Distributed Load Management Algorithms in Anycast-based CDNs[☆]

Abhishek Sinha[a,*], Pradeepkumar Mani[b], Jie Liu[d], Ashley Flavel[c,**], Dave Maltz[b]

[a]*Massachusetts Institute of Technology, Cambridge, MA*
[b]*Microsoft Inc., Redmond, WA*
[c]*Salesforce.com, Redmond, WA*
[d]*Microsoft Research, Redmond, WA*

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 services consists of multiple layers of proxy nodes for service and co-located DNS-servers for load-balancing among different proxies. Both the proxies and the DNS-servers use anycast addressing, which offers the simplicity of design and high availability of service at the cost of partial loss of routing control. Due to the very nature of anycast, redirection actions by a DNS-server also affect load at nearby proxies in the network. This makes the problem of optimal distributed load management highly challenging. In this paper, we propose and evaluate 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 Kelly-type dual decomposition technique, we propose a fully distributed load management algorithm by introducing a new type of control packets, called *FastControl* packets. This algorithm utilizes the underlying anycast mechanism to enable effective coordination among the nodes, thus obviating the need for any external control channel. In the second half of the paper, we

[☆]A preliminary version of the paper appeared in 53rd Annual Allerton Conference on Communication, Control and Computing 2015.

[*]Corresponding Author

[**]This work was done when the author was an employee at Microsoft, Redmond.

Email addresses: sinhaa@mit.edu (Abhishek Sinha), prmani@microsoft.com (Pradeepkumar Mani), Jie.Liu@microsoft.com (Jie Liu), aflavel@salesforce.com (Ashley Flavel), dmaltz@microsoft.com (Dave Maltz)

examine an alternative greedy load management heuristic, currently in production in a major commercial CDN. We study its dynamical characteristics and analyze its operational and stability properties. Finally, we critically evaluate both the algorithms and explore their optimality-vs-complexity trade-off using trace-driven simulations.

Keywords: Performance Analysis; Decentralized and Distributed Control; Optimization

---

1. Introduction

With the advent of ubiquitous computing and web access, the last decade has witnessed an unprecedented growth in Internet traffic. Popular websites, such as `bing.com`, registers more than a billion hits per day, which need to be processed efficiently in an online fashion, with as little latency as possible. Content Distribution Networks (CDN) are *de facto* architectures to transparently reduce latency between the end-users and the web-services. In CDNs, a collection of non-origin servers (also known as *proxies*) attempt to offload requests from the main servers located in the data centers, by delivering cached contents on their behalf [1]. Popular examples of internet services using CDN include web objects, live-streaming media, emails and social networks [2]. Edge servers with cached contents serve as proxies to intercept some user requests and return contents without a round-trip to the data centers. Online routing of user requests to the optimal proxies remains a fundamental challenge in managing modern CDNs. Routing to a remote proxy may introduce extra round-trip delay, whereas routing to an overloaded proxy may cause the request to be dropped. Hence, one needs an efficient load balancing policy to utilize the resources optimally.
There are two major paradigms of load balancing algorithms in use today for cloud systems and CDNs:

- (1) DNS-based load balancing [3],[4], in which the users are directly routed to a server by using its *unique* IP-address.
- (2) Anycast-based load balancing [5],[6], in which internet routing protocols, such as BGP, implicitly route the users to a server among a group of servers sharing the same IP-address.

In the first category, the server loading information is monitored centrally and users are redirected to a lightly loaded proxy. Although this architecture has the

advantage of having a fine-grained control over the incoming load to each proxy, it requires *centralized* coordination among the nodes so that a user can be directed to the least-congested proxy. Since proxies are geo-distributed throughout the globe, it is difficult to implement a centralized load balancing scheme in real time due to the high amount of control information exchange involved. Thus, DNS-based load redirection mechanisms warrant considerable investments in infrastructures [7].

To get around this issue, in this paper we consider load balancing algorithms belonging to the second category, i.e., Anycast-based load balancing. Our point of focus will be *FastRoute* - a fully distributed state-of-the-art CDN belonging to *Microsoft Azure* [8]. This large-scale commercial strength CDN architecture alleviates the need for global coordination among the nodes, which makes it fast and robust. It uses a greedy load balancing heuristic that we will describe subsequently.

Anycast-based load balancing is widespread in modern CDNs [9]. In addition to *Microsoft Azure*, other prominent CDNs using this technology include *Amazon AWS* [10], and *Google Cloud CDN* [6]. With anycast, multiple proxy servers, having the same user-content, 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 [11]. 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 is required for its use [12].

Although anycast routing can simplify the system design and provide a high level of service availability to the end-users [13], it comes at the cost of partial loss of routing control. This is because a user request may be routed to any one of the multiple Geo-replicated proxies having the same anycast address. Since the user-to-proxy routing is done by the routing protocols and is not under the control of the CDN operator, the user request may end up in an already overloaded proxy, deteriorating its loading condition further. Figure 1 illustrates the problem of load management with Anycast.

There have been several attempts in the literature to cope up with the lack of load-awareness issue with network layer anycast. Papers [14], [15] 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.* [16] formulate the anycast load management problem as a General Assignment Problem, which is NP-hard. Jaseemuddin et al. [17] propose a new CDN architecture which balances server load and network latency

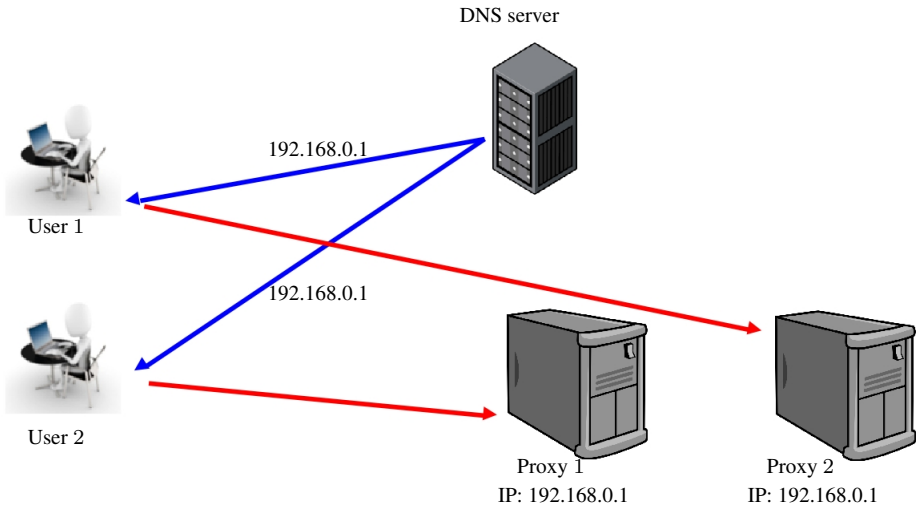


Figure 1: An example of Anycast-enabled CDN. User 1 and 2 obtain the anycast IP address from the DNS server (blue dotted arrow) and are then routed over the Internet *either* to Proxy 1 or to Proxy 2 for service (red solid arrow). Note that both proxies have the same IP address (Anycast addressing). The fraction of users landing on a given proxy is dictated by the Internet routing protocols and is beyond the control of CDN operators. This lack of load-awareness often leads to overload of certain proxies.

via detailed traffic engineering.

In the FastRoute architecture, the proxies are logically arranged in layers of anycast rings, with each layer having a distinct Anycast IP address. See Figure 3 for an example. The provisioned capacities of the proxies increase as we move from the outer layers to the inner layers of the anycast rings. DNS is responsible for moving the load across different layers by intelligently responding to users with different anycast addresses. Each proxy is equipped with a co-located authoritative DNS server. We will collectively refer to a proxy and its co-located DNS server simply as a *node*. See Figure 2 and Figure 3 for a high-level overview of the architecture.

An *overload* is said to occur when any proxy receives more service requests than its processing capacity. Since DNS is the primary control knob in this architecture, each DNS server at a node is responsible for redirecting traffic to other layers to alleviate overload in the co-located proxy. A fundamental problem with this approach is that not all users, that hit a given proxy, can be redirected successfully by the co-located DNS. This is because, due to anycast routing, DNS path and the data-flow paths are independent. Hence, a user's Local DNS (LDNS) could be obtaining a DNS response from some authoritative DNS server in a node,

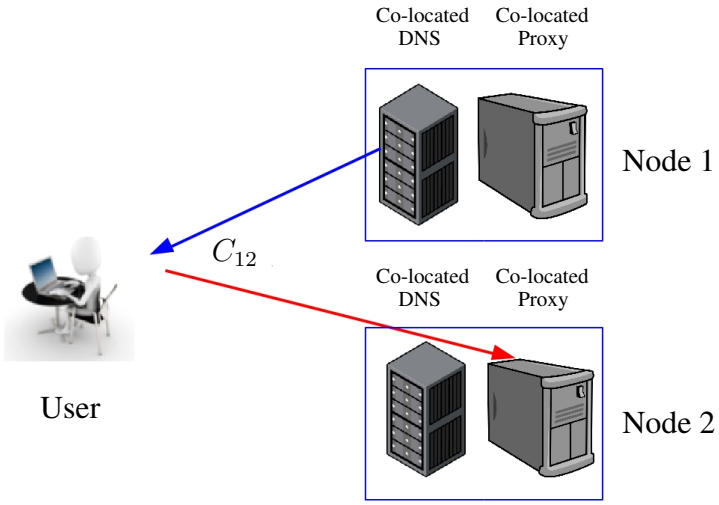


Figure 2: Two *FastRoute* CDN nodes with their corresponding co-located DNS and Proxy servers. Note that, although the user gets served by the proxy at Node 2, it obtains the anycast IP address from the DNS server at Node 1. Thus, in the event of an overload, this user cannot be redirected from node 2 by altering the DNS response from the co-located DNS server at node 2.

which is different from the DNS server co-located with the proxy which the user hits. An example is shown in Figure 2. Hence, intuitively, the ability for a DNS server at a node to control overload at the corresponding co-located proxy depends on the fraction of oncoming traffic to the proxy that is routed by the co-located DNS server to the node itself. Informally, we refer to the above quantity as the *self-correlation* [8] of a given node. A formal definition of *self-correlation* in this context will be given in Section 2.

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 overloaded proxies to the next layer. This control-decision could be based on a 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 they are needed. Thus a centralized solution is not practically feasible and the challenge is to design a provably optimal, yet completely distributed load management algorithm.

A major drawback of the current FastRoute architecture is that it uses a heuristic offloading policy with no optimality guarantees. In particular, it has been

reported in [9] that this heuristic often (for $\approx 20\%$ of the users) leads to an uncontrollable-overload situation, which needs expensive manual intervention to recover from. Also, from our numerical simulation results in Figure (9), we will see that even in the underloaded condition the delay performance of FastRoute is far from the optimal.
Our key contributions in this paper are as follows:

- In section 2, we present a simplified mathematical model for DNS-controlled load management in modern anycast-based 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 its operational characteristics.

- In section 3, we formulate the load management problem as a convex optimization problem and propose a Kelly-type *dual* algorithm [18] to solve it in a distributed fashion. The key to our distributed implementation is the *Lagrangian decomposition* and the use of *FastControl* packets, which exploit the underlying anycast architecture to enable 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 CDNs.

- In section 4, we consider an existing heuristic load management algorithm used in *FastRoute*, Microsoft's CDN for many first party websites and services it owns [19]. We model the dynamics of this heuristic using nonlinear system theory and derive its operational characteristics, which conform with the qualitative observations. 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 load management algorithm is able to control an incoming load of *any* magnitude. Unfortunately, this theoretical guarantee no longer holds once this correlation property is no longer in effect. In this case, the dual algorithm proposed in section 3 performs better than the heuristic.

- In section 5, we critically evaluate relative benefits of the proposed optimal and the heuristic algorithms through extensive numerical simulations. Our simulation is trace-driven in the sense that we use real correlation parameters collected over months from Microsoft Azure CDN [8].

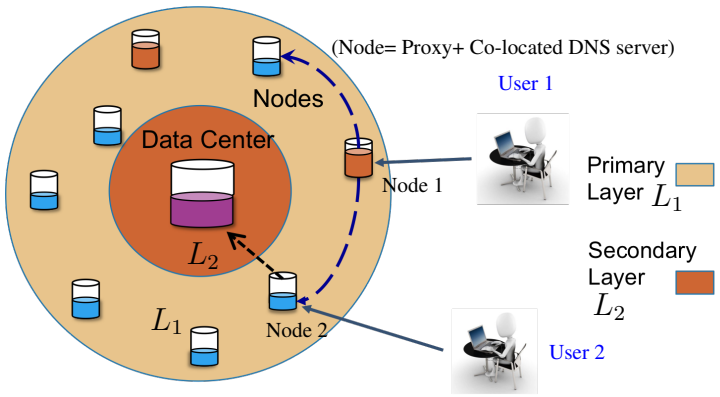


Figure 3: A CDN architecture with two anycast layers. All nodes in layer 1 have the same anycast IP-address. Solid arrows denote the DNS path of the users. The DNS of node 1 returns the anycast address of L_1 to the user 1. As a result of anycast routing, the user 1 may get redirected to some other node in L_1 for service (shown by the dashed arrows). The DNS of node 2 returns the anycast address for L_2 to the user 2. Thus, the user 2 is redirected to the data center.

2. System Model

Nodes and Layers: We consider an anycast-based CDN system consisting of two logical anycast layers: *primary* (also referred to as L_1) and *secondary* (also referred to as L_2). See Figure 3 for the overall architecture. The primary layer (L_1) hosts a total of N *nodes*, distributed worldwide. Each node consists of a co-located DNS and a proxy server. See Figure 2 for a schematic diagram. The proxies cache Web objects for faster delivery to the users. The i^{th} proxy has a (finite) capacity of serving T_i requests per unit time. The secondary layer (L_2) contains a data center with practically infinite processing capacity. We assume that both the proxy servers and the data center have the capability of serving the requested objects (i.e., no cache misses).

For popular online services, such as Microsoft's *Bing*, the nodes in L_1 are strategically placed closer to the user base. As a result, the user-to-proxy round trip latency is typically small. See [20] for optimal proxy placements and related algorithms. On the other hand, as the users are spread worldwide, the average geographical distance between the user base and the data center for L_2 is typically large. This results in significantly high user-to-data center round trip latency. The overall response time for a user is the sum of the latency and the processing time on the server (either proxy or the data center). Our objective is to devise a redirection mechanism for the user requests (explained below) that minimizes the overall response time.

Anycast Addressing: As discussed earlier, *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 addressing scheme of multiple proxies sharing the same IP-addresses is known as *anycast* addressing, which is widely in use for geo-replicated cloud services [21], [16].

Control actions: When a DNS request submitted by a user [1] arrives at a node i in L_1, requesting for the IP-address of a web-server, the co-located DNS server at node i may take *one* of the following two actions:

- **(1)** It returns the address $\mathcal{I}_2$ (which redirects the request to the data center in L_2). Or,
- **(2)** It returns the anycast address $\mathcal{I}_1$ (which redirects the request to *some* proxy in the primary layer L_1).

This (possibly randomized) **binary action** by the co-located DNS-server at node i may depend on the following local variables available at node i :

1. ***DNS-influenced request* arrival rate A_i at node i's DNS-server:** [2]. Each DNS request accounts for a certain amount of traffic load which is routed to either L_1 or L_2 according to node i's control actions. We normalize A_i so that each unit of A_i corresponds to a unit of load. As an example, if 10% of DNS responses at node i returns the address $\mathcal{I}_2$, then the total amount of load shifted to the data center (L_2) due to node i's DNS is $0.1A_i$.
2. ***User load* arrival rate $S_i(t)$ at node i's proxy:** This quantity is evidently a function of the arrival rates of DNS queries and redirection actions of other nodes (quantified later in Eqn.(2)). The current load at node i, i.e., $S_i(t)$, is locally known at the node i. The node i may incorporate this knowledge for devising appropriate control actions.

Anycasting and inter-node coupling: When a DNS request arrives at the node i and the co-located DNS server responds with the Layer-I anycast-address $\mathcal{I}_1$, the corresponding request may be routed to anyone of the N proxies (since all of them have the same IP-address $\mathcal{I}_1$). The particular proxy, where the request

[1] In the Internet DNS requests are actually generated by the Local DNS (LDNS) of the user and are recursively routed to an authoritative DNS server. However, to keep the model and the analysis simple, we will assume that individual DNS queries are submitted by the users themselves.

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

actually gets routed to, depends on the corresponding ISP's routing policy, current network congestions and many other time-varying factors. We assume that a typical query, which is routed to L_1 by the node i, is routed to node j's proxy with probability C_{ij}. Clearly,

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

The correlation matrix $\boldsymbol{C}$ may be determined empirically by setting up an experiment similar to the one described in [22].

In the ideal case where the nodes have perfect self-correlations, we have $\boldsymbol{C} \approx \boldsymbol{I}$, where $\boldsymbol{I}$ is the $N \times N$ identity matrix. In this case, the control decisions of the nodes do not interfere with each other and situations like Figure 2 rarely takes place. However, as reported in [8], in real CDN systems with ever increasing number of nodes, the system is far from ideal and the inter-node correlations are rather significant. In this paper, we theoretically investigate the consequences that arise from non-negligible inter-node correlations and design an optimal control strategy for mitigating overload.

System Equations: Assume that, due to the action of some control strategy π, the co-located DNS at node i randomly redirects $1 - x_i^\pi(t)$ fraction of incoming DNS queries 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 load arrival rate, $S_i(t)$, at node i's co-located proxy may be written as

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

Definition 2.1 (Local Control). *A control strategy π for DNS redirection is called* local *if it can be represented by a collection of control maps $\boldsymbol{\pi} = \left(x_i^\pi(\cdot), i = 1, 2, \ldots, N \right)$, with $\boldsymbol{x}_i^\pi : \Omega_i^t \times t \to [0, 1]$ where Ω_i^t is the set of all* local observables *at node i up to time t.*

Formally, our objective is to design an efficient local control strategy to avoid overload in the network.

3. An Optimization Framework

3.1. Motivation

The central objective of a load management policy in our setting is to redirect the minimal amount of traffic to the Layer 2 (due to high round-trip latency), without overloading the primary layer L_1 proxies (due to their limited capacities). Clearly, these two objectives are in conflict with each other and we need to find a suitable operating point. The added difficulty, which makes the problem fundamentally challenging is that each node is an autonomous agent and takes its redirection decisions based on its **local observables only**. As an example, a simple greedy heuristic for node i would be to redirect requests to L_2 (i.e. decrease $x_i(t)$) whenever its co-located proxy is overloaded (i.e., $S_i(t) > T_i$) and redirect requests to L_1 (i.e., increase $x_i(t)$) otherwise. This policy forms the basis of the greedy control strategy proposed in [8].
Although this simple greedy strategy seems to be appealing for large-scale deployment, we next show that, with significant cross-correlations among the nodes, this heuristic could lead to an undesirable *uncontrollable overload situation*, with a significant performance loss. This motivates us to design a more effective distributed load management algorithm, which we undertake later.

3.2. Locally Uncontrollable Overload: An Example

Consider the two-layered CDN in Figure 4, which hosts only two nodes a and b in the primary layer. DNS request arrival rates to the nodes a and b are $A_a = 1$ and $A_b = 1$. Suppose the processing capacities (also referred to as *thresholds*) of the co-located proxies are $T_a = 0.7$ and $T_b = 0.7$ respectively. Using the correlation parameters $[C_{ij}]$ as shown in Figure 4, the loads at the co-located proxies can be found as follows:

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

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

Since $0 \leq x_a(t), x_b(t) \leq 1$, it is clear that under *any* control policy $\boldsymbol{x}(t)$ the following holds

$$S_a(t) \leq 0.1 \times 1 + 0.5 \times 1 = 0.6 < 0.7 = T_a, \ \forall t \geq 0$$

Thus, the proxy at node a will be under loaded irrespective of the load management policy π in use. Consequently, under the greedy heuristic the co-located

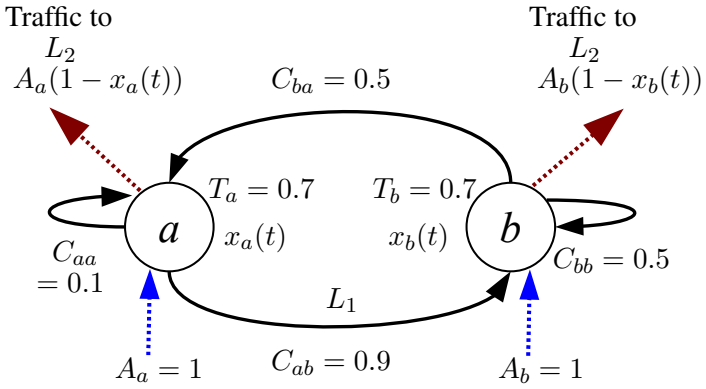


Figure 4: A two-node system illustrating locally uncontrollable overload at the node b

DNS-server at node a will steadily 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 autonomously as it does not have node b's loading information). This, in turn, overloads the co-located proxy in node b because the steady state user 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 redirecting the incoming DNS queries to the secondary layer, 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 request arrival patterns. Also, this conclusion does not depend on the detailed control mechanism of the offload probabilities (viz. instantaneous values if $\dot{x}_1(t)$ and $\dot{x}_2(t)$). Since the overload condition at the node b cannot be overcome by the isolated autonomous action of node b itself, we say that node b is facing a *locally uncontrollable overload situation*.

From the point-of-view of the entire system, the above situation is extremely inefficient, because a large fraction (45% in the above example) of the incoming requests either gets dropped or severely delayed due to the overloaded node b. This poor operating point could have been potentially avoided if the nodes could coordinate their actions, instead of acting greedily[3]. It is not difficult to realize that

[3]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

the principal reasons responsible for 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 which warrants coordination among the nodes. In our proposed algorithm [2] we address this issue by introducing the novel idea of *FastControl* packets. This strategy does not require any explicit state or control information exchange and uses the anycast mechanism to its advantage.
Regarding the factor (2), we intuitively expect that the greedy heuristic should work well if the self-correlations of the nodes (i.e. C_{ii}) are not too small. In this favorable case, each node controls a major fraction of the load coming to it. In section 4, we return to a variant of the local heuristic used in FastRoute [8] and derive analytical conditions under which the above intuition holds good.
In the following section, we take a principled approach and design an iterative load management algorithm, which is optimal for arbitrary system parameters $(\boldsymbol{A}, \boldsymbol{C})$. It will be shown that it is enough for each node i to know its own local DNS and user load arrival rates (i.e., A_i and $S_i(t)$ respectively) and the entries corresponding to the i^{th} row and column of the correlation matrix $\boldsymbol{C}$ (i.e., $C_{i\cdot}, C_{\cdot i}$), to achieve the optimal operating point. No non-local knowledge of the dynamic loading conditions of other nodes is required for implementing the algorithm.

3.3. Mathematical formulation

Consider the following optimization problem:

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

Subject to,

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

proxies in the Primary layer L_1 at all.

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

Discussion. The first component of the cost function, $g_i(S_i)$, denotes the cost for serving S_i amount of user requests by the i^{th} proxy in Layer I per unit time. Clearly, this cost component grows rapidly once the load in a proxy is close to its processing capacity, i.e. $S_i \approx T_i$ and it becomes infinite when the proxy is overloaded. As an example, the cost $g_i(\cdot)$ may be taken to be proportional to the average aggregate queuing delay for an $M/G/1$ queue with capacity T_i [23] given as follows:

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

Here η_i is a positive constant, denoting the relative cost per unit increment in latency.
The second component of the cost function $h_i(x_i)$ denotes the cost due to *round-trip latency* of requests routed to the Data center (L_2). As an example, in a popular model [24], 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 $A_i(1-x_i)$, according to this model, the cost function $h_i(x_i)$ may be taken as follows

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

where d_i is the propagation delay from the node i to the data center and θ_i and γ_i are suitable positive constants. A typical plot of the cost-surface for the case of a two-node system as in Figure 4 is shown in Figure 5.

The constraint set $X = [0,1]^N$ represents the N-dimensional unit hypercube in which the (controlled) redirection probabilities must lie. The set Σ_T captures the capacity constraints of the proxies, e.g., if the proxy i has capacity T_i then we have

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

For technical reasons, the functions $g_i(\cdot), h_i(\cdot)$ are assumed to be closed, proper and strictly convex [25]. We also assume the functions $g_i(\cdot)$ to be monotonically

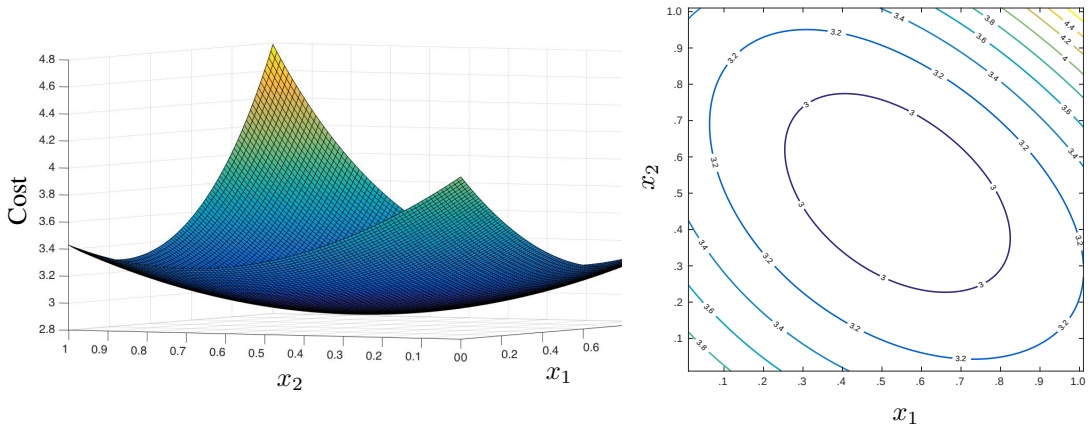


Figure 5: (a) A typical plot of cost-surface for a two-node system as a function of their redirection probabilities $x_1, x_2 (0 \leq x_1, x_2 \leq 1)$ and (b) the associated level-sets of the cost function.

increasing. As a result, we can replace the equality constraint (6) by the following inequality constraint, without loss of optimality

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

The reason being if the optimal S_i^* is strictly greater than the LHS in (9), we can strictly (and feasibly) reduce the objective value by reducing S_i^* to the level of LHS in (9). This leads to a contradiction.
Thus, the above load management problem is equivalent to the following optimization problem $\mathbf{P}_1$:

Minimize $W(\boldsymbol{x}, \boldsymbol{S}) = \sum_{i=1}^{N} \big(g_i(S_i) + h_i(x_i)\big)$
Subject to,

$$\sum_{j=1}^{N} C_{ji} A_j x_j \leq S_i, \quad \forall i = 1, 2, \ldots, N \tag{10}$$

$$\boldsymbol{x} \in X, \boldsymbol{S} \in \Sigma_{\boldsymbol{T}}, \tag{11}$$

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

Since with the above assumptions, the objective function and the constraint sets of the problem $\boldsymbol{P}_1$ are all convex [25], we immediately have the following lemma:

Lemma 3.1. *The problem $\boldsymbol{P}_1$ is convex.*

Hence a variety of optimization methods [26] may be used to solve the problem $\mathbf{P}_1$ in a centralized fashion. However, our objective is to find a distributed load balancing algorithm running at each node, which collectively solves the problem $\mathbf{P}_1$ with locally available loading information only. This problem is explored in the next subsection.

3.4. The Dual Decomposition Algorithm

In this section, we derive a dual algorithm [18], [27], [28] for the problem $\mathbf{P}_1$ and show how it leads to a distributed implementation with negligible control overhead.

By associating a non-negative dual variable μ_i to the i^{th} constraint in (10) for all i, the Lagrangian of $\mathbf{P}_1$ may be written as follows:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{S}, \boldsymbol{\mu}) = \sum_{i=1}^{N} \big(g_i(S_i) - \mu_i S_i)\big) + \sum_{i=1}^{N} \left(h_i(x_i) + A_i x_i \big(\sum_{j=1}^{N} \mu_j C_{ij} \big) \right) \quad (12)$$

This leads to the following dual objective function [25]

$$D(\boldsymbol{\mu}) = \inf_{\boldsymbol{x} \in X, \boldsymbol{S} \in \Sigma_{\boldsymbol{T}}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{S}, \boldsymbol{\mu}) \quad (13)$$

We now exploit the *separability* property of the dual objective (12) to reduce the problem (13) into following two one-dimensional sub-problems:

$$\left.\begin{aligned} S_i^*(\boldsymbol{\mu}) &= \inf_{0 \le S_i \le T_i} \left(g_i(S_i) - \mu_i S_i \right) \\ x_i^*(\boldsymbol{\mu}) &= \inf_{0 \le x_i \le 1} \left(h_i(x_i) + A_i \beta_i(\boldsymbol{\mu}) x_i \right), \end{aligned}\right\} \quad (14)$$

where the scalar $\beta_i(\boldsymbol{\mu})$ is defined as the (scaled) linear projection of the dual-vector $\boldsymbol{\mu}$ on the i^{th} row of the correlation matrix, i.e.

$$\beta_i(\boldsymbol{\mu}) = \sum_{j=1}^{N} \mu_j C_{ij} = \boldsymbol{C}_i^T \boldsymbol{\mu}, \tag{15}$$

and $\boldsymbol{C}_i$ is the i^{th} row of the correlation matrix $\boldsymbol{C}$.

Example:

The optimal solutions to one-dimensional problems in Eqns. (14), for the cost function given in (7) and (8), can be obtained in closed form as follows:

$$S_i^*(\boldsymbol{\mu}) = T_i \max\left(0, 1 - \sqrt{\frac{\eta_i}{\mu_i}}\right) \tag{16}$$

and,

$$x_i^*(\boldsymbol{\mu}) = \begin{cases} 1, & \text{if } c_{2i} > 0 \\ 1 + \frac{c_{2i}}{2c_{1i}}, & \text{if } c_{2i} \leq 0 \text{ and } 2c_{1i} \geq -c_{2i} \\ 0, & \text{o.w.} \end{cases} \tag{17}$$

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

Discussion. The scalar $\beta_i(\boldsymbol{\mu})$ couples the offload decision of node i with the state of the entire network through Eqn. (14). Once the value of $\beta_i(\boldsymbol{\mu})$ is available to the node i, the node has all the required information at its disposal to *locally* solve the corresponding sub-problems (14), for a fixed value of $\boldsymbol{\mu}$. The solutions of these one-dimensional sub-problems may even be obtained in closed form in some cases. Also, note that $\beta_i(\boldsymbol{\mu})$, being a scalar, is potentially easier to communicate than the entire N-dimensional vector $\boldsymbol{\mu}$. In sub-section 3.6, we exploit this fact and 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 is no duality gap [25]. Convex duality theory guarantees the existence of an optimal dual variable $\boldsymbol{\mu}^* \geq \mathbf{0}$ such that, the solution to the relaxed problem (14), corresponding to $\boldsymbol{\mu}^*$, gives the optimal solution to the original problem $\mathbf{P}_1$. To obtain the optimal dual variable $\boldsymbol{\mu}^*$, we solve dual $\mathbf{P}_1^*$ of the problem $\mathbf{P}_1$, given as follows

Problem $\mathbf{P}_1^*$:

$$\textbf{Maximize} \quad D(\boldsymbol{\mu}) \qquad (18)$$
$$\boldsymbol{\mu} \geq \mathbf{0}$$

The dual problem $\mathbf{P}_1^*$ is well-known to be convex [25]. To solve the problem $\mathbf{P}_1^*$, we use the **Projected Super-gradient** algorithm [29], 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 $\partial D(\boldsymbol{\mu}(k)) = \boldsymbol{S}^{\text{obs}}(k) - \boldsymbol{S}(k)$ [25], where $S_i^{\text{obs}}(k)$ is the observed rate of arrival of incoming user load at the proxy of node i, i.e.,

$$S_i^{\text{obs}}(k) \equiv \sum_{j=1}^{N} C_{ji} A_j x_j^*(k), \qquad (19)$$

and $x_i^*(k)$ and $S_i^*(k)$ are the primal variables obtained from Eqn. (14), evaluated at the current dual variable $\boldsymbol{\mu} = \boldsymbol{\mu}(k)$. Following a projected super-gradient step, the dual variables $\boldsymbol{\mu}(k)$ are iteratively updated component-wise at each node i as follows:

$$\mu_i(k+1) = \Big(\mu_i(k) + \alpha\big(S_i^{\text{obs}}(k) - S_i^*(k)\big)\Big)^+ \qquad (20)$$

Here α is a small positive step size constant, whose appropriate value will be given in Theorem (3.3). Since the system parameters might vary slowly over time, a time-invariant algorithm is practically preferable. Hence, we chose to use a constant step-size α, rather than a sequence of diminishing step-sizes $\{\alpha_k\}_{k\geq 1}$. The basic centralized version of the optimization algorithm is summarized in Algorithm 1. In the next section, we will derive a distributed version of the dual algorithm.

Algorithm 1 A Centralized Load Management Algorithm

1: *Maintain* Dual variables $\{\boldsymbol{\mu}(k)\}_{k\geq 1}$.
2: *Initialize:* $\boldsymbol{\mu}(1) \leftarrow \mathbf{0}$
3: **for** $k = 1, 2, 3, \ldots$ **do**
4: **for** each node i **do**
5: $\boldsymbol{\beta}_i(\boldsymbol{\mu}(k)) \leftarrow \sum_{j=1}^{N} C_{ij}\boldsymbol{\mu}_j(k)$
6: *Compute the current primal variables:*

$$\begin{aligned} S_i(k) &\leftarrow \inf_{0\leq S_i\leq T_i} \big(g_i(S_i) - \mu_i(k)S_i\big) \\ x_i(k) &\leftarrow \inf_{0\leq x_i\leq 1} \big(h_i(x_i) + A_i(k)\beta_i(k)x_i\big) \end{aligned}$$

7: *Compute the current load:*

$$S_i^{\text{obs}}(k) \leftarrow \sum_{j=1}^{N} C_{ji}A_j x_j(k) \tag{21}$$

8: *Update the dual variables:*

$$\mu_i(k+1) \leftarrow \big(\mu_i(k) + \alpha(S_i^{\text{obs}}(k) - S_i(k))\big)^+$$

9: **end for**
10: **end for**

3.5. Convergence of the Dual Algorithm

To show that the proposed dual algorithm converges to the optimal solution, we first analyze the super-gradients $\boldsymbol{g}(\boldsymbol{\mu}(k))$:

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

The following technical lemma is the key to the convergence result:

Lemma 3.2. *If the total external DNS-request arrival rate to the entire 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 $k \geq 1$

$$||\boldsymbol{g}(\boldsymbol{\mu}(k))||_2^2 \leq A_{\max}^2 + NT_{\max}^2 \tag{22}$$

PROOF. See Appendix 8.1.

Upon bounding the super-gradients, the convergence of the dual algorithm follows directly from Proposition 2.2.2 and 2.2.3 of [26]. In particular, we have the following theorem:

Theorem 3.3. *For a given* $\epsilon > 0$*, let the step-size* α *in Eqn.* (20) *be chosen as* $\alpha = \frac{2\epsilon}{A_{\max}^2 + NT_{\max}^2}$*. Then,*

- *The sequence of solutions, produced by the dual algorithm described above, converge within an* ϵ*-neighborhood the optimal objective value of the problem* $\boldsymbol{P}_1$*.*
- *The rate of convergence of the algorithm, around 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 more inter-node coordination to converge to the global optimum.

3.6. FASTCONTROL *Packets and Distributed Implementation*

We now examine the centralized algorithm provided in 1 in detail. We notice that the variable $S_i^{\text{obs}}(k)$ in step (7) is simply the currently observed load at node i and is known to the nodes locally. Hence, apart from step (5), which requires linearly combining the dual variables from different nodes, all other steps of the algorithm may be performed locally. Thus, if the value of the coupling factors $\beta_i(\boldsymbol{\mu}(k))$ are made available to node i, the dual algorithm can be implemented in a completely distributed fashion. To accomplish this, we now introduce the novel idea of FASTCONTROL packets. In brief, it exploits the underlying anycast

mechanism for *in-network, on-the-fly computation* of the coupling factor $\beta_i(\boldsymbol{\mu}(k))$ (Eqn. (15)) for all i.

General Descriptions. FASTCONTROL packets are special-purpose control packets (different from the regular data packets), each belonging to any one of N distinct classes, which we refer to as *categories*. The category of each FASTCONTROL packet is encoded in its packet header and hence it takes extra $\log(N)$ bits to encode the categories. Physically, these control packets originate from the users and are monitored by the proxies. However, the rates at which these packets are generated, are controlled by the DNS servers of the nodes i and are proportional to the dual variables $\mu_i(k)$, by a mechanism detailed below.

Generation of FastControl packets. 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 [8]). 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, the DNS server of each node i **forces** generation of FASTCONTROL packets of category j (through its response to DNS-queries) at the rate

$$r_{ij}(k) = \gamma\mu_i(k)\frac{C_{ji}}{C_{ij}}, \quad j = 1, 2, \ldots, N \tag{23}$$

  Note that this step is locally implementable at each node, since the value of the dual variable $\mu_i(k)$ is locally available at the 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 packets of category i, denoted by $R_i(k)$, at its co-located proxy. Using equation (23), the total rate of reception $R_i(k)$ of i^{th} category FASTCONTROL packets at node i is obtained as follows

$$\begin{aligned} R_i(k) &= \sum_{j=1}^{N} r_{ji}(k)C_{ji} = \sum_{j=1}^{N} \gamma\mu_j(k)\frac{C_{ij}}{C_{ji}}C_{ji} \\ &= \gamma\beta_i(\boldsymbol{\mu}(k)) \end{aligned}$$

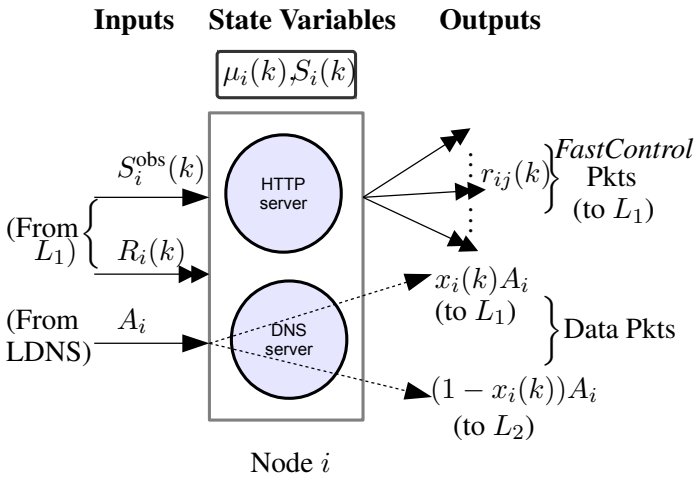


Figure 6: The i^{th} node implementing the dual algorithm

Thus,

$$\beta_i(\boldsymbol{\mu}(k)) = \frac{1}{\gamma} R_i(k) \tag{24}$$

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 co-located proxy. A complete pseudocode of the algorithm is provided below. See Figure (6) for a schematic diagram of a node implementing the algorithm.

Algorithm 2 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 the Primal variables:*

$$\begin{aligned} S_i(k) &\leftarrow \inf_{0 \le S_i \le T_i} \big(g_i(S_i) - \mu_i(k) S_i\big) \\ x_i(k) &\leftarrow \inf_{0 \le x_i \le 1} \big(h_i(x_i) + A_i(k)\beta_i(k) x_i\big) \end{aligned}$$

6: *Update the 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**

From the above description of the algorithm, we can make the following interesting observations:

- The full knowledge of the matrix $\boldsymbol{C}$ at node i is also not necessary. It is sufficient that node i knows the i^{th} row and column of the matrix $\boldsymbol{C}$.
- The optimality of the algorithm **does not depend** on the diagonal dominance property of the correlation matrix $\boldsymbol{C}$, an essential requirement for the greedy heuristic (discussed in the next section) to work reasonably well.
- The parameter γ in Eqn. (23) is directly related to the amount of control overhead required for the distributed algorithm.

This completes the description of the proposed load management algorithm, which is completely distributed and provably optimal. In the following section,

we concentrate our attention on the load management heuristic used in FastRoute and evaluate its performance. We will numerically compare the relative benefits of these two algorithms in Section 5.

4. The Greedy Load Management Heuristic

In this section, we focus exclusively on a greedy distributed load management heuristic, which is also implemented in FastRoute [8], a commercial two-layered CDN. This heuristic algorithm ignores inter-node correlations altogether and assumes that each node fully controls the oncoming traffic to it. Thus, when a proxy is overloaded, the co-located DNS server modifies its DNS response to redirect more traffic to the data center (L_2) and vice versa. This simple mechanism is reported to work well in practice only when there is a high correlation (60-80%) between the node receiving the DNS query and the node receiving the corresponding request. However, in the case of sudden bursts of traffic, e.g., *Flash Crowds* [30], this greedy heuristic results in an uncontrollable overload situation, warranting manual intervention [8].

Algorithmic challenges: Equipped with only local information at its disposal, the DNS server faces the following dilemma: offload too little to the data center and the co-located proxy, if overloaded, remains overloaded; offload too much to the data center and the users are unnecessarily directed to the remote data center and receive a delayed response, due to its high round-trip latency. The coupling among the nodes due to inter-node correlation renders this problem highly challenging and gives rise to the so-called uncontrollable overload, discussed earlier in section 3.2.

A pseudo-code for FastRoute's heuristic algorithm is provided below. An explicit control mechanism modeling the heuristic will be given in section 4.1.

Algorithm 3 Decentralized greedy load management heuristic used in *FastRoute* [8], running at the node i

```
for t = 1, 2, 3, ... do
    if the i^th proxy is under loaded (S_i(t) ≤ T_i) then
        increase x_i(t) proportional to −(S_i(t) − T_i)
    else
        decrease x_i(t) proportional to (S_i(t) − T_i)
    end if
end for
```

Motivation for analyzing the heuristic: The optimal algorithm of section 3 requires the knowledge of the correlation matrix $\boldsymbol{C}$ and needs to utilize additional FASTCONTROL packets for its operation. In this section, we analyze the performance of the greedy heuristic, currently implemented in *FastRoute* [8], which does not have these requirements. Since this heuristic completely ignores the inter-node correlations (given by the matrix $\boldsymbol{C}$), it cannot be expected to achieve the optimum 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 face an uncontrollable overload condition (refer to section 3.2) under its action. Depending on target applications, this metric is often practically good enough for gauging the performance of CDN systems.

4.1. 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 rate of incoming load to the proxy i at time t is given by,

$$S_i(t) = \sum_{j=1}^{N} C_{ji} A_j x_j(t), \quad i = 1, 2, 3, \ldots, N \tag{25}$$

The above system of linear equations can be compactly written as

$$\boldsymbol{S}(t) = \boldsymbol{B}\boldsymbol{x}(t), \tag{26}$$

where,

$$\boldsymbol{B} \equiv \boldsymbol{C}^T \mathbf{diag}(\boldsymbol{A}). \tag{27}$$

Let the vector $\boldsymbol{T}$ denote the service rates (or, thresholds) of the proxies. As discussed before, FastRoute's greedy heuristic continuously monitors the overload metric $\zeta_i(t) \stackrel{\text{def}}{=} S_i(t) - T_i$. It feasibly reduces $x_i(t)$ if $\zeta_i(t) > 0$ and feasibly increases $x_i(t)$ otherwise. The following simple control mechanism complies with the above general principle, which we analyze subsequently:

$$\frac{dx_i(t)}{dt} = -\beta R(x_i(t)) \big(\boldsymbol{B}\boldsymbol{x}(t) - \boldsymbol{T}\big)_i \quad \forall i. \tag{28}$$

The factor $R(x_i(t)) \equiv x_i(t)(1 - x_i(t))$ is called the *Regularizer*. The regularizer has the property that $R(0) = R(1) = 0$ and it is strictly positive in the open-interval $(0, 1)$. This nonlinear pre-factor is necessary for confining the dynamics

of $\boldsymbol{x}(t)$ to the N-dimensional unit hypercube $\mathbf{0} \leq \boldsymbol{x}(t) \leq \mathbf{1}$, ensuring the feasibility of the control (28)[4]. Although other choices of the functional form of the regularizer are possible, we choose the above function for its simplicity. 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 (28):

Theorem 4.1. *Consider the following system of ODE*

$$\dot{x_i}(t) = -R(x_i(t))(\boldsymbol{B}\boldsymbol{x}(t) - T)_i, \quad \forall i \tag{29}$$

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 (29) *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 8.2.

The following theorem reveals an interesting feature of the greedy algorithm. It states that, along any periodic orbit of the system dynamics, the average load at any node i is equal to the threshold T_i of that node.

Theorem 4.2. *Consider the system* (28) *with possibly time-varying arrival rate vector $\boldsymbol{A}(t)$, such that, the system operates in a periodic orbit. Then the time-averaged user load on any node i is equal to the threshold T_i of that node, i.e.*

$$\frac{1}{\tau}\int_0^\tau S_i(t)dt = \bar{T}, \ \ \forall i, \tag{30}$$

where the integral is taken along a periodic orbit of a period τ.

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

PROOF. See Appendix 8.3.

4.2. Avoiding Locally Uncontrollable Overload

Having established the feasibility and soundness of the control mechanism (28), we return to our original problem of avoiding locally uncontrollable overload, described in Section 3.2. In the following, we derive sufficient conditions for the correlation matrix $\boldsymbol{C}$ and the DNS arrival rate vector $\boldsymbol{A}$, for which the system is stable, in the sense that no locally uncontrollable overload situation takes place. For a fixed correlation matrix, the subset of arrival rates, for which the system remains stable under the greedy heuristic, is called its stability region $\boldsymbol{\Pi_C}$.

Characterization of the Stability Region $\boldsymbol{\Pi_C}$

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. The formal description and derivation of the result are provided in Appendix 8.4, which involves linearization of the ODE (28) around certain fixed points. Here we outline a simple and intuitive explanation of the stability region $\boldsymbol{\Pi_C}$.

We proceed by contradiction. Suppose that node i is undergoing a locally uncontrollable overload at time t. Hence, by definition, the following two conditions must be satisfied at node i

$$S_i(\infty) - T_i > 0, \tag{31}$$

$$x_i(\infty) = 0. \tag{32}$$

Here, Eqn. (31) denotes the fact that FastRoute node i is *overloaded* in the steady state. Eqn. (32) denotes the fact that this overload is *locally uncontrollable*, since even after node i's DNS-server has offloaded *all* incoming requests to L_2 (the best that it can do with its local information), it is overloaded. Using Eqn. (2) in conjunction with (31) and (32), we have:

$$\sum_{j \neq i} C_{ji} A_j x_j(\infty) > T_i, \tag{33}$$

Since $0 \leq x_j(\infty) \leq 1$, a necessary condition for uncontrollable overload (33) 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

$$\boldsymbol{\Pi_C} = \{\boldsymbol{A} \geq \boldsymbol{0} : \sum_{j \neq i} C_{ji} A_j \leq T_i, \ \forall i = 1, 2, \ldots, N\} \tag{34}$$

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 (28), we can show that a two-node system (as depicted in Figure 4) is able to control DNS arrivals $\boldsymbol{A}$ 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 3 (see Figure 4). Let the correlation matrix $\boldsymbol{C}$ for the system be parametrized as follows:

$$\boldsymbol{C}(\alpha, \beta) = \begin{pmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{pmatrix} \tag{35}$$

Then we have the following theorem:

Theorem 4.3. *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 two-node CDN 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 [31], whereas proof for part-(2) and (3) follows from linearization arguments. See Appendix 8.5 for details.

We emphasize that the part-(2) of the Theorem 4.3 is surprising, as it shows that the system remains locally controllable, no matter how large the incoming DNS-request arrival rate be (c.f. Section 3.2) . The 2D vector field plot in Figure 7 illustrates the above result. In Figure 7(a), the matrix $\boldsymbol{C}$ is taken to be one satisfying the condition of part (2) of lemma 4.3. 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 7(b) we plot the 2D vector field of a

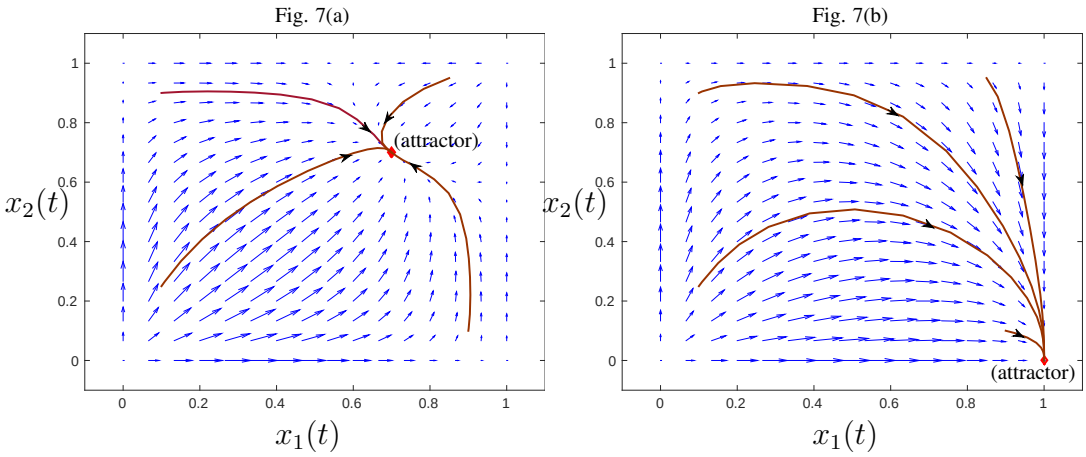


Figure 7: 2D vector fields of a two-node system, illustrating locally controllable (Fig. $7(a)$) and locally uncontrollable (Fig. $7(b)$) overloads.

locally uncontrollable system (e.g., the system in Fig. 4). 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 cannot exist in the system.

An Application of Theorem 4.3. 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

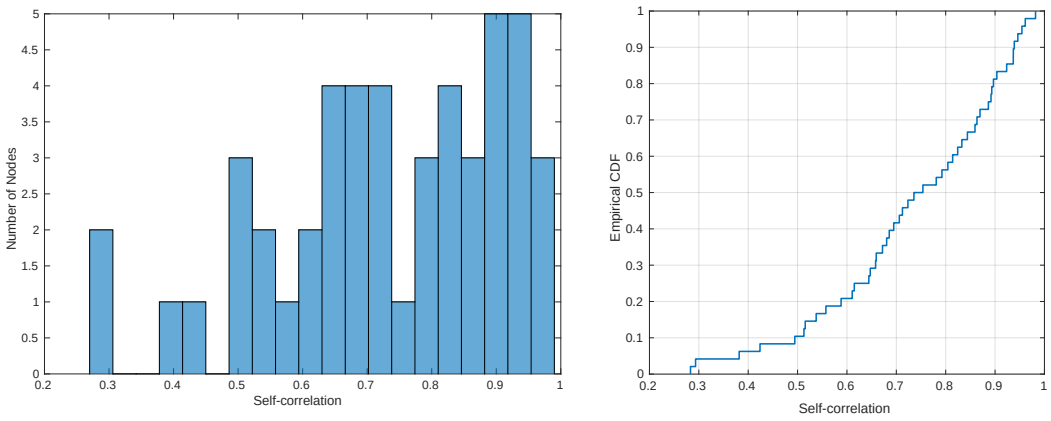
$$\alpha(G_1) = \frac{\sum_{i\in G_1}\sum_{j\in G_1} C_{ij}}{|G_1|}, \beta(G_2) = \frac{\sum_{i\in G_2}\sum_{j\in G_2} C_{ij}}{|G_2|}$$

satisfy the condition (2) of Theorem 4.3, 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.

5. Numerical Evaluation

Simulation Setup:

We use 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 [8]. The inter-node correlation matrix $\boldsymbol{C}$ is computed using system-traces collected over three



(a) (b)

Figure 8: (a) Histogram plot of the measured self-correlation values (b) Empirical Cumulative Distribution Function (CDF) of the measured self-correlation values. It is observed that there are exactly **four** nodes with self-correlation (C_{ii}) value less than 0.5.

months. The histogram and empirical CDF plot of the measured self-correlation values (obtained from the diagonal of the $\boldsymbol{C}$ matrix) are shown in Figure 8.

From the Figure 8 (a), we observe that there are four nodes in the system with self-correlation values less than 0.5. Taking our intuition from Theorem 4.3, these nodes are potentially at the danger of becoming uncontrollable under the action of the heuristic policy with high enough arrival rate. From our simulation result, we will see that this intuition is indeed correct.

For comparing the performance, we use the cost functions given in Eqns. (7) and (8). The parameters appearing in the cost functions are chosen as follows

$$\gamma_i = 1, \theta_i = 10, \eta_i = 1, T_i = 0.7 \qquad \forall i, \tag{36}$$

and the propagation delays d_is chosen i.i.d. at random from the uniform distribution supported in $[0, 1]$. DNS query arrival rates A_i's are assumed to be i.i.d. Poisson variables with expectation $\bar{A}$. For each value of $\bar{A}$, the simulation is run $N_E = 100$ times while randomizing over both A_i and d_i.

We consider the following two scenarios:

Case I: Controllable Overload

First, we consider the case of small arrival rate, so that, the greedy heuristic is able to control the overload. In other words, we restrict our attention to the scenarios when none of the nodes are overloaded under the action of the greedy

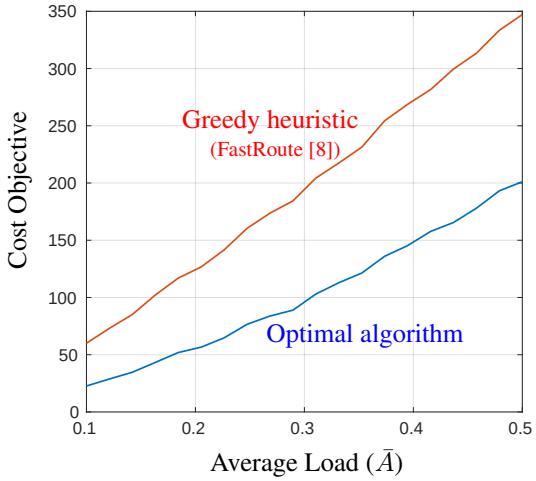


Figure 9: Comparing the performance of the greedy heuristic with the optimal algorithm under no-overload condition (small arrival rate).

heuristic. As a result, the steady state objective value achieved by the greedy heuristic is finite. We compare the performance of the greedy heuristic with the optimal algorithm of Section 3 in Figure 9. From the plot, we can clearly see that even in this favorable case, the optimal algorithm outperforms the greedy heuristic by a factor of at least 2. This plot clearly underscores the potential performance gain that CDNs can benefit from by switching to the optimal algorithm.

Case II: Uncontrollable Overload

Next, we consider the scenario with high arrival rate, which is common in the event of *Flash crowds* [30]. The performances of the CDN under the action of the optimal and the heuristic algorithm are shown in Figure 10. From the Figure 10(a) we observe that under the optimal algorithm there is a steady increase in the cost as the DNS arrival rate is increased. This is expected as system load increases with more arrivals. However, the resulting cost remains finite always. This, in turn, implies that *none* of the proxies are overloaded *regardless* of the arrival rate. This is in sharp contrast with the performance of the system under the greedy heuristic, shown in Figure 10(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 an 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 $\bar{A}$.

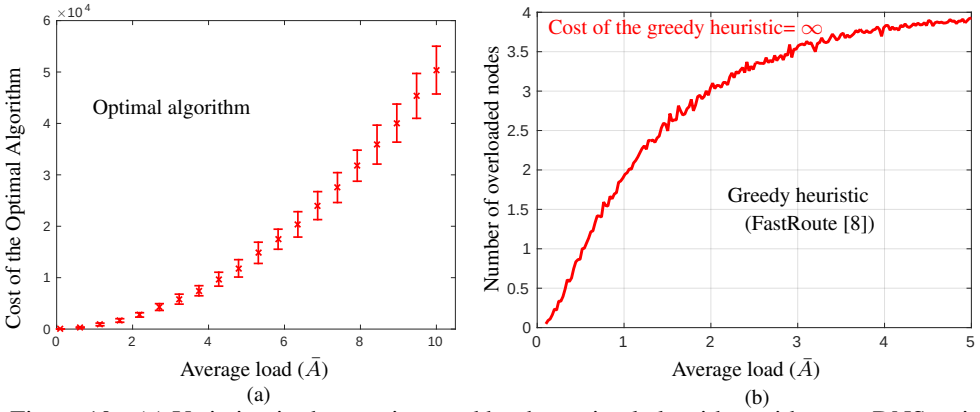


Figure 10: (a) Variation in the cost incurred by the optimal algorithm with mean DNS arrival rate $\bar{A}$. (b) Average number of nodes undergoing uncontrollable overload condition under the action of the greedy algorithm (Threshold $T_i = 0.7$ for all nodes). The total number of nodes in the system included in this study is $N = 48$. Note that, since the optimal cost is finite, the number of overloaded nodes under the optimal algorithm is zero. On the other hand, the greedy heuristic incurs an infinite cost as some nodes are overloaded (see Eqn. (7)).

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.

Saturation of number of uncontrollable nodes:

We observe from Figure 10(b), that the number of overloaded nodes saturates (tends to 4) as the arrival rate is increased. This effect can be explained by appealing to Theorem 4.3. From that theorem, we expect that the greedy heuristic might fail to control overload in those nodes whose self-correlation value is relatively small. Although the theorem is stated for a two-node system, the conclusion of the theorem is conjectured to hold in general. We mentioned earlier that in our simulation, there are exactly four nodes with self-correlation values less than 0.5 (see Figure 8). Since other nodes have self-correlation values more than this threshold value, the greedy heuristic successfully controls overload in other nodes. This explains the saturation effect in Figure 10(b).

6. Conclusion

In this paper, we extensively study the load management problem in modern CDNs, which use anycast. We first formulate the problem as a convex optimiza-

tion problem and propose a dual sub-gradient algorithm. To facilitate its practical implementation, we introduce the novel idea of FASTCONTROL packets, which effectively exploit the underlying anycast architecture. Next, we analyze the stability characteristics of a greedy heuristic, the *de facto* load management scheme for anycast-based CDNs. From our analytical and simulation studies, we find that the optimal algorithm significantly outperforms the heuristic. We also find operating points where the performance of the heuristic, although sub-optimal, does not result in an *uncontrollable overload* scenario and may be acceptable. Thus an informed choice between these two schemes 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. It would also be interesting to generalize the findings of Theorem 4.3 to more than two nodes.

7. Acknowledgement

We thank the anonymous reviewer for prompting us to investigate the saturation effect in Figure 10(b).

References

[1] Balachander Krishnamurthy, Craig Wills, and Yin Zhang. On the use and performance of content distribution networks. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 169–182. ACM, 2001.

[2] Stefan Saroiu, Krishna P Gummadi, Richard J Dunn, Steven D Gribble, and Henry M Levy. An analysis of internet content delivery systems. *ACM SIGOPS Operating Systems Review*, 36(SI):315–327, 2002.

[3] Valeria Cardellini, Michele Colajanni, and S Yu Philip. Dynamic load balancing on web-server systems. *IEEE Internet computing*, 3(3):28, 1999.

[4] YS Hong, JH No, and SY Kim. DNS-based load balancing in distributed web-server systems. In *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, pages 4–pp. IEEE, 2006.

[5] Craig Sirkin. Geo-locating load balancing, Nov 10, 2005. US Patent App. 11/271,941.

[6] Google cloud platform. `https://cloud.google.com/load-balancing/`.

[7] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010.

[8] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern cdns. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 381–394, 2015.

[9] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the performance of an anycast cdn. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, IMC '15, pages 531–537, New York, NY, USA, 2015. ACM.

[10] Amazon. `https://aws.amazon.com/route53/faqs/`.

[11] William T Zaumen, Srinivas Vutukury, and JJ Garcia-Luna-Aceves. Load-balanced anycast routing in computer networks. In *Computers and Communications, 2000. Proceedings. ISCC 2000. Fifth IEEE Symposium on*, pages 566–574. IEEE, 2000.

[12] Erol Basturk, Robert Engel, Robert Haas, Vinod Peris, and Debanjan Saha. Using network layer anycast for load distribution in the Internet. In *Tech. Rep., IBM TJ Watson Research Center*. Citeseer, 1997.

[13] Sandeep Sarat, Vasileios Pappas, and Andreas Terzis. On the use of anycast in DNS. In *Computer Communications and Networks, 2006. ICCCN 2006. Proceedings. 15th International Conference on*, pages 71–78. IEEE, 2006.

[14] Hirokazu Miura. Server selection policy in active anycast. 2001.

[15] Habibah Binti Hashim and Jamalul-lail Abd 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, volume 1, pages 5–pp. IEEE, 2005.

[16] Hussein A. Alzoubi, Seungjoon Lee, Michael Rabinovich, Oliver Spatscheck, and Jacobus Van der Merwe. Anycast CDNs revisited. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, New York, NY, USA, 2008. ACM.

[17] Muhammad Jaseemuddin, Arun Nanthakumaran, and Alberto Leon-Garcia. TE-friendly content delivery request routing in a CDN. In *Communications, 2006. ICC'06. IEEE International Conference on*, volume 1, pages 323–330. IEEE, 2006.

[18] Frank Kelly. Charging and rate control for elastic traffic. *European transactions on Telecommunications*, 8(1):33–37, 1997.

[19] Microsoft. Azure. `http://azure.microsoft.com/en-us/`, 2010.

[20] N Gautam. Performance analysis and optimization of web proxy servers and mirror sites. *European Journal of Operational Research*, 142(2):396–418, 2002.

[21] Shui Yu, Wanlei Zhou, and Yue Wu. Research on network anycast. In *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on*, pages 154–161. IEEE, 2002.

[22] Anees Shaikh, Renu Tewari, and Mukesh 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*, volume 3, pages 1801–1810. IEEE, 2001.

[23] Dimitri P Bertsekas and Robert G Gallager. *Data networks*. Prentice-hall, 1987.

[24] Tim Roughgarden. *Selfish routing and the price of anarchy*, volume 174. MIT press Cambridge, 2005.

[25] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[26] Dimitri P Bertsekas. *Convex optimization algorithms*. Athena Scientific, 2015.

[27] Ilan Lobel and Asuman Ozdaglar. Distributed subgradient methods for convex optimization over random networks. *Automatic Control, IEEE Transactions on*, 56(6):1291–1306, 2011.

[28] Atilla Eryilmaz, Asuman Ozdaglar, Devavrat Shah, and Eytan Modiano. Distributed cross-layer algorithms for the optimal control of multihop wireless networks. *IEEE/ACM Transactions on Networking (TON)*, 18(2):638–651, 2010.

[29] 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.

[30] Abhishek Chandra and Prashant Shenoy. Effectiveness of dynamic resource allocation for handling internet flash crowds. *TR03-37, Department of Computer Science, University of Massachusetts, USA*, 2003.

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

[32] HK Khalil. Nonlinear systems. *2nd Edition, Prentice Hall*, 1996.

[33] Walter Rudin. *Principles of mathematical analysis*, volume 3. McGraw-Hill New York, 1964.

[34] Katsuhiko Ogata and Yanjuan Yang. Modern control engineering. 1970.

8. Appendix

8.1. Proof of Lemma 3.2

PROOF. We have,

$$
\begin{aligned}
||\boldsymbol{g}_k||_2^2 &= \sum_{i=1}^{N}(S_i^{\text{obs}}(k) - S_i(k))^2 && (37)\\
&\leq \sum_{i=1}^{N}\big(S_i^{\text{obs}}(k)\big)^2 + \sum_{i=1}^{N} S_i^2(k) && (38)\\
&\leq \sum_{i=1}^{N}\big(\sum_{j=1}^{N} C_{ji}A_j x_j(k)\big)^2 + NT_{\max}^2 && (39)\\
&\leq \big(\sum_{i=1}^{N}\sum_{j=1}^{N} C_{ji}A_j\big)^2 + NT_{\max}^2 && (40)\\
&= \big(\sum_{j=1}^{N} A_j \sum_{i=1}^{N} C_{ji}\big)^2 + NT_{\max}^2 && (41)\\
&\leq (\sum_{j=1}^{N} A_j)^2 + NT_{\max}^2 && (42)\\
&\leq A_{\max}^2 + NT_{\max}^2 && (43)
\end{aligned}
$$

Here Eqn. (38) follows from non-negativity of S_i^{obs} and S_i, Eqn. (39) follows from the defining equation of S_i^{obs} and the constraint that $S_i \leq T_i$ (viz. Eqn. (14)), Eqn. (40) follows from the constraint $0 \leq x_i \leq 1, \forall i$, Eqn. (41) follows from the change of the order of summation and finally Eqn. (42) follows from the fact that $\boldsymbol{C}$ is a correlation matrix and hence its rows sum to unity (viz. Eqn. (1)).

8.2. Proof of Theorem 4.1

PROOF. First, we show that any solution of the system (28) (if exists) must lie in the unit hypercube $\mathcal{H}$. We prove it via contradiction. On the contrary to the claim, assume that for some solution of the system (28), there exists a component $x_i(\cdot)$ and a finite time $0 \leq \tau < \infty$ such that $x_i(\tau) < 0$. Since $x_i(\cdot)$ is continuous and $x_i(0) > 0$, by *intermediate value theorem*, there must exist a time $0 < t_0 < \tau$ such that $x_i(t_0) = 0$. Now consider the differential equation corresponding to the i^{th} component of the system (29). We substitute for all other components $\{x_j(t), j \neq i\}$ on the RHS of the following equation.

$$\dot{x_i}(t) = -R(x_i)(\sum_j B_{ij}x_j(t) - T_i), \qquad x_i(t_0) = 0 \tag{44}$$

Since the vector $\boldsymbol{x}(t)$ is $\mathcal{C}^1$ and the regularizer $R(\cdot)$ is assumed to be $\mathcal{C}^1$, the RHS of the equation (44) is $\mathcal{C}^1$. Hence (44) admits a *unique local solution*. However, note that the following is a solution to (44)

$$x_i(t) = 0, \qquad \forall t \geq t_0 \tag{45}$$

This is because $R(0) = 0$. By uniqueness, (45) is the *only* solution to (44). This contradicts the fact that $x_i(\tau) < 0$. Hence $\boldsymbol{x}(t) \geq \mathbf{0}, \forall t \geq 0$. In a similar fashion, we can also prove that $\boldsymbol{x}(t) \leq \mathbf{1}, \forall t \geq 0$. This proves that all solutions to (28) must lie in the compact set $\mathcal{H}$.
To complete the proof, we observe that the RHS of the system (28) is locally Lipschitz at each point in the compact set $\mathcal{H}$. Thus, the global existence of the solution of (28) follows directly from Theorem 2.4 of [32].

8.3. Proof of Theorem 4.2

PROOF. Let τ be the period of the orbit. Consider the i^{th} differential equation

$$\dot{x_i}(t) = -R(x_i)(S_i(t) - T_i) \tag{46}$$

$$\frac{dx_i}{R(x_i)} = -(S_i(t) - T_i)dt \tag{47}$$

Since $x_i(\cdot)$ belongs to the *interior* of the compact set $\mathcal{H}$, $\frac{1}{R(x_i(t))}$ does not have a zero in the denominator for the entire orbit. Hence $\frac{1}{R(x_i(t))}$ is continuous and its Riemann integral exists [33]. Integrating both sides from 0 to τ, we have

$$\int_0^\tau \frac{dx_i}{R(x_i)} = \int_0^\tau (S_i(t) - T_i)dt \tag{48}$$

Let $J(x_i)$ be an anti-derivative of $\frac{1}{R(x_i)}$. Hence using the fundamental theorem of calculus [33], we can write the LHS of (48) as

$$J(x_i(\tau)) - J(x_i(0)) = \int_0^\tau S_i(t)dt - \tau T_i \tag{49}$$

Since the orbit is assumed to have a period τ, we have $x_i(\tau) = x_i(0)$. Hence $J(x_i(\tau)) - J(x_i(0)) = 0$. Thus we have

$$\bar{S_i} \equiv \frac{1}{\tau}\int_0^\tau S_i(t)dt = T_i$$

8.4. Formal Derivation of the Stability Condition

Let us write the system (28) conveniently as $\dot{\boldsymbol{x}} = \boldsymbol{F}(\boldsymbol{x})$. Consider a fixed point $\overline{x}$ of the system such that the k^{th} node faces an uncontrollable overload condition. By *Hartman-Grobman* theorem [31], it is enough to consider linearized version of the system to determine the stability of fixed points. The first-order Taylor expansion about the fixed point $\overline{x}$ yields the following:

$$\dot{\boldsymbol{x}} \approx F(\bar{\boldsymbol{x}}) + \frac{\partial \boldsymbol{F}(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}=\overline{\boldsymbol{x}}}(\boldsymbol{x} - \bar{\boldsymbol{x}}) = \frac{\partial \boldsymbol{F}(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}=\overline{\boldsymbol{x}}}(\boldsymbol{x} - \bar{\boldsymbol{x}})$$

Where, $\frac{\partial \boldsymbol{F}(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}=\overline{\boldsymbol{x}}}$ denotes the Jacobian [34] of the system (28) evaluated at the fixed point $\boldsymbol{x} = \overline{\boldsymbol{x}}$ and $B_{ij} = C_{ji}A_j$. The second equation follows because $\overline{\boldsymbol{x}}$ is assumed to be a fixed point (and consequently $\boldsymbol{F}(\overline{\boldsymbol{x}}) = \boldsymbol{0}$).
Next, we proceed to explicitly compute the Jacobian of the system (28) at a given point $\boldsymbol{x}$. Note that the i^{th} row of the system equation is given by

$$F_i(\boldsymbol{x}) \equiv -x_i(1 - x_i)(\sum_j B_{ij}x_j - T) \tag{50}$$

Thus for $i \neq j$, we have

$$\frac{\partial F_i}{\partial x_j} = -x_i(1 - x_i)B_{ij} \tag{51}$$

and for $i = j$, the diagonal entry is given by

$$\begin{aligned} \frac{\partial F_i}{\partial x_i} &= -\Big(x_i(1 - x_i)B_{ii} + (1 - 2x_i)(\sum_j B_{ij}x_j - T)\Big) \\ &= -\Big(x_i(1 - x_i)B_{ii} + (1 - 2x_i)(S_i - T)\Big) \end{aligned} \tag{52}$$

Since the node k is assumed to undergo an uncontrollable overload condition, we necessarily have $x_k = 0$. Hence from Eqns (51) and (52), in the k^{th} row of the Jacobian matrix, the off-diagonal entries are all zero and the diagonal entry is $-(S_k - T)$. Hence if $S_k - T < 0$, at least one eigenvalue of the Jacobian matrix at the fixed point $\overline{x}$ is strictly positive and hence the fixed point $\overline{x}$ is unstable. This implies that a sufficient condition to avoid uncontrollable overload at node k is given by

$$\sum_{j \neq k} C_{jk}A_j \leq T_k \tag{53}$$

where we have used the fact that $x_i(t) \leq 1, \forall i$ and $x_k = 0$. The derivation of the sufficient condition for the absence of uncontrollable overload condition is completed by taking the intersection of hyperplanes (53) for all $k = 1, 2, \ldots, N$. ■

8.5. Proof of Theorem 4.3

PROOF. **part-(1) [non-existence of periodic orbits]**
We use Dulac's criterion to prove the non-existence of periodic orbits for the general two-node system. For ease of reference, we recall Dulac's criterion below :

Theorem 8.1 (Dulac's criterion [31]). *Let $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ be a continuously differentiable vector field defined on a simply connected subset R of the plane. If there exists a continuously differentiable, real-valued function $g(\boldsymbol{x})$ such that $\nabla \cdot (g\dot{\boldsymbol{x}})$ has one sign throughout $\mathcal{D}$, then there are no closed orbits lying entirely in $\mathcal{D}$.*

Now we return to the proof of the result. Let $\mathcal{H}_2 = [0, 1]^2$ be the unit square, where by virtue of theorem 4.1, the trajectory of the two-node system lies for all time $t \geq 0$. Thus, it suffices to show that there does not exist any periodic orbit in its interior $\mathring{\mathcal{H}}_2 \equiv \mathcal{D}$. It is obvious that the region $\mathcal{D}$ is simply connected. Now consider the following $g(\boldsymbol{x})$ for application of the Dulac's criterion,

$$g(\boldsymbol{x}) = \frac{1}{x_1 x_2 (1 - x_1)(1 - x_2)} \tag{54}$$

It is easy to verify that $g(\boldsymbol{x})$ is continuously differentiable throughout the region $\mathcal{D}$. We next evaluate the divergence

$$\nabla \cdot (g\dot{\boldsymbol{x}}) = -\beta \left(\frac{B_{11}}{x_2(1 - x_2)} + \frac{B_{22}}{x_1(1 - x_1)} \right) \tag{55}$$

It is easy to verify that the term within the parenthesis is always strictly positive throughout the region $\mathcal{D}$. Hence, by Dulac's criterion, there are no closed orbits in $\mathcal{D}$. This proves the result.

part-(2) and (3) [controllability of the system]
Let the arrival rates to node 1 and 2 be given by A_1 and A_2. Let x_1 and x_2 denote the operating point of the system at the steady state. Our objective is to find sufficient conditions on the arrival rate vector (A_1, A_2), under which the operating points (x_1, x_2) such that either $x_1 = 0$ or $x_2 = 0$ (i.e. full offload to Layer-II) are avoided in the *steady state*. This will ensure that no uncontrollable overload situation takes place in the system. First, we consider the fixed point

$$x_1 = 0, x_2 = 1 \tag{56}$$

This fixed point will be stable if both the eigenvalues of the Jacobian matrix at this point are negative. From Eqns. (51) and (52) we have the following two conditions:

$$S_1 > T, S_2 < T$$

i.e.,

$$(1-\alpha)A_2 > T, \beta A_2 < T$$

i.e.,

$$\frac{T}{1-\alpha} < A_2 < \frac{T}{\beta} \tag{57}$$

Similarly, analyzing the stability of the fixed points around the point $x_1 = 1, x_2 = 0$, we obtain that this fixed point will be unstable if

$$S_1 < T, S_2 > T$$

i.e.,

$$\alpha A_1 < T, (1-\beta)A_1 > T$$

i.e.,

$$\frac{T}{1-\beta} < A_1 < \frac{T}{\alpha} \tag{58}$$

Note that if $\alpha + \beta > 1$, both the above regions (57) and (58) will be empty and the fixed points $(1, 0)$ and $(0, 1)$ will be always unstable. Thus, the operating point will not converge to these undesirable fixed points in the steady state.
Now consider the (possibly feasible) fixed point (x_1, x_2) such that

$$x_1 = 0, S_2 = T \tag{59}$$

The above condition translates to,

$$\begin{aligned} x_1 = 0, A_2 x_2 \beta = T \\ x_1 = 0, x_2 = \frac{T}{A_2 \beta} \end{aligned} \tag{60}$$

This fixed point will be feasible provided $\frac{T}{A_2\beta} < 1$. The Jacobian matrix about this fixed point is evaluated as

$$\frac{\partial \boldsymbol{F}(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}} = -\begin{pmatrix} (1-\beta)\frac{T}{\beta} - T & 0 \\ \frac{T}{A_2\beta}(1-\frac{T}{A_2\beta})B_{21} & \frac{T}{A_2\beta}(1-\frac{T}{A_2\beta})B_{22} \end{pmatrix}$$

From the Jacobian matrix above, we conclude that both of its eigenvalues are negative provided the following two conditions hold

$$A_2 > \frac{T}{\beta}\,,\ \beta < \frac{1}{2}. \tag{61}$$

Doing similar analysis around the fixed point $(S_1 = T, x_2 = 0)$, we conclude that the above fixed point will be stable for all arrival rates $A_1 > \frac{T}{\alpha}$ and $\alpha < \frac{1}{2}$.
Finally, to obtain efficient operating region for the system (with no uncontrollable overload situation), we take union over stability region of all undesired fixed points and take the complement of it. Hence, if $\alpha > \frac{1}{2}, \beta > \frac{1}{2}$ all the undesirable fixed points are unstable and hence the uncontrollable overload situation is avoided for *all* DNS-request arrival rates $\boldsymbol{A}$. This proves part (2) of the theorem.
On the other hand, if either $\alpha < \frac{1}{2}$ or $\beta < \frac{1}{2}$ holds, then a sufficient condition for the stability of the system is given by

$$A_1 < \frac{T}{1-\alpha}, A_2 < \frac{T}{1-\beta}$$

This proves part (3) of the theorem.