in Section 5. A comparative analysis of our approach to the existing work is followed next. Finally, Section 7 concludes the paper.

2. Peering CDNs: Overview

A peering arrangement is a conceptual layer, i.e. overlay, over the physical CDN networks, which play the main role by establishing agreements to share peers' resources, and by cooperating to create a rich computational environment for effective content delivery. The initiator of a peering negotiation is called a *primary* CDN; while other CDNs who agree to provide their resources are called *peering* CDNs or *peers*. Resources belonging to a peering arrangement of CDNs are scattered over the globe, with the primary CDN having the authoritative right.

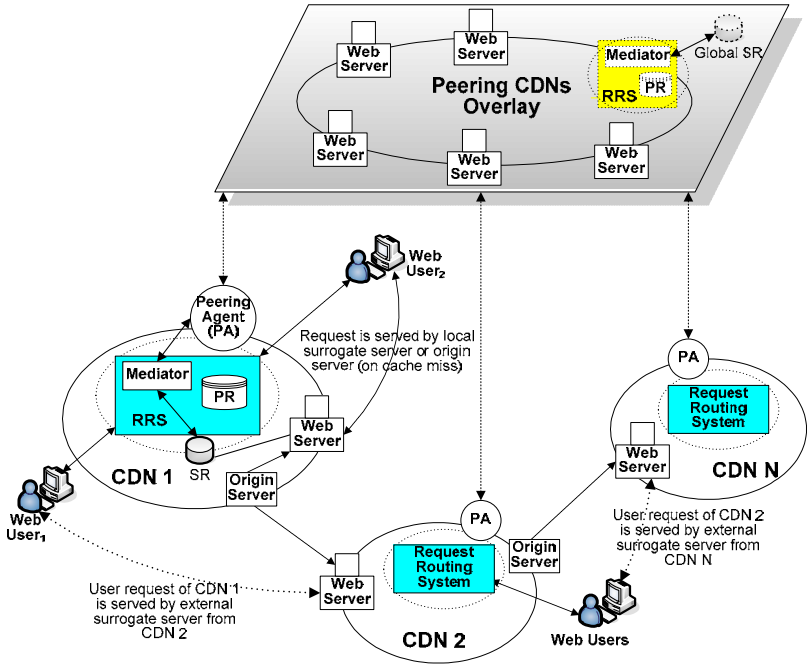

Figure 1: Abstract view of peering CDNs.

Figure 1 presents an abstract view of the peering CDNs [20], where a provider serves requests as long as it can handle the load internally. If load exceeds its capacity, the excess requests are offloaded to its peers. From the figure, we see that a given peering arrangement consists of Web servers from disparate CDNs at different geographical locations across the Internet. Each CDN has its own user request stream and a set of Web servers, but delegates only a subset of them, i.e. virtual CDN or subCDN, to take part in the peering arrangement. In the peering CDNs overlay, *Peering Agent (PA)* performs external resource discovery; *Mediator* performs policy-driven authoritative operations on behalf of the primary CDN; *Policy Repository (PR)* virtualizes all policies within the peering arrangement, and *Service Repository (SR)* encapsulates the status of CDN servers. The PA, Mediator, SR and PR collectively act as a "conduit" for a given primary CDN, and assist in external resource discovery. User requests for content are made to the Request Routing System (RRS) of the primary. These requests are then forwarded either directly to its server(s), or to a peer.

An overload condition occurs when incoming load to the primary CDN exceeds a given *alarm threshold*, as reported by its server(s).When this occurs, the server sends an alarm signal to the primary's mediator. Upon receiving it, the primary takes measure to redirect the excess requests to other optimally underloaded Web server(s) of peers. The

primary CDN directly manages the resources it has acquired, insofar that it determines what content is served and what proportion of the incoming traffic is redirected. At any time, a given CDN may be either in the role of a primary or a peer, i.e. the roles are fluid. From Figure 1, we observe that some user requests of CDN 1 are served by its local servers or the origin server (on cache miss), whereas others are being served by the external servers of a peer, CDN 2. It is important to note that depending on the load any CDN can act as a primary in a peering relationship. For instance, CDN 1 is a primary when its users are served by the peers' servers. Again, in the same peering relationship, CDN 2 acts as a primary CDN when its users are served by external Web servers from CDN N.

2.1. Resource Discovery

The candidate resource discovery algorithm for peering CDNs realizes a distributed nature, since there is no central control in the system. CDNs operate independently of each other; making local queries within its domain and transferring global information about part or all to the peering CDNs system. An SR instance, local to each CDN domain, contains local resource information. Once a peer delegates its resources for use by the primary, gateways of the participating CDNs interact to register the offered resources to the global SR, which is a distributed SR implementation (Section 4.1.2), containing information of all resources in the peering arrangement. The list of registered resources in the global SR is updated periodically. Thus, the membership management for resource discovery realizes a soft-state protocol. Contact address of the gateways can be learned through out-of-band information. The communication protocol is reminiscent of the public/subscribe paradigm. However, it differs from the public/subscribe system in terms of its functionality. While the latter deals with resource publishing and message dissemination; our main focus is on efficient resource discovery for peering CDNs. In addition, a publish/subscribe system is intended to find all potential participants who are capable of serving user requests. In contrast, the redirection strategy in peering CDNs, followed by the resource discovery, attempts to find the "optimal" participant, not all of them. As a consequence, a smaller fraction of the whole system is traversed and communicated.

2.2. Server Selection and Request-Redirection

An effective request-redirection mechanism to perform load distribution in peering CDNs could be devised by examining two alternatives. In the first case, all peers belonging to a "virtual CDN" or "subCDN" could provide a real-time load status of each surrogate in the global subCDN using standard metrics. The primary CDN (or an authoritative entity belonging to it) can compare these load indices with its metrics and choose whether requests have to be redirected to a peer. Practical constraints could be put in place to ensure that redirection cost is minimized and peers' servers are not overloaded. Alternatively, an independent third party could supervise and manage all the CDN peers for load distribution. However, security, trust, and proprietary issues make it unlikely that a CDN would agree to have an external party to make allocations of its resources according to some load distribution policy. Therefore, we follow the first approach by interleaving server selection and request-redirection to perform load distribution in peering CDNs. Our approach seeks to prevent wide oscillation in the load distribution decisions by selecting optimal server(s) through cost minimization, taking traffic load and network proximity into account.

3. Algorithms

In this section, we describe the workings of the resource discovery protocol in peering CDNs. We also provide the description of the load and proximity-aware request-redirection strategy, coupled with optimal server selection, in order to perform dynamic load distribution in peering CDNs.

3.1. Resource Discovery Formulation

Let M be the set of all possible resources from participating CDNs, with N users spreading across the system. Content request from user i is denoted as $r_i \in R$, which is a constraint on the set of available resources. Let $match(r_i) \in M$ be the set of Web servers that satisfy r_i. A primary CDN A, which receives the incoming request r_i is expected to provide the set of resources $match(r_i)$. However, under peak load, CDN A may not be able to serve r_i, and therefore, searches for other peers to serve the request on its behalf. Let us consider a peering CDN B, which can provide resources $match(r_i')$, satisfying user request r_i'. We say that r_i and r_i' overlap iff $match(r_i) \cap match(r_i') \neq \varnothing$ and there exists at least one resource that satisfies both r_i and r_i'. Therefore, optimally underloaded servers from CDN B can be utilized by CDN A to satisfy the content request.

3.1.1. Distributed Resource Discovery Algorithm

Figure 2 and Figure 3 respectively present the pseudocode for service request and service response procedures, which are required during resource discovery. The principle of resource discovery in peering CDNs is as follows. Upon receiving user requests for content, the primary CDN finds suitable Web servers to serve them (Figure 2: Lines 1 to 3). In order to preserve locality, the primary issues local queries to the local SR instance, to find the potential local resources that are able to serve the incoming requests. Under traffic surges, incoming load of a primary CDN exceeds a given *alarm threshold*. When this occurs, an *alarm_flag* is set to indicate that no optimal resource is found within the local domain. The primary uses a *lookupMethod* to contact peers and populate a list *WSList* of available resources from participating providers (Figure 2: Lines 4 to 8). The primary sends a *ServiceRequest* message to peer(s), containing the

required service requirements, and receives the *ServiceResponse* from the peers' servers (Figure 2: Lines 9 to 13). When a peer's server receives a *ServiceRequest* message, it chooses the requests that it accepts to serve and the ones that it refuses to serve, based on whether it can meet the service requirements. It then sends a *ServiceResponse* message containing the lists of acceptable and rejected requests (Figure 3: Lines 1 to 8). Upon receiving the response from the peer(s), excess requests are redirected from the primary CDN to the optimally underloaded peer(s), using the redirection mechanism outlined in the next section.

The *lookupMethod* during service request determines the way peer selection is performed. It depends on whether the peers share/advertise partial information about their services, i.e. topology, connectivity information, dynamic link properties (link weight, background traffic, and congestion level), and dynamic node properties (bandwidth and processing power that can be shared by peers' servers). The *lookupMethod* may follow *broadcast* or *selective* mode. The first mode of operation is the only choice when a CDN does not possess an SR instance, containing its local resources information. With the presence of an advertising mechanism and the existence of an SR instance at each local CDN domain, peer selection could be performed selectively. In our approach, we exploit dynamic status and availability information of offered resources from peers, by searching the global SR. We seek to preserve locality, since the attempt is to find local resources first, thus realizing *dynamic lookup* and increasing *resource encapsulation* within the primary CDN's domain.

Other complimentary selective modes could be: *controlled anycast*, *greedy random* and *greedy ordered*. Controlled anycast employs a route controller [28][29] to which the Web servers of the peering CDNs advertise the anycast address. As such, the route controller can influence the route selection to peers using some external intelligence [1]. In the greedy random mode, one peer is selected that can serve the content request on behalf of the primary, chosen uniformly at random. The greedy order mode chooses a peer according to some predefined criteria, such as network factors and QoS. We leave the full development of these mechanisms for future work.

```
      begin ...
 1:   for all r ∈ R do
 2:     if alarm_flag = false && match(r) ≠ ∅ then
 3:       Serve content request r from primary CDN's Web servers
 4:     else if alarm_flag = true || match(r) = ∅ then
 5:       if WSList = null then
 6:         Populate  n, WSList from p peers using lookupMethod
 7:       end if
 8:     end if
 9:     for i =1 to n do
10:       Send ServiceRequest(r) to i.WSList
11:       Receive ServiceResponse(AcceptedReqList, RejectedReqList) from i.WSList
12:     end for
13:   end for
      ...
      end
```

Figure 2: Service request from primary CDN during resource discovery.

```
      begin ...
 1:   for all s ∈ S do
 2:     if s can be serviced && s is not already accepted then
 3:       Add s to AcceptedReqList
 4:     else if s can not be serviced then
 5:       Add s to the RejectedReqList
 6:     end if
 7:   end for
 8:   Send ServiceResponse(AcceptedReqList, RejectedReqList) to the primary CDN
      ...
      end
```

Figure 3: Service response from a peer's server during resource discovery.

Table 1 summarizes the properties of the resource discovery algorithm. This annotation avails to analyze our approach and assists in making the right system design choice. The first property specifies the basic notion of resource discovery and the domain within which it is initiated. The next property focuses on the implementation, specifying the granularity—task unit that the system can support; distribution—particular design choice; data integration scheme—how to gain access to necessary data of interest; and peer selection—how locality is preserved for effective resource discovery. The last property delineates the perceived scalability of our approach by stating the primary means for supporting resource discovery, instrumentation of user access pattern, and identification of available resources.

Table 1. Annotation of the resource discovery algorithm.

Properties	Parameters	Description
Methodology	Notion	Fully decoupled "offline" approach (*asynchronous*)
	Interconnection topology	Peering CDNs (*system under consideration*)
Implementation	Granularity	End-user requested content
	Distribution	Distributed query-based approach
	Data integration scheme	Gateway (*information exchange and request-redirection*)
	Peer selection	Resource encapsulation and dynamic lookup (*searching within local domain first*)
Scalability	Data volume or index representativeness	Scalable content-based searching upon user requests
	Traffic intensity or user base	Incoming traffic for replicated content
	Resource identification	Request mapping (*accepted and rejected requests lists*)

3.2. Request-Redirection Formulation

Since request-redirection is critical to our load distribution strategy, we first formulate the redirection problem and then present the devised algorithm. We define a metric, redirection cost R_c, for serving requests through redirection in overload conditions. R_C depends on a server's traffic load and network proximity (in terms of round-trip response time). The cost of serving requests varies with different servers of the peers. Specifically, R_c is defined as:

$$R_c(i,j) = \begin{cases} \infty & \text{if } p_{ij} = \infty \\ l_{ij} p_{ij} & \text{otherwise} \end{cases}$$

where l_{ij} is the incoming traffic load from user i on server j, and p_{ij} is the network proximity between them. It is known that a server's response load indicates the potential load assigned to it, since request and response loads on a CDN server are linearly correlated [1]. Therefore, the request load l_{ij} (traffic volume) can be calculated based on the server load. A Web server j's load is expressed as the product $u_j S_j$, where u_j is the server utilization in [0, 1] as reported by the load monitoring apparatus (mediator and SR) and S_j is its capacity, specified by the maximum number of serviced requests/second as reported in the CDN server's configuration specifications.

The delay caused by inter-CDN redirection is denoted as p_{ij} and it is defined as in the following:

$$p_{ij} = \begin{cases} rt_{ij} & \text{if } j \text{ is an intra-CDN Web server} \\ rt_{ij} + I_D & \text{if } j \text{ is an inter-CDN Web server} \end{cases}$$

where rt_{ij} is the response time from user i to server j and I_D is the delay.

In order to minimize redirection cost during load distribution, the redirection problem is formulated as follows:

$$\text{minimize} \quad \sum_{i=1}^{N} \sum_{j=1}^{M} R_c(i,j) a_{ij}$$

$$\text{subject to} \quad \sum_{i=1}^{N} a_{ij} = 1, \forall j$$

$$\sum l_{ij} a_{ij} \le S_j, \forall j$$

$$a_{ij} \in \{0,1\}, \forall i,j$$

where a_{ij} is an indicator variable to determine whether the Web server j is in the same CDN as user i; $a_{ij} = 1$ iff server j is an inter-CDN server, and $a_{ij} = 0$ otherwise.

Table 2. Significant properties of the request-redirection scheme.

Properties	Parameters	Description	
Activation	Activation trigger (*when*)	Asynchronous (*on CDN server request*)	
	Activation decision (*where*)	Distributed (*Gateway redirection upon requests from distributed servers)*	
Implementation	Status information	Traffic load (*correlated with server response load = utilization * capacity*)	Alarm (*Asynchronous feedback*)
Redirection policy	Server selection (*how*)	Minimize redirection cost from available server list (*mapping of overloaded and underloaded server lists*)	
	Redirected entities (*what*)	End-user requests	

Table 2 summarizes the properties of the redirection scheme, according to the classifications presented by Cardellini et al. [8]. The first property specifies the mechanism in which redirection is activated and where the activation decision process is made. The next property focuses on the implementation, specifying the status information used for

redirection. The last property delineates the redirection policy by stating the server selection strategy and the entities that are redirected. We describe these properties in more detail in later sections.

```
      begin …
 1:   for all r ∈ R do
 2:     Obtain the list of available servers n, WSList from peers    // Resource discovery
 3:     for i = 1 to n do
 4:       Populate o, OWSList and u, UWSlist    //Overloaded and underloaded server lists
 5:     end for
 6:     for j = 1 to u do
 7:       Calculate incoming traffic load based on loadmetric
 8:       Measure network proximity
 9:       Calculate redirection cost Rc
10:     end for
11:     do
12:       Select optimal server opWS minimizing Rc
13:       Add opWS to targetWSList
14:       Redirect request to opWS
15:     while alarm_flag = true
16:     if |targetWSList| > 1 then
17:       if targetWSList.avgLoad <= alarm_threshold/2 then
18:         Remove least loaded server in targetWSList
19:       end if
20:     end if
21:   end for
      ...
      end
```

Figure 4: Load distribution algorithm (*LD_minCost*).

3.2.1. Dynamic Load Distribution Algorithm

Figure 4 presents the pseudo-code for the proposed load distribution algorithm, named *LD_minCost*, which integrates the resource discovery, server selection and request-redirection phases. It does not attempt to perform load distribution when the Web servers of a primary CDN are working under an acceptable load. At a given time, if an overload condition exceeding an alarm threshold is reached, as reported by a primary CDN server, servers' status (response load) is assessed and a list of lightly loaded servers is generated from each participating CDNs (Lines 1 to 5), making use of the resource discovery algorithm stated in previous section. The traffic load and network proximity for each underloaded server are measured and the redirection cost, R_c is calculated in Lines 6 to 10. The optimal server from the underloaded server list is selected such that R_c is minimized upon redirection and the server does not get overloaded due to steered traffic. This optimal server *opWS* is added to a set of usable server list *targetWSList*, which is maintained by the primary CDN to satisfy its content requests. Thus, a mapping is maintained between the requests and a set of suitable servers to serve those requests. As long as there is an overloaded primary CDN server, a new server minimizing R_c (except the optimal server selected earlier) is added to this list (Lines 11 to 15). By using our load distribution strategy, we prevent a Web server from going into an overloaded state and multiple servers can serve a peak demand or a flash crowd situation. Moreover, if *targetWSList* contains several Web servers and their average load decreases significantly, one Web server is removed at a time from the list (Lines 17 to 21). It ensures that the degree of replication for serving requests does not remain unnecessarily high when requests relinquish over time. Moreover, it also guarantees that sufficient underloaded resources are always available in the peering CDNs system so as to utilize them during load distribution.

3.2.2. Perceived Benefits

A major advantage of our approach over traditional DNS-based redirection systems is that the actual end-user requests (eyeballs) are being redirected, opposing to the local DNS requests as in DNS-based redirection. Therefore, we can achieve a finer grain redirection. Since load distribution is performed dynamically, any redirection changes take effect instantly. In contrast, due to the IP-address caching in the intermediate name server, the DNS dispatcher loses direct control on subsequent requests for a Time-To-Live (TTL) period following address resolution, and thus causes some delay before redirection changes have an effect [10]. We also seek to achieve high locality with good load balancing, since requests targeted to the primary CDN stay within its domain as much as possible and are redirected to optimally underloaded peers only during peak load conditions. In this way, our solution does not produce widely oscillated outcomes due to load distribution through request-redirection, and thus we seek to achieve service "responsiveness". Finally, our approach endeavors to neutralize any load imbalance in the system, since participating CDNs have dynamic nature to act in primary or peering roles. The performance results of our approach and the simulation analysis in Section 5 support these claims.

3.3. Time Complexity

Let us consider a peering arrangement of one primary and P peers, with N users generating R requests in the system. Let n be the number of available resources, which are found during the lookup process. The time complexity for populating an available resources list from P peers is $O(P)$. Further, $O(n)$ is the worst case complexity for contacting n available resources by the primary to identify the servers which can serve the given content requests. Then the complexity for requesting service for one request from the available resources of the peers is $O(P+n)$. Therefore, the time complexity of the algorithm in Figure 2 is $O(PR+nR)$.

A peer's server receives S service requests from the primary. A worst case scenario gives $S = R$, thus producing a time complexity of $O(nR)$ for all the n available servers to be used by the primary. Hence, the resultant complexity of the service request and response during resource discovery is also $O(PR+nR)$.

Let us consider that load distribution is performed among u number of optimally underloaded servers, from the list of n available servers. Given an overloaded condition remains for a constant time T, the complexity of the load distribution algorithm *LD_minCost* is $O(nR+uR+T)$. By omitting the constant, the resultant complexity is given by $O(nR+uR)$. However, the resource discovery, server selection and request-redirection are integrated to perform load distribution in the peering arrangement. Therefore, the overall time complexity is $O(PR+nR+uR)$.

4. Methodology

Measurement based performance studies could be advantageous and suitable when a real system testbed or prototype is available. However, they may not reproduce the problems and scenarios for which the solutions are designed, since in real testbeds several important parameters, such as server and network load conditions, can not be controlled. In addition, it is extremely difficult to have a significant amount of geographically dispersed end-users simultaneously to generate traffic causing a flash crowd. In contrast, a simulation analysis could provide a detailed representation of key system parameters. Therefore, we have developed a simulator, based on Independent Replication Method, using the CSIM/Java[1] simulation toolkit. It assists to conduct *repeatable* and *controlled* experiments that would otherwise be difficult to perform in real CDN testbeds.

4.1. Simulation Environment and Parameters

In our simulation model, we provide an approximate representation of the environment, yet representative of the key system attributes. Our simulation model is based on a reference scenario [21], which consists of four CDNs with their sets of Web servers placed at different geographical locations across the Internet. Each CDN has a set of servers and a pool of users to generate own request stream. Users request content via their own browsers and make use of a proxy server according to the same client-side policy. In order to take part in peering, each CDN defines a virtual CDN or subCDN with a subset of its resources. To provide an accurate characterization of the scenario, we have simulated the main system entities: (i) Web servers, (ii) mediator, (iii) distributed SR, (iv) network congestions, and (v) end-users. In our simulations, PA and PR have limited functionality. Exploitation of their full functionalities [20] for Service Level Agreement (SLA)-based negotiation, policy classifications and policy enforcement are left as future work.

Table 3. Parameters of the system model.

Parameter	Value
SR update frequency	30*s*
Size of load index dissemination message	11*bytes*
Size of proximity measurement query	18*bytes*
Timeout period for proximity measurement	1000*ms*
UDP packet loss rate	0.3
Average network delay	100*ms*
Average inter-CDN delay	200*ms*
Alarm threshold	80% of a server's utilization
Traffic (end-user) distribution	$f(x) = \alpha k^{\alpha} x^{-\alpha-1}, \alpha, k > 0, x \geq k$

Table 3 reports the system parameters used in our analysis. The parameter values, indicative of the simulation model, are chosen as follows. Each server calculates its utilization and updates the SR every 30*s* by sending an 11*byes* load index dissemination message. An alarm signal is sent if this value exceeds 80% of the server's utilization, i.e. alarm threshold. We use Transmission Control Protocol (TCP) for disseminating reliable and valid load index and User Datagram Protocol (UDP) for network proximity measurement. The reason for using UDP in the proximity measurement is because there are devices which will prioritize ICMP traffic (in case of TCP) over other traffic, which if there is congestion along the way, could skew things a bit. In addition, some service setups such as firewalls and

[1] It creates process-oriented discrete-event simulation models. Please check: http://www.mesquite.com/.

routers limit ICMP traffic because of various denial of service threats. Since our goal is to measure proximity under "realistic" traffic, using something closer is deemed more significant. UDP also does not require acknowledgement of packets received, which causes less messaging overhead than TCP. Specifically, we used an 18*bytes* proximity measurement query with a timeout period of 1000*ms*. To capture the unreliable nature of UDP, we model 30% packet loss rate. Finally, in order to reflect the effects of network congestions in our model, we use 100*ms* average network delay and 200*ms* average inter-CDN delay.

4.1.1. CDN Web Servers

We have implemented the CDN servers as a set of facilities[2] that provide services to user requests. We have configured the servers according to the specifications from Fourth Quarter 2006 SPECweb2005 Results[3]. Detailed description of the server configuration, along with the used service distributions, their Probability Distribution Functions (PDFs) and associated properties could be found in [21].

The response load of a Web server (*LoadMetric*) is expressed as a product of its utilization in [0, 1] at a given time during simulation and the maximum number of served requests/second (capacity). We use an asynchronous feedback mechanism, which assists the Web servers to trivially measure their actual loads and periodically update them in the SR. If a server's load exceeds a given alarm threshold, it signals the mediator to perform load distribution. A normal signal is sent when the load returns below the threshold. The use of such an asynchronous feedback mechanism suffices to consider a server as a candidate for receiving requests only if that server has not declared itself critically loaded.

CDN servers (facilities) have no queuing delay, since they are configured to have high capacity and large bandwidth. This approach leads to the logical implication that servers in the simulation can handle any size of load. This assumption is necessary to deal with request arrivals with very large processing requirements [1]. The load distribution algorithm deployed in the mediator decides, upon receiving an alarm signal, whether a server is overloaded or not, depending on its load information from the global load table maintained by the distributed SR implementation.

4.1.2. Distributed Service Registry

There is a distributed implementation of the SR in our simulation, wherein each CDN has an SR instance to get the load status of its Web servers. Once a peer delegates a set of its servers to define the subCDN, the real-time state of each surrogate is passed to a global load table of the SR using standard load metrics (*LoadMetric*). The global load table stores the overall state of the servers in the peering arrangement using a tuple

(Web Server, LoadMetric)

Two key data structures are maintained to realize the distributed SR implementation. First, each SR instance in a peering CDN is responsible for keeping a local copy of the load status of its servers. This load information is not necessarily propagated straightaway to the peering CDNs system. Second, the primary CDN maintains the global load table to perceive the global load information of the peering CDNs system. Therefore, it must globally communicate load information amongst the peers so that intelligent load distribution decisions may be made. The asynchronous feedback by the Web servers is used for consistency of global load information. Such best-effort data consistency mechanism allows proceeding with the operations without querying the servers for updated load information. In this way, the servers are made to decide on the tradeoff between serving requests and updating load information.

Maintaining a data structure for the global load table in the primary CDN may seem infeasible with regard to security and fault tolerance. However, assuming a set of trustworthy participants, mendacious behavior from a provider, posing security threats, is not expected to be usual. This assumption could be justified by the fact that a given peering arrangement is likely to contain a handful of providers. The overhead of load update messages could be high if a large number of Web servers from the participating CDNs are present in the system. Nevertheless, at an instant it is unlikely to have more than a few hundred or thousand nodes in a peering arrangement from a small number of participants. The other mitigating factor is that load information is updated periodically and the only data that needs to be sent is a few bytes of load index. For example: a total number of 1000 servers in a given peering arrangement, 30*s* update frequency and 11*bytes* of data transfer for each load table entry "*WebServer_ID LoadMetric*" would lead to only about 22*KB/min*, or less than 0.5*KBps* for the messaging overhead due to the asynchronous feedback by the servers to the SR.

4.1.3. Mediator

Figure 5 illustrates the workings of the mediator, which is simulated to act as an authoritative entity in a given peering arrangement. It monitors the global load table to get Web servers' status and uses the pinger logic to get their network proximity information, in terms of round-trip response time (in milliseconds), assuming that they are directly correlated. The pinger logic uses the User Datagram Protocol to send an 18*bytes* proximity measurement query (as UDP packet) of the format "*PING Time CRLF*" to each Web server for the network proximity measurement. Time represents the timestamp when query is sent and *CRLF* represents the carriage return and line feed characters that terminate the query message. We also set a timeout period of 1000*ms* to check whether the servers are reachable. Thus, along with the proximity measurement, the pinger logic tests a Web server's ability to respond to a proximity measurement query, as well as its level of responsiveness under the current load.

---

[2] Each facility is a simulated resource with a single server and a queue for waiting requests.
[3] Standard Performance Evaluation Corporation. http://www.spec.org/

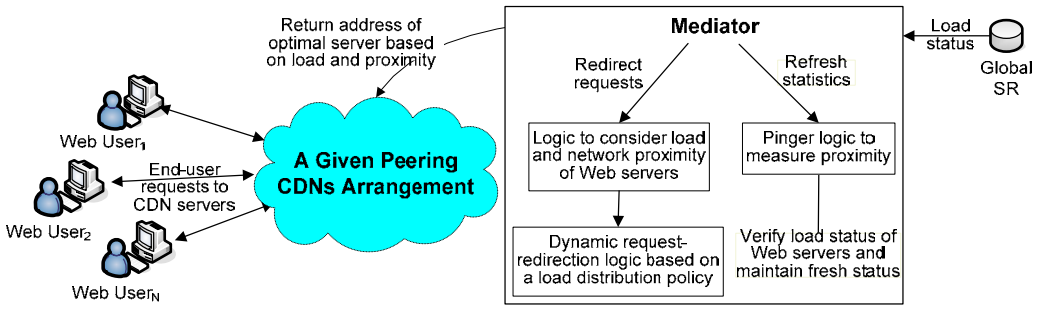

Figure 5: Operations of the mediator

The evaluation of network proximity among end-users and edge servers is a function of network topology and dynamic link characteristics. Therefore, the pinger logic is deemed to be located close to the users so that the mediator ideally can have the same view of the network status as the user's browser (Figure 6). This approach allows estimating, reasonably accurately and possibly offline, the network proximity between the user and CDN Web servers. It may appear that this approach leads to a load distribution system that does not scale enough as it requires an instance of pinger logic to be placed close to each of the numerous number of users in the Internet. However, in practice, it is observed that most of the end-users make use of a proxy server which filters Web accesses through caching. Content which are not available in the proxy server are retrieved from the origin server(s) and stored locally in a cache. Additional requests for the same content are served by the proxy until the expiration time after which the content is considered 'stale'. Therefore, the scalability problem can be solved by placing the pinger logic in the proxy server. This solution is feasible because the pinger logic is activated by the mediator for network proximity measurement only when the requested content is not in the proxy cache.

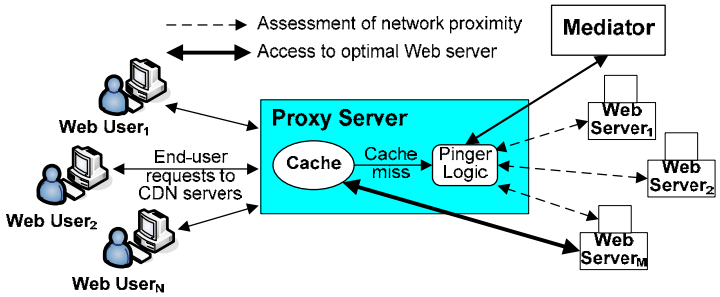

Fig. 6. Network proximity measurement

4.1.4. Network Congestions

In order to consider the impact of network congestions on load distribution strategies, we model two types of networks delays, namely, *inter-CDN* and *intra-CDN* delays. In addition, for network proximity measurement we have simulated the UDP packet loss (*LOSS_RATE = 0.3*) due to network congestions. Intra-CDN delays have three components: (i) minimum round-trip time between server and the user browser, (ii) queuing delays at the mediator[4], and (iii) packet transmission time for each link on the user browser and CDN Web server path. While (i) and (ii) are measured through simulation, (iii) is modeled as average network delay of 100*ms*. Inter-CDN delays are random variables that model communication latency between different geographical CDN domains. The Inter-CDN delays are 200*ms* in average. Within the simulation model, we do not characterize the delays spent for address resolution of Web servers.

4.1.5. End-Users Traffic

User requests are implemented as CSIM processes[5]. We assume that end-users request content via their own browsers to the CDN, according to the same client-side policy. The hidden load weight [10] is implicitly taken into account through the user distribution to CDNs, as requests to different CDNs are properly weighed and are distributed to the servers in a given peering arrangement.

Alike the Internet access workloads, user requests show self-similarity. A self-similar process has observable bursts in all time scales. It exhibits long-range dependence, where values at any instance are typically correlated with all future values. This self-similar nature can be described by using a heavy-tailed distribution [12][13]. Therefore, user requests to each CDN Web server follow a highly variable Pareto distribution with Probability Density Function (PDF),

$$f(x) = \alpha k^{\alpha} x^{-\alpha-1}, \alpha, k > 0, x \geq k$$

where the weight of the tail of the distribution is determined by $\alpha < 2$.

[4] The queuing delay at the mediator occurs for any possible congestion during load distribution in the peering CDNs system.
[5] CSIM processes are objects, based on Java threads, which make use of simulated resources.

Table 4. List of performance indices.

Performance Index	Description
Concurrent flows	Number of ongoing requests at each server. A desirable scheme should keep the number below the capacity limit of each server all the time.
Completions	Number of completed requests at each server. It is used to evaluate the performance of the proposed resource discovery mechanism.
Rejection rate	Number of dropped requests due to service unavailability. It is used to investigate the service disruptions in each scheme.
Utilization	A server's utilization in [0, 1] as reported by the load monitoring apparatus It is used to demonstrate the performance of the proposed redirection scheme.
Maximum utilization	Highest utilization at a given instant among all primary CDN servers. It is used to emphasize the impact of redirection on load distribution of primary CDN servers.
Cumulative frequency of maximum utilization	The probability that the maximum utilization of the primary CDN is below a certain value. It is a major performance criterion which determines whether the primary CDN is overloaded or not, by focusing on the highest utilization among all primary CDN servers.

4.2. Schemes and Metrics for Comparison

Traditional use of DNS scheduler for load balancing generally takes the Round-Robin (RR) algorithm to map requests to servers [10]. Therefore, we use an RR-based and a probabilistic version of this policy to assess the effectiveness and to evaluate the performance of our approach. We also use a simple deterministic scheme for comparison purposes. Specifically, we experiment with the follow policies: *LD_RR*, *LD_PRR*, and *LD_LL*. When incoming load exceeds the alarm threshold, the LD_RR policy uses a Round-Robin (RR) approach to redirect excess requests to all available underloaded servers in a cyclic order. The LD_PRR policy is a variant of the LD_RR policy. The basic idea is to make probabilistic round-robin type assignment to the servers. The probability is based on the residual capacity of servers in a given peering arrangement using the latest server load index. For this purpose, we generate a random number v ($0 \leq v \leq 1$) and, under the assumption that i–1 is the last chosen server, we assign the new requests to server i with *loadmetric* (utilization) u_i, only if $v \leq u_i$. Otherwise, we skip the server i and consider i+1 repeating the same process. The LD_LL policy uses a trivial approach that performs load distribution by redirecting excess requests to the least loaded server. Just for comparison purpose, we also consider no redirection which tries to assign requests to the closest server, without considering the load. Table 4 lists the performance indices that are used in the experimental evaluation.

5. Experimental Evaluation

In this section, simulation results are presented to evaluate the performance and to provide critical assessment of our approach. We run our experiments for the reference simulation model of Section 4, with one provider as primary (CDN 1) and others as peers. Results are obtained from ten simulation runs, where each run is for duration of 10000*s* (approx. 3 hours) of the peering CDNs system activities. While our simulations are designed to converge to the "true solution" of the model, running the experiments for a finite amount of time may not provide the exact true solution. However, choosing the right length of a simulation run is not obvious. Overly short simulation runs result in highly inaccurate performance statistics, whereas too long simulation runs unnecessarily waste computing resources and delay the completion of the simulation study. This problem can be address by estimating a confidence interval, which is a range of values in which the true answer is believed to lie with a high probability. Therefore, we have calculated confidence intervals in order to show the accuracy in the results of simulation output. In our case, confidence interval with 95% confidence level is estimated to be within 4% of the mean.

5.1. Traffic Load on Servers

Figure 7 shows the number of connections at each server in different schemes. By keeping track of the concurrent flows at each server, we examine the performance of load distribution. These results indicate whether a scheme can keep the number of active connections (requests) to each server below respective capacity limit. For the clarity of presentation, we plot samples after each simulation run, using two scales—Scale 1 and Scale 2—to show the concurrent flows at servers of primary and peering CDNs, respectively.

Server load is not taken into account in the no redirection policy and user requests are sent to the closest server. Hence, from Figure 7(a) we observe that the load at a few servers, in the primary CDN domain, grow significantly. For example: throughout the simulations, Server 1 and Server 2 receive more requests than their capacity. Unless they are provisioned with enough capacity to serve more concurrent connections, they will end up dropping many requests. In Figure 7(b), we see that LD_RR performs some load distribution by sending extra requests to the underloaded servers from primary and peering CDNs. However, it does not take cost (in terms of traffic load and proximity) into account and can potentially lead to high redirection cost.

Figure 7(c) presents the performance of LD_PRR. Since it assigns some probability to the underloaded servers based on their residual capacity, it redirects more requests to the server(s) with high probability. Therefore, it performs some load distribution but may cause sudden surge to a particular server and thus lead to imbalance load situations. For example: during simulation time 8 and 10 (x10000*s*), Server 1 receives many more requests than its capacity. LD_LL does not perform well as it fails to distribute loads to multiple servers. As for instance, from Figure 7(d), we observe that as simulation time passes, Server 1 receives more requests than its capacity, specifically, during simulation times 2, 4, 6, and 8 (x10000*s*), and Server 7 constantly receives extra requests than what it can handle.

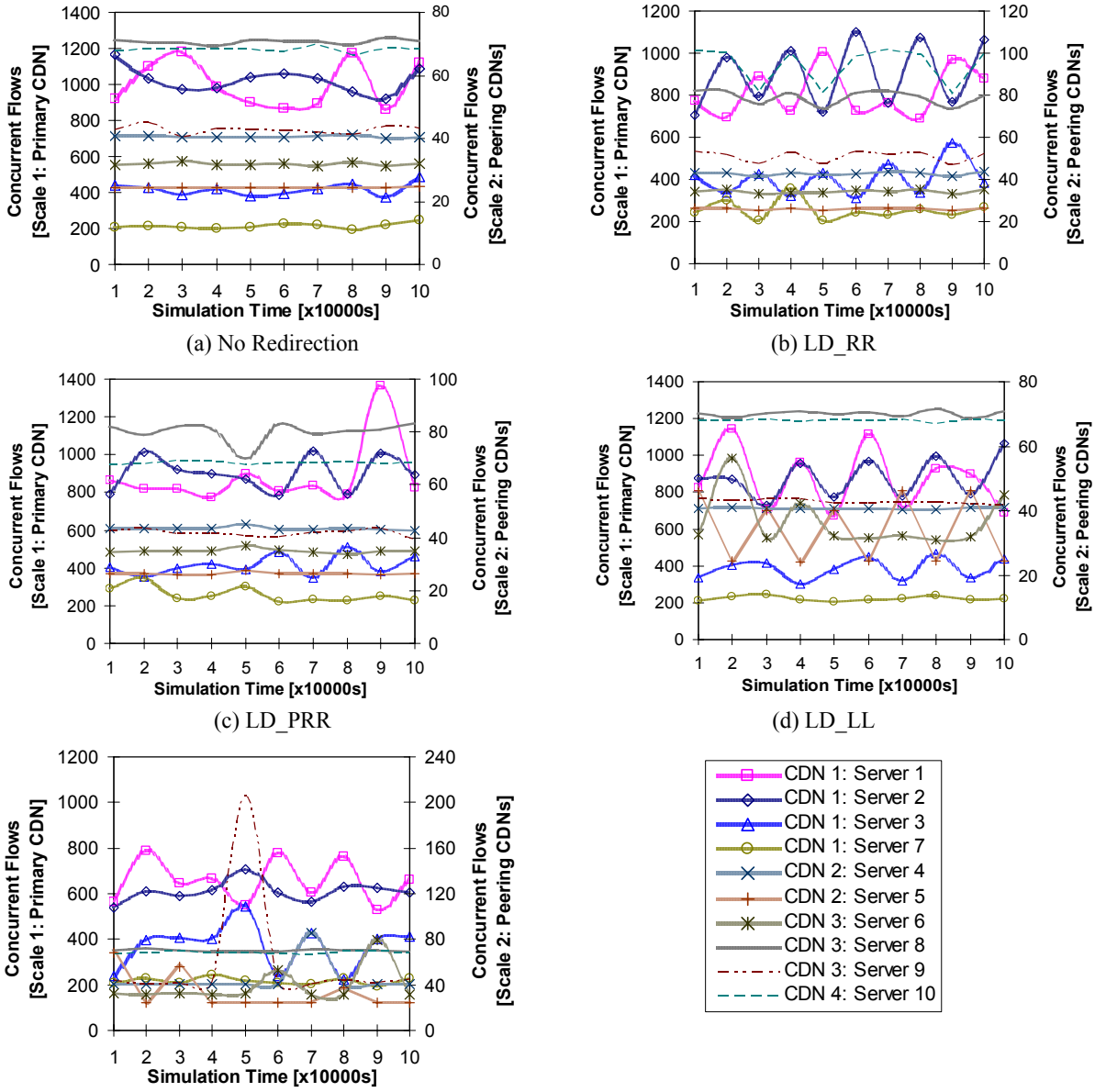

Figure 7: Number of concurrent requests for each scheme.

In Figure 7(e), we present the performance of LD_minCost. The main objective of LD_minCost is to redirect extra requests in an imbalanced load situation, minimizing the redirection cost in terms of traffic load and network proximity, without violating the (practical) server capacity constraints. We observe that none of the primary CDN servers operate beyond their capacity during simulations. In order to prevent wide oscillation in the load distribution decisions, incoming requests to the primary CDN stay within its domain as much as possible and only cross the inter-CDN barrier during excessive load imbalance. Since redirection cost is taken into account, requests are served by the optimally underloaded servers, in terms of network cost. As a result, a few primary CDN servers receive only relatively few requests, while other better located servers run close to their capacity. In addition, request rejections are minimized as LD_minCost leads to optimal server selections. Moreover, any dynamic load changes, e.g. sudden load increase in Server 9 at simulation time 5 (x10000*s*), are also taken into account and load is distributed in a timely fashion. Therefore, our approach performs well even under high traffic surges as flash crowds.

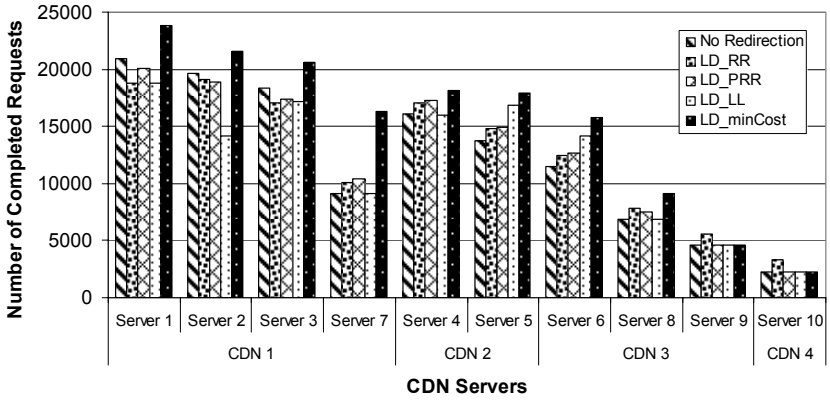

Figure 8: Number of completed requests at each server in different schemes.

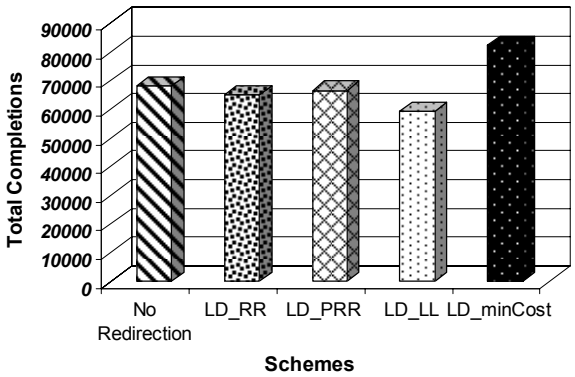

Figure 9: Total completions in each scheme.

5.2. Number of Completions

Now we evaluate the efficiency of our approach in terms of request completions. Figure 8 presents the average number of completed requests at each server over the simulation runs. It is found that in comparison to other alternatives, our approach is susceptible to handle more requests during load imbalance. LD_LL shows the worst performance among all the schemes. With no redirection, each CDN server attempts to serve the incoming requests to it, without any provision to redirect the request to a peer's server. Although servers are over-utilized during load imbalance, it does not perform well to serve all the incoming requests. Notably, LD_RR and LD_PRR show almost similar performance in terms of the number of completions at each server. While it is expected that LD_RR and LD_PRR would exhibit better performance than no redirection, they perform as poorly as the no redirection policy. This is because many requests are dropped due to service disruptions. We further elaborate on this aspect with supporting results in the next section.

Figure 9 demonstrates a similar trend, which presents the total completions in the peering CDNs system for different schemes. As adverted, LD_minCost assists in serving the highest number of requests collectively in the peering CDNs system. LD_RR and LD_PRR, along with no redirection demonstrate almost similar performance in terms of the total completions. As expected, LD_LL performs the worst and leads to serving the least number of requests in the system.

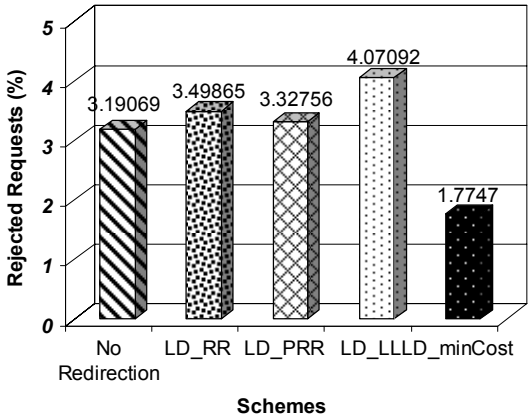

Figure 10: Service rejection rate for each scheme.

5.3. Service Disruptions

In this section, we investigate the impact of service unavailability for each scheme. Our redirection scheme directs comparatively more requests during load imbalance [21]. On the contrary, LD_RR, LD_PRR, and LD_LL do not show a high redirection percentage under traffic surge. Eventually, many requests are dropped as incoming requests arrive to a primary CDN server and find that the server is operating at its highest capacity. Thus, the inability to redirect more requests due to limited server capacity leads to significant service disruptions in these schemes. It is evident from Figure 10, which presents the percentage of disrupted services for each scheme, in terms of the average service rejection rate. In order to compute this performance metric, we first calculate the rejected service ratio as the number of requests that yielded a negative response (i.e. the system has not found a resource to serve this request), over the number of incoming requests. We then computed the average service rejection ratio as the average value over the number of total requests in the system. Figure 10 is obtained with a fixed number of 100000 requests. From the figure, we observe that LD_minCost clearly outperforms the other schemes, by exhibiting the lowest service rejection rate.

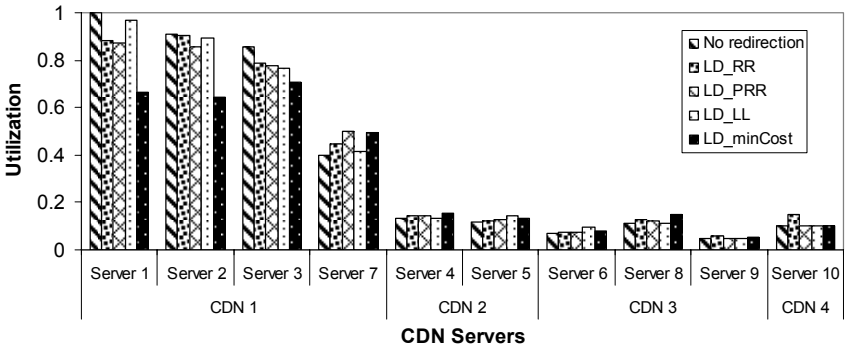

Figure 11: Server utilization for each scheme.

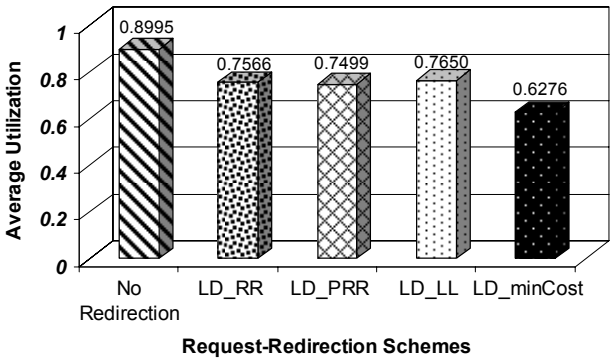

Fig. 12. Average utilization of the primary for each scheme.

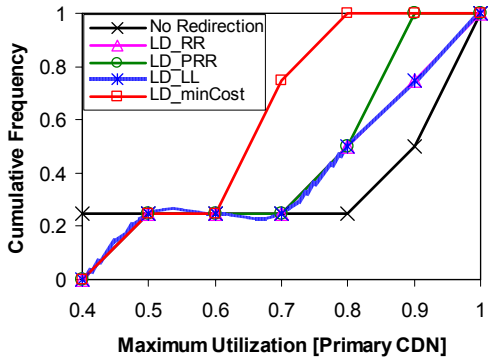

Figure 13: Comparison of the request-redirection schemes.

5.4. Server Utilization

We use utilization to determine whether the primary CDN servers operate under an acceptable level of load during high traffic surges. Figure 11 presents the utilization of the primary CDN servers for each request-redirection scheme. We observe that LD_minCost performs better than the other redirection schemes in order to perform load distribution of the overloaded primary CDN servers. With no redirection, most of the primary CDN servers receive excessive traffic and utilization stays over the alarm threshold (80% of a server's capacity). While LD_RR and LD_PRR demonstrate similar

characteristics to perform some load balancing, none of them reduces utilization of all the primary CDN servers below the threshold. LD_LL policy fails to perform load distribution among multiple servers and always overloads Server 1 and Server 2. On the other hand, LD_minCost exhibits the best performance by reducing all the primary CDN utilization below the threshold. Since requests stay within the primary CDN domain as much as possible to minimize redirection cost, a better located server, e.g. Server 3 in Figure 11, may show more utilization than its allies. However, still the primary CDN operates under an acceptable level of load. It is also evident from Figure 12, which presents the average utilization of the primary CDN system in each redirection scheme. LD_RR, LD_PRR, and LD_LL do not reduce the average utilization significantly below the threshold. Therefore, with these schemes primary CDN servers may be unable to receive more requests during sudden excessive traffic (flash crowds) in future and thus requests may have to be redirected outside the domain. With LD_minCost the average utilization of the primary CDN system is significantly brought down and makes its servers possible to cope up with succeeding traffic outbursts.

5.5. Redirection Performance

We investigate the impact of our request-redirection scheme on avoiding that a primary CDN serve is overloaded. Hence, we do not adopt traditional metrics such as the standard deviation of server utilizations for this purpose. We rather evaluate the performance of the request-redirection policies through the maximum utilization observed during a simulation run. The main performance metric we use is the cumulative frequency of maximum utilization, i.e. the probability for each utilization level that all server utilizations stay within that level. This metric provides an indication on the relative frequency of overloading. As for instance, a probability value of 0.8 for all servers to be less than 90% utilized implies that at least one server exceeds the 90% utilization level with probability 0.2.

Figure 13 summarizes the performance of the request redirection schemes in terms of cumulative frequency of maximum utilization of primary CDN servers. It shows that LD_minCost has a probability of 1.0 of not causing any primary CDN server to exceed 80% utilization (alarm threshold). From the figure we can see that other schemes such as LD_RR, LD_PRR and LD_LL do not perform well. Specifically, they exhibit a probability of 0.5 of not causing any Web server to exceed the threshold. Moreover, as predicted, with no redirection there is very low probability of only 0.25 that primary CDN servers will operate under the threshold.

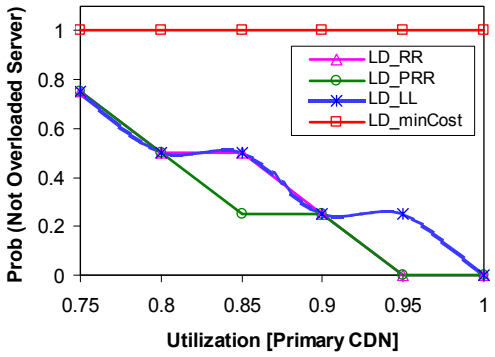

Figure 14: Sensitivity to utilization.

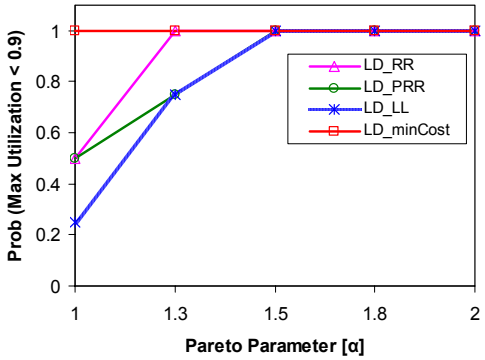

Figure 15: Sensitivity to traffic distribution.

5.6. Sensitivity Analysis

The performance of the redirection schemes could also be evaluated as a function of system parameters. We conduct a sensitivity analysis for the primary CDN, considering critical parameters such as *utilization* and *traffic distribution*. Changing other system parameters, such as the average number of requests or total simulation time, does not show noticeable differences among the redirection approaches. Therefore, we do not present those results here.

In Figure 14, we compare the sensitivity to the utilization for the primary CDN in each redirection scheme, using the probability that no server in its domain is overloaded as the performance metric. Therefore, we vary the alarm threshold from 0.75 to 1. We observe that our redirection scheme, LD_minCost, shows the best result in all the cases. Analogous conclusion can be drawn when we vary the distribution of users among the CDN servers, and obtain results based on 90% of the maximum utilization, i.e. the *Prob*(*Maximum Utilization* < 0.9). Figure 15 shows the probability that no server has a utilization higher than 90% as a function of the shape parameter α, which is varied from 1 (high variability) to 2 (moderate variability). Here, the performance metric (threshold) is changed from 80% to 90% to show the novelty of our scheme even under highly variable request pattern.

6. Related Work

Resource discovery is a popular topic in large-scale distributed systems; whereas, request-redirection is an indispensable enabling cornerstone for CDNs. Many research efforts have focused on these two topics separately in different domains, such as Grid computing, P2P-based systems, overlay networks, multi-agent systems, and ad-hoc networks. Analyses of previous research efforts, in relation to content internetworking, suggest that there has been only modest progress on dealing with resource discovery and request-redirection for peering CDNs. This is mainly due to

the lack of careful design to realize dynamism, and the presence of common stoppers such as technical complexity, operational, legal and business related issues in practical context.

The Content Distribution Internetworking (CDI) model [14] assumes a federation of CDNs, considering neighboring content networks as 'black-boxes'. It marks the participation in peering as an advertisement of capability, not a reservation. While the CDI model lays the foundation for CDN peering, detailing a usable resource discovery mechanism and an effective method for request-redirection is unexplored. While it recommends using a supervision function or an independent third party to supervise and manage all the CDN peers, the characteristics and implications of such supervision is yet to be defined. In addition to the CDI model, Barbir et al. [3] present request-routing mechanisms for content networks. Our work is complimentary to the CDI model in that we are aiming at devising mechanisms for resource discovery and request-redirection to perform dynamic load distribution in peering CDNs.

Previous work, as follow-ups of the CDI initiative, has mostly focused on peering performance evaluation, redirection modeling, and the applicability of anycasting as a redirection technique. Specifically, a brokerage system, called CDN Brokering [4], is developed and deployed on the Internet on a provisional basis. It presents IDNS, a specified request-routing DNS server, with a proprietary routing mechanism. The aim of this work is not to devise an optimal load distribution strategy, but to demonstrate the usefulness of brokering. Ercetin et al. [15] model request-redirection for CDN brokering as a delay-constrained routing problem. Unlike our work, the devised solution could only be applied to a snapshot of the system, thus limiting its scalability in peering CDNs domain. Alzoubi et al. [1] present a load-aware IP Anycast CDN architecture, incorporated with a route-controller, which takes server and network load into account to realize anycasting. This work establishes the applicability of anycasting as a redirection technique in CDNs. Our work is in line with this work, as we are aiming at minimizing the cost associated with request-redirection. However, our work differs by avoiding the use of a centralized route controller and by not following a post-processing approach to offload overloaded servers. Another notable work in this area is a request distribution system for PlanetLab [26], which considers server loads and known proximity information from pre-defined landmarks to route requests. The authors rely on a centralized approach for load index dissemination and arbitrary server selections. On the contrary, we follow a decentralized approach to improve adaptability to dynamic changes.

Amongst the work on load distribution strategies across geographically distributed Web servers, efforts by Conti et al. [11] and Cardellini et al. [8] could be adverted. The first is targeted to a QoS-based architecture for load distribution among replicated Web servers. The latter investigates the impact of redirection algorithms for load sharing. It proposes a Web cluster architecture where a DNS dispatcher is integrated with a redirection mechanism based on the HTTP protocol. While this approach might be effective to handle highly skewed loads, the focus is particularly on a single domain of clustered Web servers.

In the context of modeling traffic redirection between geographically distributed servers, work of Amini et al. [2] and Ranjan et al. [22] can be mentioned. The first presents a model for intelligent server selections over multiple, separately administrated server pools. While this work is appealing, it does not show the effectiveness of any particular redirection or load distribution policy. The latter presents WARD—an architecture for redirecting dynamic content requests from an overloaded Internet Data Center (IDC) to a remote replica. It is targeted to IDCs under the control of a single administrative entity.

There are also several representative research initiatives on resource discovery (and request redirection, to some extent) in large distributed networks, i.e. non-CDN domains such as Grids and P2P systems. Harchol-Balter [16] present a distributed resource discovery algorithm, called name-Dropper, for large distributed networks of computers, which is reported to be licensed to Akamai[6] for building Internet-wide content distribution system. This algorithm achieves near-optimal performance both with respect to time and network communication complexity. However, it may lead to the same particular target server selection by multiple originating machines. Lamnitchi et al. [17] study the resource discovery problem in a resource-sharing environment that combines the complexity of Grid and the dynamism of P2P networks. They propose a Grid emulator for evaluating resource discovery techniques based on request propagation. Unlike our work, this work is targeted to a P2P-based Grid system, which does not realize the full characteristics of peering CDNs. P2P networks support unstructured (e.g. Gnutella[7], Kazaa[8]) or structured (e.g. CAN [23], Chord [27], Pastry [24]) discovery services. In the first approach, hosts and resources are made available on the network without a global overlay planning. The latter exploits highly structured overlays, using distributed indexing data structure, such as Distributed Hash Tables (DHTs) to route queries over a P2P network. The peer discovery and membership services are mainly used for the construction and startup of P2P networks. Nevertheless, with the presence of mechanisms for virtualizing multiple providers and realizing request-redirection for dynamic load distribution, P2P techniques could be made effective for resource discovery in the context of peering CDNs.

7. Conclusion and Future Works

In this paper, we present resource discovery and request-redirection algorithms for a dynamic load distribution strategy, which alleviates any load imbalance in a peering CDNs system. Resource discovery in our approach follows a

[6] Akamai Technologies, Inc. http://www.akamai.com
[7] The Gnutella Protocol Specification. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf
[8] Kazaa. http://www.kazaa.com/

distributed and asynchronous nature, using a communication protocol that conservatively implements the public/subscribe paradigm. In addition, request-redirection occurs over distributed sets of servers, minimizing redirection cost. Specifically, in our approach, when any Web server in a peering CDNs arrangement reaches an overload condition exceeding the alarm threshold, the load distribution strategy reacts to redirect loads by selecting available optimally underloaded server(s), while not compromising network proximity. We validate our proposal with the aid of simulations, considering practical constraints and significant system parameters; and demonstrate its performance using performance metrics such as number of ongoing connections, number of completions, service disruptions, server utilization, and the cumulative frequency of maximum utilization. We also perform a sensitivity analysis of our redirection scheme by taking critical system parameters into account.

Our approach is novel as it seeks to perform load distribution in peering CDNs through distributed resource discovery and dynamic request-redirection, taking network cost (in terms of load and network proximity) into account, and coping up with the practical constraints in the CDN domain. Our approach can alleviate problems with the commonly used DNS-dispatching policies for load balancing, which does not provide sufficient control on user requests, e.g. it can have as little as 5% of requests in many instances [10]. Moreover, DNS-dispatching is less useful for fine-grained server selections. In contrast, we achieve a significant level of granularity, since the actual user requests (eyeballs) are redirected during load distribution among the Web servers in peering CDNs. Therefore, it endeavors to produce optimal Web server selections, even under degenerated load conditions.

There is room for further improving our approach in terms of communication cost. While the asynchronous feedback mechanism has little communication overhead, it may suffer from unnecessary message passing at periods when the traffic load matrix remain stable. As an alternative, we can use a triggered update strategy. In this technique, updates are triggered by the CDN servers as soon as they discover a potential change (measurement of old and new utilization and check whether it exceeds alarm threshold) in the user request patterns. Consequently, the communication overhead in the system is even reduced from periodic update, in order of the number of bytes/second.

Our approach can also be aided with a market model that takes into account the QoS-oriented aspects (user demand and satisfaction) of peer selection for request-redirection and results in tractable solutions. This model could be used to analyze the sensitivity of various performance metrics such as welfare and the expected net utility [18] of the involving entities with respect to parameters such as initial cost, economic value and popularity of the content, the available information, and the network cost. We are currently working on developing an economic model for this market managed peering CDNs. In future, we intend to implement this model in a system prototype, called MetaCDN[9] [5]. Our future work[10] also includes performing experiments in the real-world settings, such as PlanetLab[11], to validate the methodology presented in this paper. As far as the simulation is concerned, we aim to implement the full functionalities of the peering CDNs system within the simulator. We also seek to adopt traces of the Internet traffic for load characterization purposes. The use of actual CDN traces will help us to obtain reasonable load and network proximity estimates, whose accuracy could be improved through refinement, in response to changes in the network over time.

Acknowledgements

We are indebted to Christian Vecchiola for his selfless help and support during the simulation setup and experiments. We would like to thank Marcos Assunção, SungJin Choi, Mustafizur Rahman, Rajiv Ranjan and Saurabh Garg from the University of Melbourne, Australia for sharing thoughts and for making incisive comments and suggestions on this paper. We are also thankful to anonymous reviewers for their valuable and constructive comments to improve the paper's structure, quality and readability. This work is supported in part by the Australian Research Council (ARC), through the discovery project grant and Department of Innovation, Industry, Science and Research (DIISR), through the International Science Linkage (ISL) grant.

References

[1] Alzoubi, H. A., Lee, S., Rabinovich, M., Spatscheck, O., and Van der Merwe, J. E. Anycast CDNs revisited. In *Proc. of the 17th International Conference on World Wide Web*, ACM Press, NY, USA, pp. 277-286, Apr. 2008.

[2] Amini, L., Shaikh, A., and Schulzrinne, H. Modeling redirection in geographically diverse server sets. In *Proc. of the 12th International Conference on World Wide Web*, ACM Press, NY, USA, pp. 472-481, 2003.

[3] Barbir, A., Cain, B., Nair, R., and Spatscheck, O. Known content network (CN) request-routing mechanisms. RFC 3568, Jul. 2003.

[4] Biliris, A., Cranor, C., Douglis, F., Rabinovich, M., Sibal, S., Spatscheck, O., and Sturm, W. CDN brokering. *Computer Communications*, 25(4), pp. 393-402, 2002.

[5] Broberg, J. and Tari, Z. MetaCDN: Harnessing storage clouds for high performance content delivery. In *Proc. of the 6th International Conference on Service-Oriented Computing (ICSOC 2008)*, LNCS 5364, pp. 730–731, 2008.

[6] Buyya, R., Pathan, M., and Vakali, A. (Eds.) *Content Delivery Networks*. Springer-Verlag, Germany, 2008.

[7] Cardellini, V., Colajanni, M., and Yu, P. S. Dynamic load balancing on Web-server systems. *IEEE Internet Computing*, 3(3), 1999.

[9] It is an integrated overlay network that exploits existing "Storage Cloud" resources to provide high performance CDN for content creators. More information could be found at: www.metacdn.org

[10] For more information about our efforts on peering CDNs, please visit the project Web site at www.gridbus.org/cdn

[11] PlanetLab Consortium. http://www.planet-lab.org

[8] Cardellini, V., Colajanni, M., and Yu, P. S. Redirection algorithms for load sharing in distributed Web-server systems. In *Proc. of 19th IEEE International Conference on Distributed Computing Systems*, Austin, Texas, USA, May 1999.

[9] Chand, R., Cosnard, M., and Liquori, L. Powerful resource discovery for Arigatoni overlay network. *Future Generation Computer Systems*, 24(1), pp. 31-38, 2008.

[10] Colajanni, M. and Yu, P. S., and Cardellini, V. Dynamic load balancing in geographically distributed heterogeneous Web servers. In *Proc. of the 18th International Conference on Distributed Computing Systems*, pp. 295-302, 1998.

[11] Conti, M., Gregori, E., and Panzieri, F. QoS-based architectures for geographically replicated Web servers. *Cluster Computing*, Vol. 4, Kluwer Academic Publishers, The Netherlands, pp. 109-120, 2001.

[12] Crovella, M. E. and Bestavros, A. A self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6), 1997.

[13] Crovella, M. E., Taqqu, M. S., and Bestavros, A. Heavy-tailed probability distributions in the World Wide Web. *A Practical Guide To Heavy Tails*, Birkhauser Boston Inc., Cambridge, MA, USA, pp. 3-26, 1998.

[14] Day, M., Cain, B., Tomlinson, G., and Rzewski, P. A model for content internetworking. IETF RFC 3466, Feb. 2003.

[15] Ercetin, O. and Tassiulas, L. Request routing in content distribution networks. Technical Report, Sabanci University, Turkey, 2003. http://digital.sabanciuniv.edu/elitfulltext/3011800000049.pdf

[16] Harchol-Balter, M., Leighton, T., and Lewin, D. Resource discovery in distributed networks. In *Proc. of the 18th ACM Symposium on Principles of Distributed Computing*, ACM Press, NY, USA, pp. 229-237, 1999.

[17] Lamnitchi, A. and Foster, I. A peer-to-peer approach to resource location in grid environments. *Grid Resource Management* (Weglarz, J., Nabrzyski, J., Schopf, J., and Stroinski, M. *Eds.*), Kluwer Publishing, 2003.

[18] Mortazavi, B. and Kesidis, G. Model and simulation study of a peer-to-peer game with a reputation-based incentive mechanism. In *Proc. of the Information Theory and Applications (ITA) Workshop*, UC San Diego, Feb. 2006.

[19] Pallis, G. and Vakali, A. Insight and perspectives for content delivery networks. *Communications of the ACM*, 49(1), ACM Press, NY, USA, pp. 101-106, Jan. 2006.

[20] Pathan, M., Broberg, J., Bubendorfer, K., Kim, K. H., and Buyya, R. An architecture for virtual organization (VO)-based effective peering of content delivery networks. UPGRADE-CN'07, In *Proc. of the 16th IEEE International Symposium on High Performance Distributed Computing*, Monterey, California, USA, Jun. 2007.

[21] Pathan, M., Vecchiola, C., and Buyya, R. Load and proximity aware request-redirection for dynamic load distribution in peering CDNs. In *Proc. of the 16th International Conference on Cooperative Information Systems*, Monterrey, Mexico, 12-14 November, 2008.

[22] Ranjan, S., Karter, R., and Knightly, E. Wide area redirection of dynamic content by Internet data centers. In *Proc. of 23rd Annual IEEE Conference on Computer Communications*, 2004.

[23] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, ACM Press, NY, USA, 2001.

[24] Rowstron, A. and Druschel, P. Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[25] Schwartz, M. F., Emtage, A., Kahle, B., and Neuman, B. C. A comparison of Internet resource discovery approaches. *Computing Systems*, 5(4), pp. 461-493, 1992.

[26] Shnayder, V. Load and proximity aware request distribution. Princeton University Report, 2003. citeseer.ist.psu.edu/636009.html

[27] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, ACM Press, NY, USA, 2001.

[28] Van der Merwe, J. E. et al. Dynamic connectivity management with an intelligent route service control point. In *Proc. of ACM SIGCOMM Workshop on Internet Network Management*, ACM Press, NY, USA, pp.29-34, Oct. 2006.

[29] Verkaik, P., Pei, D., Scholl, T., Shaikh, T., Snoeren, A., and Van der Merwe, J. E. Wresting control from BGP: scalable fine-grained route control. In *Proc. of 2007 USENIX Annual Technical Conference*, Santa Clara, CA, USA, pp.295-308, Jun. 2007.

[30] Zhu, Y. and Hu, Y. Efficient, proximity-aware load balancing for DHT-based P2P systems. *IEEE Transactions on Parallel and Distributed Systems*, 16(4), IEEE Computer Society, Los Alamitos, CA, USA, pp. 349-361, Apr. 2005.