Distributed Load Management in Anycast-based CDNs

Abhishek Sinha *, Pradeepkumar Mani†, Jie Liu†, Ashley Flavel† and David A. Maltz†
*MIT, †Microsoft
Email: sinhaa@mit.edu, {prmani, Jie.Liu, ashleyfl, dmaltz}@microsoft.com

***Abstract*—Anycast is an internet addressing protocol where multiple hosts share the same IP-address. A popular architecture for modern Content Distribution Networks (CDNs) for geo-replicated HTTP-services consists of multiple layers of proxy nodes for service and co-located DNS-servers for load-balancing on different proxies. Both the proxies and the DNS-servers use anycast addressing, which offers simplicity of design and high availability of service at the cost of partial loss of routing control. Due to the very nature of anycast, load-management actions by a co-located DNS-server also affects loads at nearby proxies in the network. This makes the problem of distributed load management highly challenging. In this paper, we propose an analytical framework to formulate and solve the load-management problem in this context. We consider two distinct algorithms. In the first half of the paper, we pose the load-management problem as a convex optimization problem. Following a dual decomposition technique, we propose a fully-distributed load-management algorithm by introducing *FastControl* packets. This algorithm utilizes the underlying anycast mechanism itself to enable effective coordination among the nodes, thus obviating the need for any external control channel. In the second half of the paper, we consider an alternative greedy load-management heuristic, currently in production in a major commercial CDN. We study its dynamical characteristics and analytically identify its operational and stability properties. Finally, we critically evaluate both the algorithms and explore their optimality-vs-complexity trade-off using trace-driven simulations.**

***Index Terms*—Performance Analysis; Decentralized and Distributed Control; Optimization**

I. Introduction

Content Distribution Networks (CDN) are *de facto* architectures to transparently reduce latency between the end-users and geo-replicated Internet services. Edge-servers with cached contents serve as proxies to intercept some user requests and return contents without a round-trip to the data centers. Routing user-requests to the optimal proxies remains a challenge in managing modern CDN. Routing to a remote proxy may introduce extra round-trip delay, whereas routing to an overloaded proxy may cause the request to be dropped.

Anycast is a relatively new paradigm for CDN-management and there are already several commercial CDN in place today using anycast [1], [2], [3]. With anycast, multiple proxy-servers share the same IP-address. Anycast relies on routing protocols (such as, BGP) to route service-requests to any one of the geo-replicated proxies, over a *cheap* network-path [4]. Anycast based mechanisms have the advantage of being simple to deploy and maintain. Being available as a service in IPv6 networks, no global topology or state information are required for its use [5].

Although anycast routing can simplify the system-design and provide a high level of availability to users [6], it comes at the cost of partial loss of routing-control. This is because, a request may be routed to any one of the different geo-replicated proxies, determined solely by the routing protocol and network state. Since this routing is not under the control of the CDN-operator, the request may end up in an already overloaded proxy, deteriorating the situation further. There have been several attempts in the literature to tackle this lack of load-awareness issue with network-layer anycast. Papers [7], [8] consider "Active Anycast" where additional intelligence is incorporated into the routers based on RTT and network congestion. It is specifically targeted to reduce pure latency rather than server overload, thus yielding sub-optimal performance in a CDN setting. Alzoubi *et al.* [9] poses the anycast load-management problem as a General Assignment Problem, which is NP-hard to solve in general. The paper [10] proposes a new CDN architecture which balances server-load and network-latency via detailed traffic engineering.

In this paper, we focus our attention to the state-of-the-art CDN-architecture such as *FastRoute* [1], which uses DNS-based-redirection by the co-located DNS-servers for overload control in the local proxies. Since DNS is the first point-of-contact of users to the internet, DNS-redirection is a popular and effective way to mitigate overload [11], [12]. In this architecture, the proxies are arranged in layers of anycast rings and DNS is responsible for moving load across different layers. In the sequel, each proxy and the co-located DNS unit will be referred to simply as a *node*. See Figures 1 and 3 for an overview of the architecture.

An *overload* is said to occur when any individual proxy receives more requests than it can process. Since DNS is the primary control knob in this architecture, it is responsible for redirecting traffic to the next layer to alleviate overload. A fundamental problem with this approach is that not all users, that hit a given proxy, can be redirected by the co-located DNS. This is because an user's ISP could be obtaining a DNS-response from a DNS-server different than the co-located proxy. Hence, intuitively, the ability for a DNS-server to control overload at the corresponding co-located proxy depends on the fraction of oncoming traffic to the proxy that are routed by the co-located DNS-server. Informally, we refer to the above quantity as the *self-correlation* [1] of a given node. A formal definition of correlation and associated quantities will be given in section II.

Poor self-correlation could impair a node's ability to control overload in isolation. Hence, successful load management in layered CDN should involve coordinated action by DNS-servers in multiple nodes to alleviate overload. Thus the problem reduces to the DNS-plane determining the appropriate offload or redirection probabilities at each node to move traffic from the overloaded proxies to the next layer. This control-decision could be based on variety of information such as load on each proxy, DNS-HTTP correlation etc. From a practical point of view, not all of these quantities are easily measured and communicated to wherever is needed. Thus a centralized solution is not practically feasible and the challenge is to design a provably optimal, yet completely distributed load management algorithm.

Our key contributions in this paper are as follows:

- In section II, we present a simplified mathematical model for anycast-based load-management in modern CDNs. Our model is general enough to address the essential operational problems faced by the CDN-operators yet tractable enough to draw meaningful analytical conclusions about their operational characteristics.
- In section III, we pose the load management problem as a convex optimization problem and derive a *dual* algorithm to solve it in a distributed fashion. The key to our distributed implementation is the *Lagrangian decomposition* and the use of *FastControl* packets, which exploits the underlying anycast capability to enforce coordination among the nodes in a distributed fashion. To the best of our knowledge, this is the first instance of such a decomposition technique employed in the context of load-management in CDN.
- In section IV, we consider an existing heuristic used in *FastRoute*, Microsofts CDN for many first party websites and services it owns[13]. We model the dynamics of the heuristic using non-linear system-theory and derive its several important operational characteristics. To provide additional insight, a two-node system is analyzed in detail and it is shown, rather surprisingly, that given the "self-correlations" of the nodes are sufficiently high, this heuristic algorithm is able to control an incoming-load of *any* magnitude, however large. Unfortunately, this theoretical guarantee breaks down once this correlation property no longer holds. In this case, the dual algorithm developed in section III performs better than the heuristic.
- In section V, we critically evaluate relative performances of both the optimal and heuristic algorithms through extensive simulations. Our simulation is trace-driven in the sense we use real correlation parameters collected over months from an operational CDN [1].

II. System Model

Nodes and Layers: We consider a CDN system consisting of two layers: *primary* (also referred to as L_1) and *secondary*

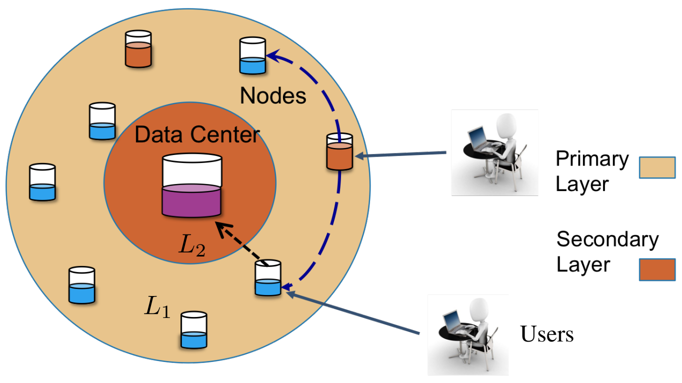


Fig. 1. A toy CDN with two anycast layers. Solid arrows denote user connections, while dotted arrows denote the effect of diverting traffic by overloaded nodes.

(also referred to as L_2), as shown in Fig 1. The primary layer hosts a total of N nodes. Each node consists of a collocated DNS and a proxy server. See Figure 3 for a schematic diagram. The proxy servers are the end-points for HTTP-requests. The i^{th} proxy has a (finite) processing-capacity of T_i requests per unit time. The secondary layer (L_2) consists of a large data-center, with practically infinite processing-capacity. Since the proxies are distributed throughout the world, an average user is typically located near to some proxy, resulting in relatively small round-trip latency. On the other hand, the data-center in L_2 is typically located further from the average user-base, resulting in significantly high round-trip latency. Total response time for a user is the sum of round-trip latency and processing time of the server (either proxy or the data-center), which we would like to minimize.

Anycast Addressing : All proxies in the primary layer share the same IP-address $\mathcal{I}_1$ and the data-center in L_2 has a distinct IP-address $\mathcal{I}_2$. This method of multiple proxies sharing the same IP-addresses is known as *anycast* addressing and is widely used for geo-replicated cloud services [14], [9].

Control decisions : When a DNS-query arrives at a node i, asking for an IP-address of an HTTP-server, the collocated DNS-server decides whether to return the address $\mathcal{I}_2$ (meaning, to offload the incoming request to the data-center) or the anycast address $\mathcal{I}_1$ (meaning, not to offload the incoming request to the far-away data-center and serve it in some proxy in the primary layer itself). This (potentially randomized) binary decision could be based on the following observables available to the node i :

1) *DNS-query* arrival rate at node i's co-located DNS-server, given by A_i requests per unit time[1]. Each DNS-query accounts for a certain amount of user HTTP-load which is influenced by the DNS-response at node i; We normalize A_i so that each unit of A_i corresponds to a unit of HTTP-load; As an example, if 10% of DNS responses at node i returns the address $\mathcal{I}_2$, then the total

[1]We assume that A_i's are piecewise constant and do not change during the transient period of the load-management algorithms discussed here.

load shifted to L_2 due to node i's DNS is $0.1A_i$.

2) *Content-request* arrival rate (or simply, the *load*) at the i^{th} collocated proxy, given by $S_i(t)$ requests per unit time. This is clearly a result of A_i's and different nodes' offload-decisions (Eqn.(2)). Since we focus on web-applications, we will use the phrase content-requests and HTTP-requests interchangeably. The observable $S_i(t)$ is available locally at node i. We assume that the workloads introduced by different requests are roughly the same.

Anycasting and inter-node coupling: When a DNS-query arrives at a node i and the corresponding DNS-server returns the L_1 anycast-address $\mathcal{I}_1$, the request may be routed to any of one of the N proxy-nodes in the primary layer for service, depending on the corresponding ISP's routing policy, network congestion and other random factors. We assume that a typical DNS-query, routed to L_1 by the DNS at node i, arrives for service at node j's proxy with probability $C_{ij} > 0$, where

$$\sum_{j=1}^{N} C_{ij} = 1, \quad \forall i = 1, 2, \ldots, N \tag{1}$$

The matrix $\boldsymbol{C} \equiv [C_{ij}]$ can be determined empirically by setting up an experiment similar to the one described in [12]. In this paper, we primarily focus on effects that arise from non-trivial couplings among the nodes.

Input-Output Equations: Assume that, due to action of some control-strategy π, the collocated DNS at node i decides to divert $1 - x_i^{\pi}(t)$ fraction of incoming DNS-queries (given by A_i) to Layer L_2 at time t ($0 \leq x_i^{\pi}(t) \leq 1$). Thus it routes $x_i^{\pi}(t)$ fraction of the incoming requests to different proxies in the layer L_1. Hence the total HTTP-request arrival rate, $S_i(t)$, at the i^{th} proxy may be written as

$$S_i(t) = \sum_{j=1}^{N} C_{ji} x_j^{\pi}(t) A_j, \quad \forall i = 1, 2, \ldots, N \tag{2}$$

A local control strategy π is identified by a collection of mappings $\boldsymbol{\pi} = \Big(x_i^{\pi}(\cdot), i = 1, 2, \ldots, N\Big)$, given by $\boldsymbol{x}_i^{\pi} : \Omega_i^t \times t \to [0, 1]$, where Ω_i^t is the set of all observables at node i up to time t.

III. An Optimization Framework

A. Motivation

In our context, the central objective of a load-management policy π is to route as few requests as possible to the secondary layer (due to its high round-trip latency), without overloading the primary-layer proxies (due to their limited capacities). Clearly, these two objectives are at conflict with each other and we need to find a suitable compromise. The added difficulty, which makes the problem fundamentally challenging is that, the nodes are autonomous agents and take their redirection decisions on their own, based on their local observables only. As an example, a simple locally-greedy heuristic for node i could be to redirect requests to L_2 (i.e. decrease $x_i(t)$) whenever its co-located proxy is overloaded

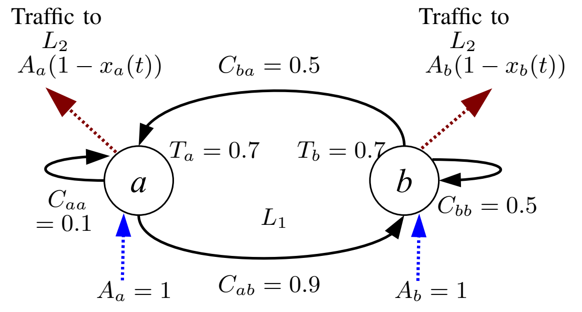


Fig. 2. A two-node system illustrating locally uncontrollable overload at the node b

(i.e., $S_i(t) > T_i$) and redirect requests to L_1 (i.e., increase $x_i(t)$) whenever the proxy is under-loaded (i.e., $S_i(t) < T_i$). This forms the basis of the strategy adopted in [1].

This greedy strategy appears to be quite appealing for deployment, due to its extreme simplicity. However, in the next subsection III-B, we show by a simple example that, in the presence of significantly high cross-correlations among the nodes, this simple heuristic could lead to an *uncontrollable overload situation*, an extremely inefficient operating point with degraded service quality. This example will serve as a motivation to come up with a more efficient distributed load-management algorithm, that we develop subsequently.

B. Locally Uncontrollable Overload: An Example

Consider a CDN, hosting only two nodes a and b in the primary layer, as shown in Figure 2. The (normalized) DNS-query arrival rates to the nodes a and b are $A_a = A_b = 1$. Suppose the processing capacities (also referred to as *thresholds*) of the corresponding proxies are $T_a = T_b = 0.7$. With the correlation values shown in figure 2, the HTTP-request arrival rates (load) to the proxies at a and b are given as

$$S_a(t) = 0.1x_a(t) + 0.5x_b(t) \tag{3}$$
$$S_b(t) = 0.9x_a(t) + 0.5x_b(t) \tag{4}$$

Since $0 \leq x_a(t), x_b(t) \leq 1$, it is clear that

$$S_a(t) \leq 0.1 \times 1 + 0.5 \times 1 = 0.6 < 0.7 = T_a, \ \forall t$$

Thus, the proxy at node a will be under-loaded irrespective of the load-management policy π in use. Consequently, under the greedy-heuristic (formally, algorithm 2 in Section IV), the collocated DNS-server at node a will *greedily* increase its L_1 redirection probability $x_a(t)$ such that $x_a(t) \nearrow 1$ in the steady-state (note that, node a acts independently as it does not have node b's loading information). This, in turn, overloads the proxy in node b because the steady-state HTTP-load at proxy b becomes

$$\begin{aligned} S_b(\infty) &= 0.9x_a(\infty) + 0.5x_b(\infty) \\ &= 0.9 \times 1 + 0.5x_b(\infty) \\ &\geq 0.9 > 0.7 = T_b. \end{aligned}$$

Since node b is overloaded in the steady-state, under the action of the above greedy heuristic, it will (unsuccessfully) try to avoid the overload by offloading the incoming DNS-queries to L_2, as much as it can, by letting $x_b(t) \searrow 0$.

Thus the steady-state operating point of the algorithm will be $x_a(\infty) = 1, x_b(\infty) = 0$, with node b overloaded. It is interesting to note that poor self-correlation of node a ($C_{aa} = 0.1$) causes the other node b to overload, even under the symmetric DNS-query arrival patterns. Also, this conclusion does not depend on the detailed control-dynamics of the offload probabilities (viz. the instantaneous values if $\dot{x}_1(t)$ and $\dot{x}_2(t)$). Since the overload condition at the node-b can not be overcome by isolated action of node-b itself, we say that node-b is undergoing a *locally uncontrollable overload situation*.

From the point-of-view of the overall system, this is an extremely inefficient operating condition, because a large fraction of the incoming requests either gets dropped or delayed due to the overloaded node-b. This poor operating point could have been avoided provided the nodes somehow mutually co-ordinate their actions[2]. It is not difficult to realize that the principal reasons behind the locally uncontrollable overload situation in the above example are as follows:

- (1) distributed control with local information
- (2) poor self-correlation of node a ($C_{aa} = 0.1$)

The factor (1) is fundamentally related to the distributed nature of the system and requires coordinations among the nodes. In our distributed algorithm [1] we address this issue by introducing the novel idea of *FastControl* packets. This strategy does not require any explicit state or control-information exchange. Regarding the factor (2), we intuitively expect that the local greedy-heuristic should work well if the self-correlation of the nodes (i.e. C_{ii}) are not too small. In this favorable case, the system will be loosely coupled, so that each node accounts for a major portion of the oncoming load to itself. In section IV, we will return to a variant of this local heuristic used in FastRoute [1] and derive analytical conditions under which the above intuition holds good.
In this section, we take a principled approach and propose an iterative load-management algorithm, which is provably optimal for arbitrary system-parameters ($\boldsymbol{A}, \boldsymbol{C}$). In this algorithm, it is enough for each node i to know its own local DNS and HTTP-request arrival rates (i.e., A_i and $S_i(t)$ respectively) and the entries corresponding to the (static) i^{th} row and column of the correlation matrix $\boldsymbol{C}$ (i.e., $C_{i\cdot}, C_{\cdot i}$). No non-local knowledge of the dynamic loading conditions of other nodes $j \neq i$ is required for its operation.

C. Mathematical formulation

Consider the following optimization problem. The decision variables $(\boldsymbol{x}, \boldsymbol{S})$ have the same interpretation as above. The cost-functions, constraints and their connection to the load-management problem are discussed in detail subsequently.

$$\text{Minimize} \qquad W(\boldsymbol{x}, \boldsymbol{S}) \equiv \sum_{i=1}^{N} \big(g_i(S_i) + h_i(x_i)\big) \tag{5}$$

[2]Another trivial solution to avoid overload could be to offload all traffic from all nodes to L_2, i.e. $x_i(t) = 0, \forall i, \forall t$. However, this is highly inefficient because it is tantamount to not using the primary servers at all.

Subject to,

$$S_i = \sum_{j=1}^{N} C_{ji} A_j x_j, \quad \forall i = 1, 2, \ldots, N \tag{6}$$

$$\boldsymbol{x} \in X, \boldsymbol{S} \in \Sigma_T$$

Discussion: The <u>first component</u> of the cost function, $g_i(S_i)$, denotes the cost for overloading the i^{th} proxy. In our numerical work, we take $g_i(\cdot)$ to be proportional to the average aggregate queuing delay for an $M/G/1$ queue with processing-capacity T_i [15], i.e.,

$$g_i(S_i) \quad = \quad \begin{cases} \frac{\eta_i S_i}{1 - \frac{S_i}{T_i}}, & \text{if } S_i \leq T_i \\ \infty \quad \text{o.w.} \end{cases} \tag{7}$$

Here η_i is a positive constant, denoting the relative cost (say, in dollars) due to per unit delay.
The <u>second component</u> of the cost function $h_i(x_i)$ denotes the cost due to *round-trip-latency* of requests routed to the secondary layer (L_2). As an example, in a popular model [16] the delay incurred by a single packet over a congested path varies *affinely* with the offered load. Since the rate of traffic sent to the secondary layer by node i is $A_i(1 - x_i)$, in this model, the cost-function $h_i(x_i)$ may be written as follows

$$h_i(x_i) = \gamma_i A_i (1 - x_i)\big(d_i + A_i(1 - x_i)\big) \tag{8}$$

where d_i is a (suitably normalized) round-trip-latency parameter from the node i to L_2 and γ_i is a positive constant denoting the relative cost due to per unit latency. We use the cost-functions described above for our numerical work.
The <u>constraint set</u> $X = [0, 1]^N$ represents the N-dimensional unit hypercube and the set Σ_T captures the capacity constraints of the proxies, e.g., if the proxy i has capacity T_i then we have

$$\Sigma_{\boldsymbol{T}} = \{\boldsymbol{S} : S_i \leq T_i, \forall i = 1, 2, \ldots, N\}$$

In general, the functions $g_i(\cdot), h_i(\cdot)$ are required to be closed, proper and convex [17]. We also assume the functions $g_i(\cdot)$s to be monotonically increasing. Hence, we can replace the equality constraint (6) by the following inequality constraint, without loss of optimality

$$\sum_{j=1}^{N} C_{ji} A_j x_j \leq S_i, \quad \forall i = 1, 2, \ldots, N$$

This is because, if the optimal S_i^* is strictly greater than the LHS, we can strictly (and feasibly) reduce the objective value by reducing S_i^* to the level of LHS, resulting in contradiction. Hence, the above load management problem is equivalent to the following optimization problem $\mathbf{P}_1$:
Minimize $\quad W(\boldsymbol{x}, \boldsymbol{S}) = \sum_{i=1}^{N} \big(g_i(S_i) + h_i(x_i)\big)$
Subject to,

$$\sum_{j=1}^{N} C_{ji} A_j x_j \leq S_i, \quad \forall i = 1, 2, \ldots, N \tag{9}$$

$$\boldsymbol{x} \in X, \boldsymbol{S} \in \Sigma_{\boldsymbol{T}} \tag{10}$$

Where, $X = [0,1]^N$ and $\Sigma_{\boldsymbol{T}} = \{\boldsymbol{S} : S_i \leq T_i, \forall i = 1, 2, \ldots, N\}$.

Since the objective functions as well as the constraint sets of the problem $\boldsymbol{P}_1$ are all convex [17], we immediately have the following lemma:

Lemma 3.1: The problem $\mathbf{P}_1$ is convex.

D. The Dual Decomposition Algorithm

In this section we derive a dual algorithm [18], [19], [20] for the problem $\mathbf{P}_1$ and show how it leads to a distributed implementation with negligible control overhead.
By associating non-negative dual variable μ_i to the i^{th} constraint in (9) for all i, the Lagrangian of $\mathbf{P}_1$ reads as follows:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{S}, \boldsymbol{\mu}) = \sum_{i=1}^{N} \big(g_i(S_i) - \mu_i S_i\big) + \sum_{i=1}^{N} \Bigg(h_i(x_i) + A_i x_i \Big(\sum_{j=1}^{N} \mu_j C_{ij}\Big)\Bigg) \tag{11}$$

This leads to the following dual objective function [17]

$$D(\boldsymbol{\mu}) = \inf_{\boldsymbol{x} \in X, \boldsymbol{S} \in \Sigma_{\boldsymbol{T}}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{S}, \boldsymbol{\mu}) \tag{12}$$

We now exploit the *separability* property of the dual objective (11) to reduce the problem (12) into following two one-dimensional sub-problems:

$$\left.\begin{aligned} S_i^*(\boldsymbol{\mu}) &= \inf_{0 \leq S_i \leq T_i} \Big(g_i(S_i) - \mu_i S_i\Big) \\ x_i^*(\boldsymbol{\mu}) &= \inf_{0 \leq x_i \leq 1} \Big(h_i(x_i) + A_i \beta_i(\boldsymbol{\mu}) x_i\Big) \end{aligned}\right\} \tag{13}$$

The scalar $\beta_i(\boldsymbol{\mu})$ is defined as

$$\beta_i(\boldsymbol{\mu}) = \sum_{j=1}^{N} \mu_j C_{ij} = \boldsymbol{C}_i^T \boldsymbol{\mu}, \tag{14}$$

where $\boldsymbol{C}_i$ is the i^{th} row of the correlation-matrix $\boldsymbol{C}$.
The factor $\beta_i(\boldsymbol{\mu})$ couples the offload decision of node i with the entire network. Once the value of $\beta_i(\boldsymbol{\mu})$ is available to the node i, it has all the required information to *locally* solve the corresponding sub-problems (13) and hence evaluate the dual objective $D(\boldsymbol{\mu})$ for a fixed $\boldsymbol{\mu} \geq \mathbf{0}$. These solutions may even be obtained in closed form in some cases. In sub-section III-F, we will show how this factor $\beta_i(\boldsymbol{\mu})$ may be made available to each node i on-the-fly.
With the stated assumptions on the cost functions, there will be no duality-gap [17]. Convex duality theory guarantees the existence of an optimal dual variable $\boldsymbol{\mu}^* \geq \mathbf{0}$ such that solution to the relaxed problem (13) corresponding to $\boldsymbol{\mu}^*$ gives an optimal solution to the original constrained optimization problem $\mathbf{P}_1$. To obtain the optimal dual variable $\boldsymbol{\mu}^*$, we solve dual of the problem $\mathbf{P}_1$, given as follows

$$\begin{aligned} \textbf{Maximize} \quad & D(\boldsymbol{\mu}) \\ \textbf{subject to,} \quad & \boldsymbol{\mu} \geq \mathbf{0} \end{aligned} \tag{15}$$

The dual problem given in Eqn. (15) is well-known to be convex [17]. To solve the dual problem, we use the dual super-gradient algorithm [21], which will be shown to be amenable to a distributed implementation.
At the k^{th} step of the iteration, a super-gradient $\boldsymbol{g}(\boldsymbol{\mu}(k))$ of the dual function $D(\boldsymbol{\mu})$ at the point $\boldsymbol{\mu} = \boldsymbol{\mu}(k)$ is given by $\partial D(\boldsymbol{\mu}(k)) = \boldsymbol{S}^{\text{obs}}(k) - \boldsymbol{S}(k)$ [17], where $S_i^{\text{obs}}(k)$ is the observed rate of arrival of incoming traffic at proxy i, i.e.,

$$S_i^{\text{obs}}(k) \equiv \sum_{j=1}^{N} C_{ji} A_j x_j^*(k), \tag{16}$$

and $x_i^*(k)$ and $S_i^*(k)$ are the primal variables obtained from Eqn. (13), evaluated at the current dual variable $\boldsymbol{\mu} = \boldsymbol{\mu}(k)$. Following a super-gradient step, the dual variables $\boldsymbol{\mu}(k)$ are iteratively updated component-wise at each node i as follows:

$$\mu_i(k+1) = \Big(\mu_i(k) + \alpha\big(S_i^{\text{obs}}(k) - S_i^*(k)\big)\Big)^+ \tag{17}$$

Here α is a small positive step-size constant, whose appropriate value will be given in Theorem (3.3). Since the problem-parameters slowly vary over time, a stationary algorithm is practically preferable. Hence, we used a constant step-size α, rather than a sequence of diminishing step-sizes $\{\alpha_k\}$. The above constitutes theoretical underpinning of steps (5) and (6) of the distributed algorithm given in Algorithm 1.

E. Convergence of the Dual Algorithm

To prove the convergence of the above algorithm, we first *uniformly* bound the ℓ_2 norm of the super-gradients $\boldsymbol{g}(\boldsymbol{\mu}(k))$:

$$\boldsymbol{g}(\boldsymbol{\mu}(k)) \equiv \boldsymbol{S}^{\text{obs}}(k) - \boldsymbol{S}(k) = \Bigg(\sum_{j=1}^{N} C_{ji} A_j x_j^*(k) - S_i^*(k)\Bigg)_{i=1}^{N}$$

We start with the following lemma.

Lemma 3.2: If the total external DNS-query arrival rate to the system is bounded by $A_{\max}$ (i.e. $\sum_i A_i \leq A_{\max}$) and the maximum processing-capacity of individual proxies is bounded by $T_{\max}$ (i.e. $T_i \leq T_{\max}, \forall i$) then, for all $k \geq 1$

$$||\boldsymbol{g}(\boldsymbol{\mu}(k))||_2^2 \leq A_{\max}^2 + N T_{\max}^2 \tag{18}$$

Proof: See Appendix A. of [22]. ■

Upon bounding the super-gradients uniformly for all k, the convergence of the dual algorithm follows directly from Proposition 2.2.2 and 2.2.3 of [23]. In particular, we have the following theorem:

Theorem 3.3: For a given $\epsilon > 0$, let the step-size α in Eqn. (17) be chosen as $\alpha = \frac{2\epsilon}{A_{\max}^2 + N T_{\max}^2}$. Then,

- The sequence of solutions produced by the dual algorithm described above converges within an ϵ-neighbourhood the optimal objective value of the problem $\mathbf{P}_1$.
- The rate of convergence of the algorithm to the ϵ-neighborhood of the optima after k-steps is given by $c/\sqrt{k}$ where $c \sim \Theta(\sqrt{N})$.

The above result states that the rate of convergence of the dual algorithm decreases roughly at the rate of $\Theta(\sqrt{N})$, where N

is the total number of nodes in the system. This is expected as more nodes in the system would warrant greater amount of inter-node coordination to converge to the optima.

F. FastControl *Packets and Distributed Implementation*

In the previous subsection, we derived a provably optimal load-management algorithm, which is implementable in practice, provided each node i knows how to obtain the value of $\beta_i(\boldsymbol{\mu}(k))$ in a decentralized fashion. To accomplish this, we now introduce the novel idea of FastControl packets. In brief, it exploits the underlying anycast architecture of the system (Eqn.(2)) for *in-network computation* of the coupling factor $\beta_i(\boldsymbol{\mu}(k))$ (Eqn. (14)) for all i.
FastControl packets are special-purpose control packets (different from the regular data-packets), each belonging to any one of the N distinct *categories*. The category of each FastControl packet is encoded in its packet-header. These packets are generated in a controlled manner by using a javascript embedded in responses to user DNS-requests (similar to how data was generated to calculate the C matrix offline [1]). The javascript forces users to download a small image from a URL that is not affected by the load management algorithm. DNS-servers in each node are configured to respond back with anycast IP address for the primary layer (i.e. $\mathcal{I}_1$) for this special DNS-query. The use of various categories of FastControl packets will be clear from the description of the following distributed protocol used for determining $\beta_i(\boldsymbol{\mu}(k))$:

- At step k, each node i **forces** generation of FastControl packets of category j (through its response to DNS-queries) at the rate
$$r_{ij}(k) = \gamma\mu_i(k)\frac{C_{ji}}{C_{ij}}, \quad j = 1, 2, \ldots, N \tag{19}$$
Note that this is locally implementable, since the value of the dual variable $\mu_i(k)$ is locally available at each node i. Here $\gamma > 0$ is a fixed system parameter, indicating the rate of control packet generation.
- At each step k, each node i also **monitors** the rate of reception of FastControl packet of category i, denoted by $R_i(k)$. Using equation (19), the total rate of reception $R_i(k)$ of i^{th} category FastControl packets at node i is obtained as follows
$$\begin{aligned} R_i(k) &= \sum_{j=1}^{N} r_{ji}(k)C_{ji} = \sum_{j=1}^{N} \gamma\mu_j(k)\frac{C_{ij}}{C_{ji}}C_{ji} \\ &= \gamma\beta_i(\boldsymbol{\mu}(k)) \end{aligned}$$
Thus,
$$\beta_i(\boldsymbol{\mu}(k)) = \frac{1}{\gamma}R_i(k) \tag{20}$$

Hence, the value of $\beta_i(\boldsymbol{\mu}(k))$ at node i can be obtained locally by monitoring the rate of receptions of FastControl packets at the collocated proxy. A complete pseudocode of the algorithm is provided below. See Figure (3) for a schematic diagram of a node implementing the algorithm.

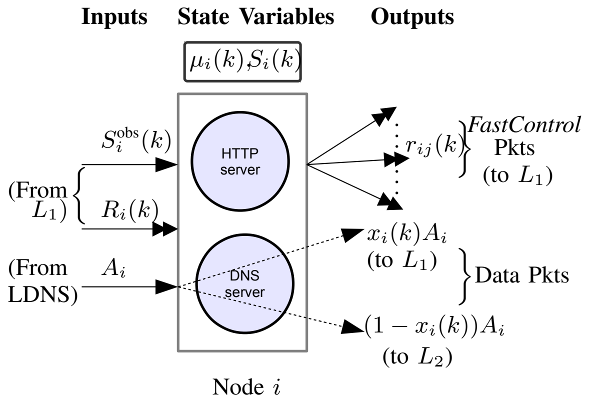


Fig. 3. Node-i implementing the dual algorithm

Algorithm 1 Distributed Dual Decomposition Algorithm Running at Node i

```
1: Initialize: $\mu_i(0) \leftarrow 0$
2: for $k = 1, 2, 3, \ldots$ do
3:     Monitor $A_i(k), S_i^{\text{obs}}(k), R_i(k)$;
4:     Set $\beta_i(k) \leftarrow \frac{1}{\gamma}R_i(k)$;
5:     Update Primal variables:
           $S_i(k) \leftarrow \inf_{0 \le S_i \le T_i} \big(g_i(S_i) - \mu_i(k)S_i\big)$
           $x_i(k) \leftarrow \inf_{0 \le x_i \le 1} \big(h_i(x_i) + A_i(k)\beta_i(k)x_i\big)$
6:     Update Dual variable:
           $\mu_i(k+1) \leftarrow \big(\mu_i(k) + \alpha(S_i^{\text{obs}}(k) - S_i(k))\big)^+$
7:     Via DNS-response, force users to generate FastControl
       packets of category $j$, destined to $L_1$, at the rate
           $r_{ij}(k+1) = \gamma\mu_i(k+1)\frac{C_{ji}}{C_{ij}}, \quad \forall j = 1, 2, \ldots, N$
8:     For an incoming DNS-query, respond with the anycast IP-
       address for $L_1$ with probability $x_i(k)$ and IP-address for $L_2$
       with probability $(1 - x_i(k))$.
9: end for
```

IV. The Greedy Load-Management Heuristic

In this section, we focus exclusively on the distributed load management heuristic implemented in FastRoute [1], a commercial layered-CDN. This heuristic ignores inter-node correlations altogether. Thus, when an individual proxy becomes overloaded, the co-located DNS-server modifies its DNS-response to redirect more traffic to the data-centers (L_2) and vice versa. This simple mechanism is reported to work well in practice when there is high correlation (60-80%) between the node receiving the DNS-query and the node receiving the corresponding HTTP-request. However, in the event of sudden bursts of traffic, e.g., *Flash Crowds*, this greedy heuristic often leads to an uncontrollable overload situation which necessitates manual intervention [1].

Algorithmic challenges: With only local information available at each DNS-server, it faces the following dilemma: offload too little to L_2 and the collocated proxy, if overloaded, remains overloaded; offload too much and the users are directed to remote data-centers and receive an unnecessarily delayed response, due to high round-trip latency. The coupling among the nodes because of inter-node correlation makes this problem highly challenging and gives rise to the so-called uncontrollable overload, discussed earlier in section III-B.

A pseudo-code showing the general structure of FastRoute's heuristic, running at a node i, is provided below. An explicit control-law modeling the heuristic will be given in section IV-A.

Algorithm 2 Decentralized greedy load-management heuristic used in *FastRoute* [1], running at the node i

```
1: for $t = 1, 2, 3, \dots$ do
2:     if the $i^{\text{th}}$ proxy is under loaded $(S_i(t) \leq T_i)$ then
3:         increase $x_i(t)$ proportional to $-(S_i(t) - T_i)$
4:     else
5:         decrease $x_i(t)$ proportional to $(S_i(t) - T_i)$
6:     end if
7: end for
```

Motivation for analyzing the heuristic: The optimal algorithm of section-III requires the knowledge of the correlation matrix $\boldsymbol{C}$ and needs to utilize additional FASTCONTROL packets for its operation. In this section we analyze performance of the greedy heuristic, currently implemented in *FastRoute* [1], which does not have these implementation complexities. Since this heuristic completely ignores the inter-node correlations (given by the matrix $\boldsymbol{C}$), it cannot be expected to achieve the optima of the problem $\boldsymbol{P}_1$, in general. Instead, we measure its performance by a coarser performance-metric, given by the number of proxies that undergo uncontrollable overload condition (refer to section III-B) under its action. Depending on target applications, this metric is often practically good enough for gauging the performance of CDNs.

A. Analysis of the Greedy Heuristic

As before, let $x_i(t)$ denote the probability that an incoming DNS-query to the node i at time t is returned with the anycast address of the primary layer L_1. Hence, the total rate of incoming load to the proxy i at time t is given by,

$$S_i(t) = \sum_{j=1}^{N} C_{ji} A_j x_j(t), \quad i = 1, 2, 3, \dots, N \tag{21}$$

The above system of linear equations can be compactly written as follows

$$\boldsymbol{S}(t) = \boldsymbol{B}\boldsymbol{x}(t) \tag{22}$$

Where,

$$\boldsymbol{B} \equiv \boldsymbol{C}^T \textbf{diag}(\boldsymbol{A}). \tag{23}$$

Let the vector $\boldsymbol{T}$ denote the processing-capacities (thresholds) of the proxies. As described above, FastRoute's greedy heuristic (2) monitors the overload-metric $S_i(t) - T_i$ and if it is positive (i.e., the node i overloaded), it reduces $x_i(t)$ (i.e., $\frac{dx_i(t)}{dt} < 0$) and if the overload-metric is negative (i.e., the node i under-loaded), it increases $x_i(t)$ (i.e., $\frac{dx_i(t)}{dt} > 0$) proportional to the overload. We consider the following explicit control-law complying with the above general principle:

$$\frac{dx_i(t)}{dt} = -\beta R(x_i(t))\big(\boldsymbol{B}\boldsymbol{x}(t) - \boldsymbol{T}\big)_i, \quad \forall i \tag{24}$$

The factor $R(x_i(t)) \equiv x_i(t)(1 - x_i(t))$ is a non-negative *damping* component, having the property that $R(0) = R(1) = 0$. This non-linear factor is responsible for restricting the trajectory of $\boldsymbol{x}(t)$ to the N-dimensional unit hypercube $\boldsymbol{0} \leq \boldsymbol{x}(t) \leq \boldsymbol{1}$, ensuring the feasibility of the control (24)[3]. The scalar $\beta > 0$ is a sensitivity parameter, relating the robustness of the control-strategy to the local observations at the nodes. The following theorem establishes soundness of the control (24):

Theorem 4.1: Consider the following system of ODE

$$\dot{x_i}(t) = -R(x_i(t))(\boldsymbol{B}\boldsymbol{x}(t) - T)_i, \quad \forall i \tag{25}$$

where $R : [0, 1] \to \mathbb{R}_+$ is any $\mathcal{C}^1$ function, satisfying $R(0) = R(1) = 0$.
Let $\boldsymbol{x}(0) \in \text{int}(\mathcal{H})$, where $\mathcal{H}$ is the N-dimensional unit hypercube $[0, 1]^N$. Then the system (25) admits a unique solution $\boldsymbol{x}(t) \in \mathcal{C}^1$ such that $\boldsymbol{x}(t) \in \mathcal{H}, \forall t \geq 0$.

Proof: See Appendix B. of [22]. ■

B. Avoiding Locally Uncontrollable Overload

Having established the feasibility and soundness of the control-law (24), we return to the original problem of locally uncontrollable overload, described in section III-B. In the following, we derive sufficient conditions for the correlation matrix $\boldsymbol{C}$ and the external arrival rate $\boldsymbol{A}$, for which the system is stable, in the sense that no locally uncontrollable overload situation takes place.

Characterization of the Stability Region

For a fixed correlation matrix $\boldsymbol{C}$, we show that if the arrival rate vector $\boldsymbol{A}$ lies within a certain polytope $\boldsymbol{\Pi_C}$, the system is stable in the above sense, under the action of the greedy load-management heuristic. The formal derivation of the result is provided in Appendix D. of [22], which involves linearization of the ODE (24) around certain fixed points. Here we outline a simple and intuitive derivation of the stability region $\boldsymbol{\Pi_C}$. We proceed by contradiction. Suppose that node i is facing a locally uncontrollable overload at time t. Hence, by definition, the following two conditions must be satisfied at node i

$$S_i(\infty) - T_i > 0, \text{ and } x_i(\infty) = 0 \tag{26}$$

Here Eqn. (26) denotes the fact that FastRoute node i is *overloaded*, i.e., the incoming traffic to node i's proxy is more than the capacity of the node i. Eqn. (26) denotes the fact that this overload is *locally uncontrollable*, since even after node

[3]Remember that $x_i(t)$'s, being probabilities, must satisfy $0 \leq x_i(t) \leq 1, \forall t, \forall i$

i's DNS-server has offloaded *all* incoming DNS-influenced arrivals to L_2 (the best that it can do with its local information), it is facing the overload situation. The above two equations imply that the following condition holds at the node i:

$$\sum_{j\neq i} C_{ji}A_j x_j(\infty) > T_i, \tag{27}$$

where we have used Eqn. (21) and the fact that $x_i(\infty) = 0$. Since $0 \leq x_j(\infty) \leq 1$, a necessary condition for uncontrollable overload (27) at node i is $\sum_{j\neq i} C_{ji}A_j > T_i$. Thus, if $\sum_{j\neq i} C_{ji}A_j \leq T_i$, then the locally uncontrollable overload is avoided at the node i by the greedy heuristic. Taking into account all nodes, we see that if the external DNS-query arrival rate $\boldsymbol{A}$ lies in the polytope $\boldsymbol{\Pi_C}$ defined as

$$\boldsymbol{\Pi_C} = \{\boldsymbol{A} \geq \boldsymbol{0} : \sum_{j\neq i} C_{ji}A_j \leq T_i,\ \ \forall i = 1, 2, \ldots, N\} \tag{28}$$

then the locally uncontrollable overload situation is avoided at *every* node and the system is stable. Somewhat surprisingly, by exploiting the exact form of the control-law (24), we also show that a two-node system (as depicted in Figure 2) is able to control DNS-load $\boldsymbol{A}$ of any magnitude, under certain favorable conditions on the correlation matrix $\boldsymbol{C}$.

Special Case: [**Two-node System**]

Consider a two-node CDN discussed earlier in Section III (see Figure 2). Let the correlation matrix $\boldsymbol{C}$ for the system be parametrized as follows:

$$\boldsymbol{C}(\alpha,\beta) = \begin{pmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{pmatrix} \tag{29}$$

Then we have the following theorem :

Theorem 4.2: 1) The system does not possess *any* periodic orbit for *any* values of its defining parameters: $\boldsymbol{A}, \boldsymbol{C}(\alpha,\beta), \boldsymbol{T}$. Thus the system never oscillates.
2) If $\alpha > \frac{1}{2}$ and $\beta > \frac{1}{2}$ then the system is locally controllable (i.e., no locally uncontrollable overload) for *all* arrival rate-pairs (A_1, A_2).
3) If $\alpha < \frac{1}{2}$ and $\beta < \frac{1}{2}$ then a sufficient condition for local controllability of the system is $A_1 < \frac{T_1}{1-\alpha}$, $A_2 < \frac{T_2}{1-\beta}$.

Proof: The proof of part-(1) follows from Dulac's criterion [24], whereas proof for part-(2) and (3) follows from linearization arguments. See Appendix E. of [22] for details. ■

We emphasize that the part-(2) of the Theorem 4.2 is surprising, as it shows that the system remains locally controllable, no matter how large the incoming DNS-query arrival rate be (c.f. Section III-B) . The 2D vector-field plot in Figure 4 illustrates the above result. In Figure 4(a), the matrix $\boldsymbol{C}$ is taken to be one satisfying the condition of part (2) of lemma 4.2. As shown, all four phase-plane trajectories with different initial conditions converge to an interior fixed point $(x_1(\infty) > 0, x_2(\infty) > 0)$. For the purpose of comparison, in Figure 4(b) we plot the 2D vector-field of a locally uncontrollable system (e.g., the system in Fig. 2). It is observed that all four previous trajectories converge to the uncontrollable attractor $(x_1(\infty) = 1, x_2(\infty) = 0)$. From the vector-field plot, it is also intuitively clear why a periodic orbit can not exist in the system.

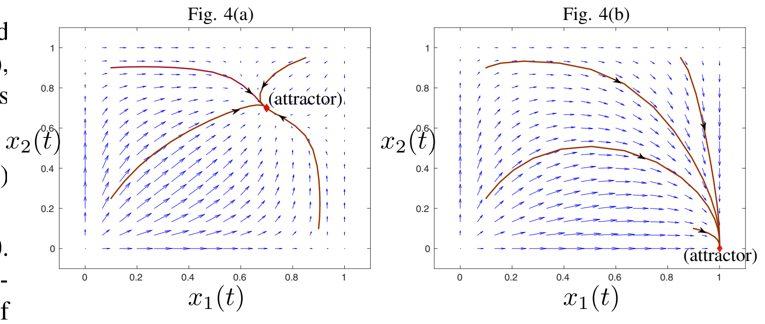


Fig. 4. 2D vector-fields of a two-node system illustrating locally controllable (Fig. 4(a)) and locally uncontrollable (Fig. 4(b)) overloads.

Application of Theorem 4.2: Consider a setting where, instead of making locally greedy offloading decisions, nodes are permitted to partially coordinate their actions. Also assume that there exists a partition of the set of nodes into two non-empty disjoint sets G_1 and G_2, such that their effective self-correlation values $\alpha(G_1)$ and $\beta(G_2)$, defined by

$$\alpha(G_1) = \frac{\sum_{i\in G_1}\sum_{j\in G_1} C_{ij}}{|G_1|}, \beta(G_2) = \frac{\sum_{i\in G_2}\sum_{j\in G_2} C_{ij}}{|G_2|}$$

satisfy the condition (2) of Theorem 4.2, i.e. $\alpha(G_1) > \frac{1}{2}, \beta(G_2) > \frac{1}{2}$. Then if the nodes in the sets G_1 and G_2 coordinate and jointly implement the greedy policy, then the system is locally controllable for all symmetric arrivals.

V. Numerical Evaluations

We use the operational FastRoute CDN to identify relevant system parameters for critical evaluations of the optimal algorithm and the heuristic. Currently, FastRoute has many operational nodes, spreading throughout the world [1]. The inter-node correlation matrix $\boldsymbol{C}$ is computed using system-traces spanning over three months.

For our performance evaluations, we use the cost-functions given in Eqns. (7) and (8). Different (normalized) parameters appearing in the cost-functions are chosen as follows

$$\gamma_i = 10, \eta_i = 1, T_i = 0.7, d_i \sim U_i[0,1] \qquad \forall i \tag{30}$$

where $U_i[0,1]$'s denote i.i.d. uniformly distributed random variables in the range $[0,1]$. The DNS-query arrival rates A_i's are assumed to be i.i.d. distributed according to a Poisson variable with mean $\bar{A}$, which varies across the range $[0.1, 10]$. The optimal solutions of the Lagrangian relaxations in Eqns. (13) for the above cost-functions can be obtained in closed form as follows :

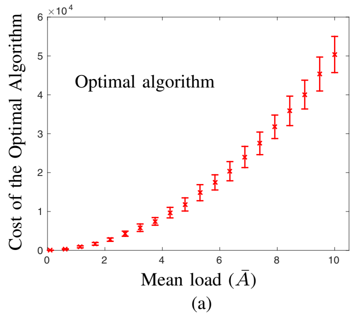
$$S_i^*(\boldsymbol{\mu}) = T_i \max\left(0, 1-\sqrt{\frac{\eta_i}{\mu_i}}\right) \tag{31}$$

and,

$$x_i^*(\boldsymbol{\mu}) = \begin{cases} 1, & \text{if } c_{2i} > 0 \\ 1 + \frac{c_{2i}}{2c_{1i}}, & \text{if } c_{2i} \leq 0 \text{ and } 2c_{1i} \geq -c_{2i} \\ 0, & \text{o.w.} \end{cases} \tag{32}$$

where $c_{1i} \equiv A_i\gamma_i$ and $c_{2i} \equiv \gamma_i d_i - \beta_i(\boldsymbol{\mu})$.

For each value of $\bar{A}$, we run the simulation $N_E = 100$



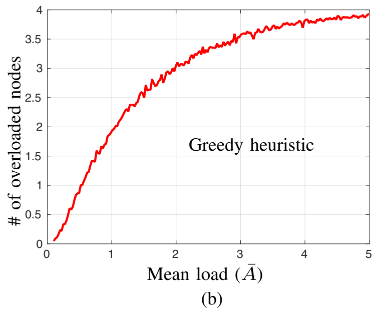


Fig. 5. (a) Statistical variation of the cost of the optimal algorithm with mean load (DNS-query arrival rate) $\bar{A}$. (b) Average number of nodes undergoing uncontrollable overload condition under the action of the greedy algorithm (Threshold $T_i = 0.7$ for all nodes).

times, by randomizing over both A_i and $d_i, \forall i$. The mean and standard-deviation of the resulting optimal cost values are plotted in the Figure 5(a). As expected, the average cost increases as the overall DNS-query arrivals to the system increases. However, the resulting cost remains finite always. This implies that *none* of the proxies are overloaded.

The above observation is in sharp contrast with the situation using the greedy-heuristic, the subject of Figure 5(b). Here we plot the number of overloaded proxies for different values of $\bar{A}$, keeping all other system-parameters the same. While the greedy algorithm does yield acceptable result for small values of $\bar{A} << T_i = 0.7$, we see that as many as four proxies undergo locally uncontrollable overload for relatively large values of $\bar{A}$.

Thus, depending on the computed correlation-matrix $\boldsymbol{C}$ and a projected bound of the DNS-query arrival rate $\boldsymbol{A}$, we can make an informed decision about the choice of the algorithms to employ in a CDN and the inherent complexity-vs-optimality trade-off it entails.

VI. Conclusion

In this paper we formalize the load management problem in modern CDNs utilizing anycast. We first formulate the problem as a convex optimization problem and study its dual and an associated algorithm. The novel idea of FastControl packets facilitates distributed implementation of the proposed dual algorithm. Next we analyze stability properties of a greedy heuristic, currently in operation in a major commercial CDN. We find that the optimal algorithm significantly out-performs the heuristic for moderate-to-high value of system load. However, the heuristic performs satisfactorily given that the system is *loosely-coupled* and the offered load is low. Thus an informed choice between these two algorithms may be made depending on the range of system-parameters and the desired optimality/complexity trade-off for a particular CDN. Future work would involve investigating the amount of FastControl packets necessary for the dual algorithm to work in the presence of random packet-loss and delayed feedback. Also, It would be interesting to generalize the findings of Theorem 4.2 for the case of more than two nodes.

References

[1] "Fastroute: A scalable load-aware Anycast routing architecture for modern CDNs," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/flavel

[2] R. Engel, V. Peris, D. Saha, E. Basturk, and R. Haas, "Using IP anycast for load distribution and server location," in *Proc. of IEEE Globecom Global Internet Mini Conference*. Citeseer, 1998, pp. 27–35.

[3] E. S.-J. Swildens, Z. Liu, and R. D. Day, "Global traffic management system using IP anycast routing and dynamic load-balancing," Aug. 11 2009, uS Patent 7,574,499.

[4] W. T. Zaumen, S. Vutukury, and J. Garcia-Luna-Aceves, "Load-balanced anycast routing in computer networks," in *Computers and Communications, 2000. Proceedings. ISCC 2000. Fifth IEEE Symposium on*. IEEE, 2000, pp. 566–574.

[5] E. Basturk, R. Engel, R. Haas, V. Peris, and D. Saha, "Using network layer anycast for load distribution in the Internet," in *Tech. Rep., IBM TJ Watson Research Center*. Citeseer, 1997.

[6] S. Sarat, V. Pappas, and A. Terzis, "On the use of anycast in DNS," in *Computer Communications and Networks, 2006. ICCCN 2006. Proceedings. 15th International Conference on*. IEEE, 2006, pp. 71–78.

[7] H. Miura, "Server selection policy in active anycast," 2001.

[8] H. B. Hashim and J.-l. A. Manan, "An active anycast RTT-based server selection technique," in *Networks, 2005. Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication., 2005 13th IEEE International Conference on*, vol. 1. IEEE, 2005, pp. 5–pp.

[9] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van der Merwe, "Anycast CDNs revisited," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008.

[10] M. Jaseemuddin, A. Nanthakumaran, and A. Leon-Garcia, "TE-friendly content delivery request routing in a CDN," in *Communications, 2006. ICC'06. IEEE International Conference on*, vol. 1. IEEE, 2006, pp. 323–330.

[11] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan, "On the responsiveness of DNS-based network control," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 21–26.

[12] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of DNS-based server selection," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2001, pp. 1801–1810.

[13] "Microsoft azure," http://azure.microsoft.com/en-us/.

[14] S. Yu, W. Zhou, and Y. Wu, "Research on network anycast," in *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*. IEEE, 2002, pp. 154–161.

[15] D. P. Bertsekas and R. G. Gallager, *Data networks*. Prentice-hall, 1987.

[16] T. Roughgarden, *Selfish routing and the price of anarchy*. MIT press Cambridge, 2005, vol. 174.

[17] D. P. Bertsekas, *Nonlinear programming*. Athena scientific Belmont, 1999.

[18] F. Kelly, "Charging and rate control for elastic traffic," *European transactions on Telecommunications*, vol. 8, no. 1, pp. 33–37, 1997.

[19] I. Lobel and A. Ozdaglar, "Distributed subgradient methods for convex optimization over random networks," *Automatic Control, IEEE Transactions on*, vol. 56, no. 6, pp. 1291–1306, 2011.

[20] A. Eryilmaz, A. Ozdaglar, D. Shah, and E. Modiano, "Distributed cross-layer algorithms for the optimal control of multihop wireless networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 18, no. 2, pp. 638–651, 2010.

[21] A. Nedic and A. Ozdaglar, "Convex optimization in signal processing and communications, chapter cooperative distributed multi-agent optimization. eds., eldar, y. and palomar, d," 2008.

[22] A. Sinha, P. Mani, J. Liu, A. Flavel, and D. A. Maltz, "Distributed Load Management in Anycast-based CDNs," Tech. Rep. [Online]. Available: http://arxiv.org/abs/1509.08194

[23] D. Bertsekas, "Convex Optimization Algorithms," *Athena Scientific, United States*, 2015.

[24] S. H. Strogatz, *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. Westview press, 2014.