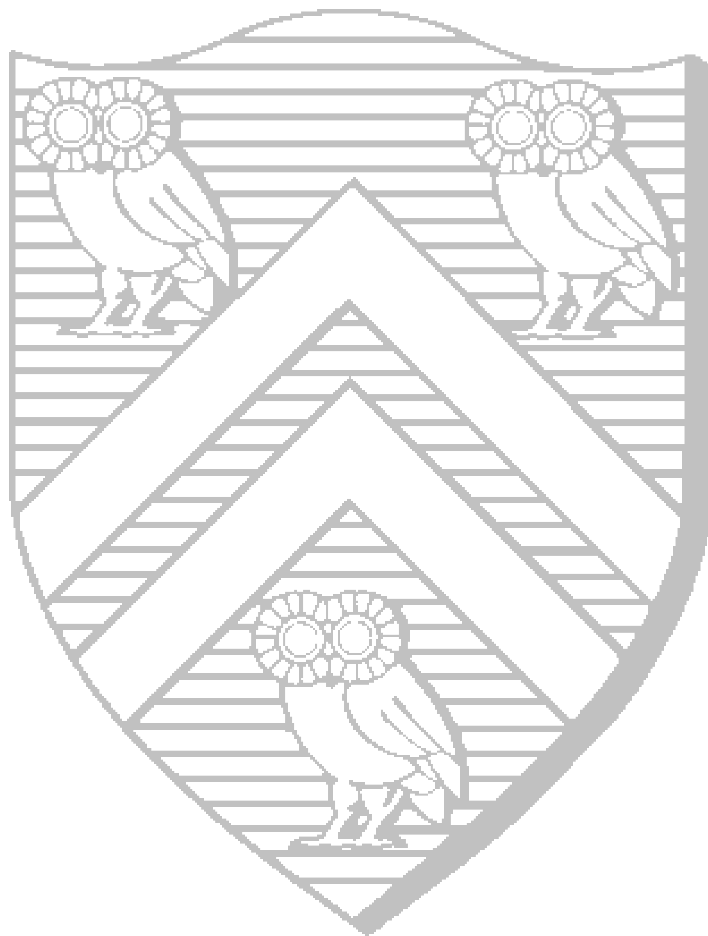
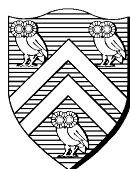
LALA: Location Aware Load Aware Overlay Anycast

Animesh Nandi





Thesis: Master of Science
Computer Science
Rice University, Houston, Texas (April 2004)

RICE UNIVERSITY

LALA: Location Aware Load Aware Overlay Anycast

by

Animesh Nandi

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:

Dr. Peter Druschel, Chair
Professor of Computer Science,
Rice University

Dr. Ion Stoica
Assistant Professor of Computer Science,
University of California, Berkeley

Dr. Dan.S Wallach
Assistant Professor of Computer Science,
Rice University

Dr. T.S. Eugene Ng
Assistant Professor of Computer Science,
Rice University

Houston, Texas

April, 2004

LALA: Location Aware Load Aware Overlay Anycast

Animesh Nandi

Abstract

Anycast is a powerful paradigm for managing and locating resources in decentralized distributed systems. Ideally, an anycast system must be scalable, location-aware and load-aware. Location-awareness means that the anycast system should be able to locate a resource that is near the client in the network. Load-awareness means that it must be able to disperse load to avoid overloading group members in the case of high demand in a certain region of the network.

Existing anycast systems are either location-aware or load-aware, but not both. We motivate LALA, a generic architecture for doing scalable, location-aware, load-aware anycast that realizes the following anycast functionality: Given a client request, our goal is to select the closest anycast server that has enough resources to satisfy the client's request. We show how LALA can be designed on some of the existing overlay anycast architectures and close with an evaluation that demonstrates its effectiveness.

Acknowledgments

The work in this thesis was done under the guidance of my advisor Dr. Peter Druschel and Dr. Ion Stoica. I am thankful to them for their guidance and constant encouragement throughout the duration of my thesis. The genesis of the thesis took place in the summer of 2003 when I was at University of California, Berkeley as a visiting graduate student working with Dr. Ion Stoica.

I would like to thank my peers who gave me feedback on my work and discussed issues that needed to be sorted out. I would specially like to thank Alan Mislove for providing me with code libraries and valuable information on space filling curves. I would also like to thank Rongmei Zhang for providing us with gt-itm trace data and Eugene Ng for helping us with GNP.

I would like to thank my parents and my elder brother who constantly kept me motivated at all times in my life to realize my dreams. I take this opportunity to thank them. I would also like to take this opportunity to thank my friends Abhishek, Ajay, Alan, Andreas, Ansley, Anwis, Atul, Arindam, Karthik, Krishnendu, Lakshmi, Sonesh, Sumit and others who encouraged me from time to time to accomplish this work.

Contents

Abstract ii

Acknowledgments iii

List of Illustrations vi

1 Introduction of LALA 1

1.1 Background: Overlay anycast 1

1.2 Motivation for LALA 3

1.3 Central idea behind LALA 4

1.4 Contributions of the thesis 6

1.5 Thesis Organization 6

2 Design of LALA on DHT-based anycast systems like $i3$ 7

2.1 Background: Anycast in $i3$ 7

2.2 Responsible region (RRegion) in $i3$ 8

2.3 LALA Algorithm 9

2.3.1 Request Assignment 9

2.3.2 Overload Protection 10

2.4 Realization of LALA on DHT-based anycast systems 11

2.5 LALA in $i3$ (**i3-lala**) 15

3 Design of LALA on Tree-based anycast systems like Scribe 16

3.1 Background: Anycast in Scribe 16

3.1.1 Pastry 16

3.1.2 Scribe 18

3.2 Responsible region (RRegion) in Scribe 19
3.3 LALA Algorithm 20
3.4 LALA in Scribe (**Scribe-lala**) 22

4 Evaluation 23

4.0.1 Simulation settings 24
4.0.2 Evaluation metrics 24
4.0.3 Simulation Methodology 25
4.0.4 Simulation Results 26

5 Background and Related Work 33

6 Conclusions and Future Work 38

Bibliography 41

Illustrations

1.1 Self-Adjusting Responsible Regions 5

2.1 Partioning of Geographic space using RRegions 9
2.2 RRegions in Circular ID Space 13

3.1 Routing a message from the node with nodeId $65a1fc$ to key $d46a1c$ 17
3.2 Responsible regions in Tree-based anycast system 19
3.3 Adjusting size of RRegions in Scribe 20

4.1 Maximum waiting time vs. utilization 26
4.2 Weighted rank metric vs. utilization 27
4.3 Maximum waiting time vs. utilization : Flashcrowd scenario 29
4.4 Weighted rank metric vs. utilization : Flashcrowd scenario 29
4.5 Scaled weighted rank metric vs. Anycast Group Size 30
4.6 Maximum waiting time vs Anycast Group Size 31

Chapter 1

Introduction of LALA

In this chapter, we first give some background on anycast and recently proposed scalable overlay anycast services. We discuss the current limitations of these approaches and motivate the need for a scalable, location-aware and load-aware (LALA) anycast architecture. We define the goal of such an architecture and abstractly discuss our idea in realizing this goal. We highlight the main contribution of the thesis and finally conclude with a description of the organization of the thesis.

1.1 Background: Overlay anycast

Anycast is a group communication paradigm. The anycasting paradigm was designed to aid in server selection by directing a client's request to the nearest among a relatively small and static group of 'functionally equivalent' servers. The anycast group members share the same anycast address, and as originally defined in RFC 1546 [PMM93], anycasting provides: " a stateless best effort delivery of an anycast datagram to at least one host, and preferably only one host, which serves the anycast address ". The anycasting paradigm as originally designed to be implemented at the network IP level, had some drawbacks. Network level anycast did not work well with highly dynamic groups because of the high overhead involved in joining/leaving a group. Moreover, it needed support from network routers and its wide usage relied on the deployment of supporting routers. Moreover, as pointed out by Bhattacharjee et al. [BAZ+97], it is beneficial to support a variety of metrics such as server load while selecting an anycast server instead of always choosing the closest server based on some shortest path metric like hop count. This led to the motivation of implementing anycast services on top of overlay networks.

Recently, scalable overlay anycast services [SAZ+02, CDKR03, ZKJ01] built on structured peer-to-peer (p2p) overlays [RFH+01, SMK+01, RD01, ZKJ01] have been proposed. Structured p2p overlays have recently gained popularity as a platform for building scalable, self-organizing, decentralized applications. These structured overlays conform to a specific graph structure that enables them to locate objects by exchanging $O(logN)$ messages where N is the number of nodes in the overlay using only $O(logN)$ state. Common to all these overlay protocols is the concept of a node, nodeId and identifier space and keys. As defined by Dabek et al. [DZD+03], a *node* represents an instance of a participant in the overlay (one or more nodes may be hosted by a single physical IP host). Participating nodes are assigned uniform random *nodeIds* from a large *identifier space*. Application-specific objects are assigned unique indentifiers called *keys*, selected from the same Id space.

All the existing systems provide higher level abstractions like distributed hash tables (DHT). The DHT abstraction provides the same functionality as a traditional hashtable, by storing the mapping between a key and a value. The interface implements a simple store and retrieve functionality where the value is always stored at the live node to which the key is mapped, referred to as the root node for the key. Values can be objects of any type. For instance, they can be disk blocks that are stored and retrieved on the basis of their content-hashed keys in CFS [DKK+01]. Thus, with the DHT abstraction, each data item stored in the overlay is associated a unique identifier that is used to store and retrieve that data item from the DHT.

While all nodes in the overlay play a similar role in the overlay algorithms, the P2P system as such can be highly heterogenous. A P2P system like Gnutella [Gnu00] and Kazaa may consists of peers that range from old desktops behind modem lines to powerful servers connected to the Internet through high-bandwidth lines. Providing a first degree of load balancing in these heterogeneous systems can be done with *virtual servers*. A virtual server resembles a single peer to the underlying DHT, but each physical node can be responsible for a number of virtual servers proportional to its capacity. For example in Chord or Pastry, although each virtual server is responsible for a contiguous chunk of the

identifier space, a node with multiple virtual servers can be responsible for owning multiple non-contiguous portions of the ring.

With the advent of the scalable anycast services [SAZ+02, CDKR03, ZKJ01] built on these peer-to-peer overlays, the anycasting paradigm started being viewed as powerful building block for managing and locating resources in large-scale decentralized distributed systems. As in original anycast, the anycast group was viewed as consisting of 'functionally equivalent' group of resources and the anycasting paradigm was designed in helping the client locate the 'best' resource among them. 'Best' here is defined according to some metric which optimizes the service received by the client. Nodes having a particular resource join a group to advertise the availability of that resource. Nodes that seek to locate the resource send an anycast message to the group. The message is delivered to a nearby member, thus matching provider and consumer of a resource while minimizing delay and network usage.

1.2 Motivation for LALA

Our motivation for this work arises from the need of a overlay anycast architecture that is scalable, location-aware and load-aware. Consider a set of clients that issue requests and a set of servers placed at different locations in the Internet that can serve these requests. Example of requests can be downloading a file, allocating storage for replicating popular content, etc. Anycast aims to match each client request to a server that optimizes (by some metric) the service received by the client.

To enable efficient resource discovery in large-scale decentralized systems, an anycast service must (1) itself be scalable and decentralized; (2) be location aware, i.e., it must be able to deliver anycast messages to a group member that is near the client in the network; at the same time, it must be (3) load aware, i.e., it must be able to disperse load to avoid overloading group members in the case of high demand in a certain region of the network.

Previous anycast systems fail to meet at least one of these requirements. Existing scalable and decentralized overlay anycast services such as Scribe [CDKR03] anycast, delivers

messages to nearby group members. Once a group member's resources are consumed, the member is required to leave the group. This is appropriate for relatively coarse-grained resource allocation, but fine-grained resources require a more continuous strategy for load shedding. The anycast service in $i3$ [SAZ+02] lends itself to more fine-grained load balancing, but it does not necessarily deliver messages to nearby group members.

We assume a simple model in which the service provided to a client depends solely on the server's current load and the distance between the client and the server. Further, we assume that

1. As long as the load of a server is lower than a threshold L_{high}, a client will see little difference in the service provided by the server. In other words, the client will be as happy with the service provided by an idle server as with the service provided by the same server when its load is just below L_{high}.

2. The shorter the distance (latency or hop count) between the server and the client, the better the service the client receives. Note that we implicitly assume that the network is not the bottleneck. We believe this is a reasonable assumption in today's Internet where the congestion mostly occurs at the edge of the network.

In particular, our location-aware and load-aware (**LALA**) anycast architecture strives to realize the following **anycast functionality**: *given a client request, our goal is to select the closest anycast server that has enough resources to satisfy the client's request (i.e., whose load does not exceed L_{high}).*

1.3 Central idea behind LALA

For an anycast architecture to be location-aware, we require that anycast requests from a particular geographic area should be delivered to anycast servers residing in that area. For instance, anycast requests arising from a region in Texas should be delivered to anycast servers residing in Texas. In the absence of anycast servers in Texas, the anycast request should be delivered to anycast servers that lie in the vicinity of Texas. In order to achieve

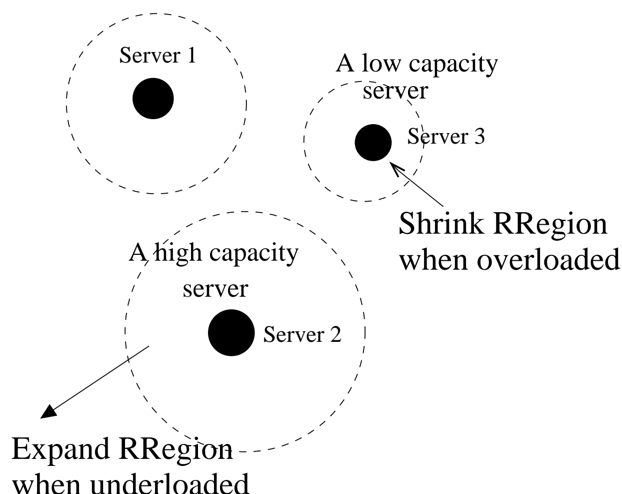


Figure 1.1 : Self-Adjusting Responsible Regions

this effect, each anycast server is associated with a *responsible region (RRegion)*, which approximately represents the geographic area centered around the location of the anycast server. The responsible region of an anycast server logically represents the willingness of the anycast server to serve clients belonging to that region. We would additionally like to have the size of the responsible region of a anycast server to be proportional to its capacity since a large capacity anycast server can potentially serve many more clients as compared to a low capacity server. Having static-sized responsible regions does not help since there could be a sudden flashcrowd of requests from a particular region. Therefore for the architecture to be load-aware, the anycast server should dynamically shrink or increase the size of its responsible region depending on whether the server is overloaded or underloaded, respectively. This is shown in Figure 1.1. The notion of self-adjusting responsible regions is the key idea to realize our location-aware and load-aware anycast (LALA) architecture.

The concept of the responsible region of a server can have different representations in different anycast architectures. Amongst the recently proposed scalable overlay anycast architectures [SAZ+02, CDKR03, ZKJ01], we have identified DHT-based anycast systems like $i3$ [SAZ+02] and tree-based anycast systems like Scribe [CDKR03] to represent two different designs of how anycast systems might be built. Depending on the representa-

tion of the responsible region in the anycast architecture, we then develop techniques to dynamically adjust the size of the responsible region in order to realize LALA's anycast functionality.

1.4 Contributions of the thesis

We make the following contributions in this thesis:

1. We point out the limitations of the current overlay anycast architectures and motivate the need for a location-aware load-aware (LALA) anycast architecture. We define the goal of such an architecture and discuss abstractly our idea in realizing this goal.

2. We sketch the design of the LALA architecture on two classes of anycast systems: DHT-based anycast systems like $i3$ and tree-based anycast systems like Scribe. Simulation results show the effectiveness of our proposed designs in realizing the goal of LALA.

1.5 Thesis Organization

In Chapter 2, we will describe the design of LALA on DHT-based anycast systems like $i3$. In Chapter 3, we will describe the design of LALA on tree-based anycast systems like Scribe. In Chapter 4, we will evaluate our proposed designs of LALA on $i3$ and Scribe. In Chapter 5, we discuss background and related work. Finally in Chapter 6, we will conclude and discuss future work.

Chapter 2

Design of LALA on DHT-based anycast systems like $i3$

In this chapter, we outline the design of the LALA architecture on DHT-based anycast systems like $i3$. We first give some background on how anycast is done in $i3$ and discuss its limitations. We then identify how LALA can be applied to such an architecture. Finally, we show how our design can be implemented on $i3$.

2.1 Background: Anycast in $i3$

$i3$ [SAZ^{+}02] is an overlay-based Internet Indirection Infrastructure that offers a rendezvous-based communication abstraction. Instead of explicitly sending a packet to a destination, each packet is associated with an identifier and this identifier is used by the receiver to obtain delivery of the packet. In the simplest model, packets are pairs (*id, data*) where *id* is an m bit identifier and *data* consists of a payload. Receivers use *triggers* to indicate their interest in packets. In the simplest form, triggers are pairs (*id, addr*), where *id* represents the trigger identifier and *addr* represents a node's address, which consists of an IP address and a port number. A trigger (*id, addr*) indicates that all packets with an identifier *id* should be forwarded (at the IP level) by the $i3$ infrastructure to the node identified by *addr*. $i3$ uses inexact matching between identifiers. Assuming that identifierrs are m bits long, there exists some *exact-match-threshold* k with $k < m$. Inexact matching implies that an trigger identifier id_t *matches* a packet identifier id if and only if id_t is a longest prefix match among all other trigger identifiers and this prefix match is at least as long as the exact-match-threshold k.

Anycast is achieved in $i3$ by having all anycast group members maintain triggers that are identical in the k most significant bits. These k bits serve as the anycast group identifier.

To send a packet to this anycast group, a sender uses an identifier whose k bit prefix matches the anycast group identifier. The packet is then delivered to the member of the group whose trigger identifier matches the packet identifier according to the longest prefix matching rule. Basic server selection using the anycast paradigm of $i3$ can be done by end-hosts choosing the last $m-k$ bits of the identifier carefully. $i3$ proposes two ways of doing server selection: (1) Given a set of web servers the goal is to balance the client requests among these servers. (2) Given a set of web servers the goal is to select a server that is close to the client in terms of latency.

Load balancing client requests among a set of web servers can be done by setting the $m - k$ least significant bits of both trigger and packet identifiers to random values. Moreover, if servers have different capacities then each server can insert a number of triggers proportional to its capacity. More generally, the servers can dynamically adjust the number of triggers on the basis of their current load. Selecting the closest server can be done by encoding the server's location in the $m-k$ least significant bits of the trigger identifiers and similarly encoding the client location in the last $m - k$ bits of the packet identifiers. Location encodings such as postal codes (zip codes) can be used and would cause the packet to be forwarded to a server that is near the client. The above anycast proposals in $i3$ are either load-aware or location-aware, but not both.

2.2 Responsible region (RRegion) in $i3$

As mentioned in Section 1.3, the concept of responsible region has different representations in different anycast architectures. In this section we will try to identify the representation of a responsible region in $i3$. Consider the way in which location encodings such as postal codes is used to deliver an anycast request from a client to a close anycast server whose postal code matches the client's postal code closely (This was explained in detail in Section 2.1 above). Here the entire geographic space is partitioned into the responsible regions of the different anycast servers. The responsible region of each anycast server approximately represents the postal code zone in which the anycast server is located. We will

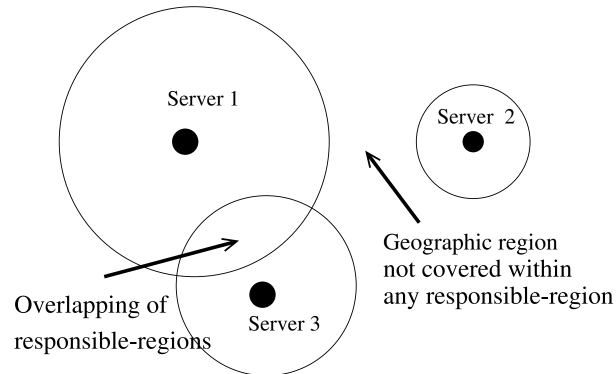


Figure 2.1 : Partioning of Geographic space using RRegions

describe later in Section 2.4 how different location encoding techniques other than postal codes could be used. Observe that in general, the responsible regions of different servers could overlap and additionally there could be regions of geographic space which are not covered within any responsible region. This is shown in Figure 2.1.

2.3 LALA Algorithm

We will now describe abstractly how the LALA algorithm works. Assume for now, that each server s is associated a responsible region, denoted *RRegion(s, k)*, that represents the region of network within distance k of server s.* As per our definition of responsible region, *RRegion(s, k)* expresses the willingness of server s to serve any client that belongs to the responsible region.

We are now in position to describe (1) how client requests are assigned to servers, and (2) how a server can avoid being overloaded.

2.3.1 Request Assignment

Consider a request issued by client c. We consider two cases:

*Distance k is also called the diameter of the responsible region.

1. If client c belongs to at least one responsible region, then among all servers whose responsible regions cover c, the request is delivered to the closest one.

2. If there is no responsible region covering client c, the request is delivered to a server s with probability $P(s)$, where $P(s)$ is proportional to the size of s's responsible region. This heuristic is motivated by the fact that a server with a larger responsible region is in general willing to receive more requests. We are effectively adopting a weighted random strategy of distributing anycast requests, where the weights are proportional to the available capacities of the servers. Strategies such as choosing a close server in this scenario does not work, because a flashcrowd of requests from a particular region would overload a nearby located server even if it has shrinked its responsible region. Request assignment in this case is therefore load-aware only and not location-aware. However, in a system with enough aggregate server capacity we expect this case to be invoked very rarely.

2.3.2 Overload Protection

Recall that the server's primary goal is to maintain its load below the *overload threshold*, L_{high}. The overload protection algorithm is based on the following simple observation: a server receives a number of requests that is proportional to the size of its responsible region. This means that a server can control the number of requests it receives by simply increasing or decreasing its responsible region.

Let L be the current utilization of the server, and let L_{low} be an *underload threshold*, where $L_{low} < L_{high}$. Then, the server updates the diameter of the responsible region, k, by using the algorithm outlined below. This algorithm is of the Additive-Increase Multiplicative-Decrease (AIMD) [CJ89,Jac88] type similar to algorithms used for congestion avoidance in computer networks.

1. Initialize $k = k_{max}$, where k_{max} is a value proportional to the server's capacity.

2. If $L < L_{low}$, then $k = k_{new}$, where k_{new} is such that $RRegion(s, k_{new}) = RRegion(s, k_{old}) +$

β. Thus, an under-loaded server increases its responsible region by an additive constant β. Assuming clients are uniformly distributed in the network space, this will also lead to an additive increase in the number of clients covered by the responsible region (Experimental evaluations in Chapter 4 show that even in the case of clients not being distributed uniformly, our mechanism of overload protection is effective).

3. If $L > L_{high}$, then $k = k_{new}$, where k_{new} is such that $RRegion(s, k_{new}) = \alpha \times RRegion(s, k_{old})$. Thus, the server decreases the size of its responsible region by a multiplicative constant $\alpha < 1$.

4. If $k > k_{max}$, then $k = \min(k, k_{max})$.

2.4 Realization of LALA on DHT-based anycast systems

In this section, we will first describe how to employ alternative location encoding techniques, which will allow us to map the network distance space into the DHT's one-dimensional identifier space. The original proposal for using postal codes as location encodings in $i3$ does not work well in this regard because the postal code framework provides only a very coarse level granularity in partioning the network space. We advocate the use of a system like Global Network Positioning (GNP) [NZ02] to assign coordinates in a d-dimensional space to each overlay node, and then use space filling curves (such as the Hilbert curve [SSK02]) to map the d-dimensional GNP coordinates to the one-dimensional ID space.

GNP uses coordinate based mechanisms to predict Internet network distance (i.e round trip propagation and transmission delay). GNP computes d-dimensional absolute coordinates for nodes by modelling the Internet as a geometric space and the inter-host distance can be calculated from the coordinates of the two hosts. GNP alleviates the problem of on-demand network measurements, which are time-consuming and costly. A space filling curve linearizes a multidimensional space by assigning each point in the space a unique value or address on the curve and still preserving locality in the multidimensional space.

That is, points that are close to each other in the multidimensional space are with high probability close to each other in the one-dimensional space. The use of GNP together with the space filling curve as the location encoding technique guarantees with high probability that two overlay nodes which are close to each other in underlying network space, will have DHT identifiers that are numerically close to each other.

We will now give a realization of the LALA algorithm described in Section 2.3 using DHTs. Consider a DHT that uses a circular ID space such as Chord [SMK+01] or Pastry [RD01] and a mapping of the underlying network space to this DHT identifier space using the GNP and space filling curves. Assume each server is associated with a number of virtual servers, and each request is assigned to the virtual server with the closest ID. In the remainder of this section, we show how one can implement the LALA algorithm in this setting by (1) choosing location-aware IDs for both virtual servers and requests, and by (2) having each server dynamically adjust the number of virtual servers as its load changes.

Assume the ID space is divided into a M equal size intervals, where M is greater than the number of servers in the system. Let id_i be the ID at the center of interval i. Next we explain how to assign IDs to virtual servers and requests. Consider a physical server s whose location-aware ID belongs to the interval centered at id_s. Let *RRegion(s,k)* be the responsible region of the anycast server s. Server s with RRegion(s,k) maintains $3k + 1$ virtual servers as follows:

- $2k + 1$ virtual servers (referred as *locationVS*) with IDs: $id_{s-k} - k, \ldots, id_{s-1} - 1, id_s, id_{s+1} + 1, \ldots, id_{s+k} + k$, and
- k virtual servers (referred as *randomVS*) with IDs *randomly* distributed in the ID space.

Similarly, consider a client c whose location-aware id belongs to the interval with center at id_r. Then every request issued by c is assigned ID id_r. We are now in a position to describe the request assignment and the overload protection algorithms. Using the ids of the locationVS of a server s, we define the RRegion(s,k) to comprise of the $2k + 1$ contiguous

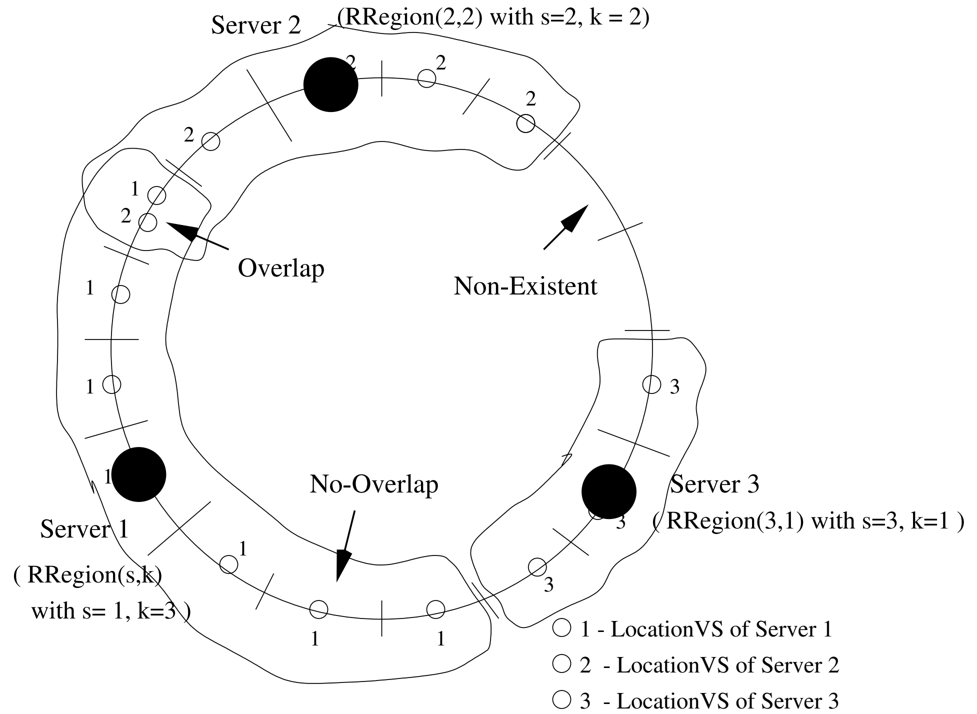


Figure 2.2 : RRegions in Circular ID Space

intervals centred around id_s in which its locationVSes are present. In particular, the server has one locationVS present in each of the 2k+1 intervals within its RRegion. Depending on the span of the respective RRegions, the request can originate in either of the three cases as shown in Fig 2.2 : *No-Overlap*, *Overlap* and *Non-Existent*. We will first describe at a high level what we want to accomplish in each of the following cases using the guidelines we developed in Section 2.3.1. In the case of no-overlap, the client request should go the server which has the locationVS present in the interval with center at id_r. In the case of overlap, more than one server has a locationVS present in the interval with center at id_r. Among these locationVSes we must be able to choose the server such that the id distance between its locationVS and its locationId id_s is minimum. In the case of non-existent, the request should be delivered to a server with a probability proportional to the size of its RRegion (Size of RRegion is proportional to k in this case). We can do this by delivering the request to a server with probability proportional to the number of their respective randomVS. We

will now describe how choosing the Ids of the locationVS and randomVS as mentioned above accomplishes what we want with regard to request assignment in each of the three cases of no-overlap, overlap and non-existent.

Request assignment Using DHT mapping, a request is simply assigned to the virtual server with the closest ID.

To see why this works consider a request with ID id_r. Let s be a physical server (if any) with the location-aware ID id_s, whose responsible region covers the interval centered at id_r. Then server s will maintain a virtual server with ID $id_r + r - s$. Among all servers that cover the interval centered at id_r the request will be delivered to server s such that $|s - r|$ is minimized. But $|s - r|$ represents exactly the distance (in number of intervals) between the client who generated the request and server s. In other words, the request is mapped on the closest server whose responsible region covers the client. This accomplishes the cases no-overlap and overlap.

In the case of non-existent (no server covers the interval centered at id_r), the request will be most likely mapped to a virtual server with a random[†] ID, which is exactly what we want. Indeed, since each server maintains a number of virtual servers (with random IDs) proportional to the size of the server's responsible region, the request will be delivered to a server with a probability proportional to the size of server's responsible region.

Overload Protection Each server executes the algorithm in Section 2.3.2. Again, assume the location-aware identifier of server s belongs to the interval centered at id_s, and assume the responsible region of s is *RRegion(s, k)*. Since in this case *RRegion(s, k)* represents an interval in the ID space, changing k reduces either to an additive increase operation ($k = k + \beta$), or a multiplicative decrease operation ($k = \alpha k$).

There are three things to note about our DHT-based algorithm. First, the algorithm is completely decentralized, and thus highly scalable. Second, as will be shown in Section 2.5, the algorithm can be trivially implemented in $i3$. Finally, as will be shown in Chapter 4,

---

[†]This is under the assumption that each interval contains at least one server with a random ID with high probability.

the algorithm provides a good approximation of the anycast functionality.

2.5 LALA in $i3$ (i3-lala)

The DHT-based solution described in Section 2.4 can be easily implemented in $i3$, which was briefly described in Section 2.1. In particular, a virtual server joining the network is equivalent to inserting a trigger with the same ID, while a virtual server leaving the network is equivalent to removing the trigger with the corresponding ID from $i3$. To compute the location encoding of a node we use the Global Network Positioning (GNP) to assign a 3-dimensional coordinate with each node. Then, we use a Hilbert curve to map these coordinates into the one-dimensional $i3$ ID space. One advantage of the $i3$ solution over the generic DHT solution is that trigger insertion and removal operations are significantly cheaper than virtual server joining and leaving operations. On the downside, with the current $i3$ implementation, all triggers belonging to the same anycast group are inserted at the same $i3$ server, which can become a bottleneck.

Chapter 3

Design of LALA on Tree-based anycast systems like Scribe

In this chapter, we outline the design of the LALA architecture on tree-based anycast systems like Scribe. We first give some background on how anycast is done in Scribe and discuss its limitations. We then identify how LALA can be designed on such an architecture. Finally, we show how our design can be implemented on Scribe.

3.1 Background: Anycast in Scribe

The Scribe anycast functionality is built on Pastry. We will briefly describe Pastry and Scribe and describe how to do anycast using Scribe.

3.1.1 Pastry

Pastry is a scalable, self-organizing structured peer-to-peer overlay network similar to CAN [RFH$^+$01], Chord [SMK$^+$01], and Tapestry [ZKJ01]. In Pastry, nodes and objects are assigned random identifiers (called *nodeIds* and *keys*, respectively) from a large id space. NodeIds and keys are 128 bits long and can be thought of as a sequence of digits in base 2^b (b is a configuration parameter with a typical value of 3 or 4). Given a message and a key, Pastry routes the message to the node with the nodeId that is numerically closest to the key, which is called the key's *root*. This simple capability can be used to build higher-level services like a distributed hash table (DHT) or an application-level group communication system like Scribe.

In order to route messages, each node maintains a routing table and a leaf set. A node's routing table has about $log_{2^b} N$ rows and 2^b columns. The entries in row r of the routing table refer to nodes whose nodeIds share the first r digits with the local node's nodeId.

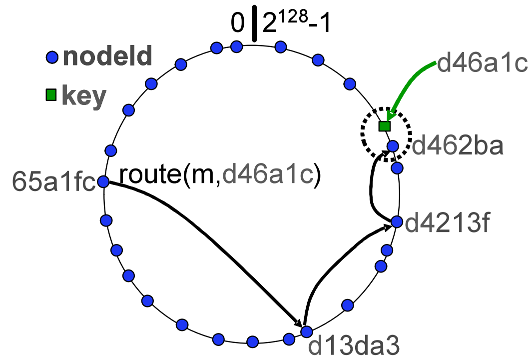


Figure 3.1 : Routing a message from the node with nodeId $65a1fc$ to key $d46a1c$

The $(r+1)$th nodeId digit of a node in column c of row r equals c. The column in row r corresponding to the value of the $(r+1)$th digit of the local node's nodeId remains empty. At each routing step, a node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit longer than the prefix that the key shares with the present node's id. If no such node is known, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node's nodeId but is numerically closer. Figure 3.1 shows the path of an example message between two nodes. The dots in the figure depict the nodeIds of live nodes in Pastry's circular namespace.

Each Pastry node maintains a set of neighboring nodes in the nodeId space (called the leaf set), both to ensure reliable message delivery, and to store replicas of objects for fault tolerance. The expected number of routing hops is less than $log_{2^b} N$. The Pastry overlay construction observes proximity in the underlying Internet. Each routing table entry is chosen to refer to a node with low network delay, among all nodes with an appropriate nodeId prefix. As a result, one can show that Pastry routes have a *low delay penalty*; i.e. the average delay of Pastry messages is less than twice the IP delay between source and destination [CDHR02]. Similarly, one can show the *local route convergence* of Pastry routes; i.e. the routes of messages sent to the same key from nearby nodes in the underlying Internet tend to converge at a nearby intermediate node. Both of these properties are important for the construction of efficient multicast/anycast trees, described below. A full description

of Pastry can be found in Rowstron et al. [RD01] and Castro et al. [CDHR02].

3.1.2 Scribe

Scribe [CDKR02, CDKR03] is an application-level group communication system built upon Pastry. A pseudo-random Pastry key, known as the *groupId*, is chosen for each multicast group. A multicast tree associated with the group is formed by the union of the Pastry routes from each group member to the groupId's root (which is also the root of the multicast tree). Messages are multicast from the root to the members using reverse path forwarding [DM78].

The properties of Pastry ensure that the multicast trees are efficient. The delay to forward a message from the root to each group member is low due to the low delay penalty of Pastry routes. Pastry's local route convergence ensures that the load imposed on the physical network is small because most message transmission occurs at intermediate nodes that are close in the network to the leaf nodes in the tree.

Group membership management in Scribe is decentralized and highly efficient, because it leverages the existing, proximity-aware Pastry overlay. Adding a member to a group merely involves routing towards the groupId until the message reaches a node in the tree, followed by adding the route traversed by the message to the group multicast tree. As a result, Scribe can efficiently support large numbers of groups, arbitrary numbers of group members, and groups with highly dynamic membership.

The anycast primitive was recently added to Scribe. Castro et al. [CDKR03] describes in detail how to perform distributed resource discovery using Scribe. Tree-based anycast systems like Scribe [CDKR03] represent an anycast group by a location-based spanning tree embedded in a structured overlay network that uses proximity neighbor selection. Anycast is performed by routing a message in the overlay from the client towards the root, which intersects the tree at a node that is near the client in the network. From there, the message is forwarded in a depth-first search of the tree, until a group member is reached.

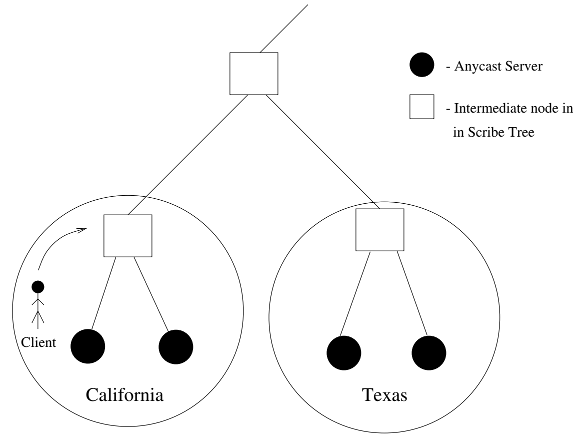


Figure 3.2 : Responsible regions in Tree-based anycast system

3.2 Responsible region (RRegion) in Scribe

As mentioned in Section 1.3, the concept of a responsible region has different representations in different anycast architectures. In this section, we will try to identify the representation of a responsible region in a tree-based anycast system like Scribe.

Since the Scribe tree is built in a locality-aware fashion, members in a particular network region will with high probability have the same intermediate node as parent in the tree. Depending on the location of the client making the anycast query, the request hits with high probability an intermediate node in the tree whose subtree consists of members that are close to the client. Figure 3.2 shows such a scenario where an anycast request from a client in California hits with high probability the intermediate node whose children are members in California. Therefore, anycast servers that are close to each other in network proximity will with high probabality belong to the same lowest level subtree. In this architecture therefore, the responsible region of an anycast server is defined by the lowest level subtree in which the anycast server resides. All anycast servers residing in the same subtree share the same responsible region. Moreover, the entire geographic space is approximately partitioned into the responsible regions corresponding to the lowest level subtrees.

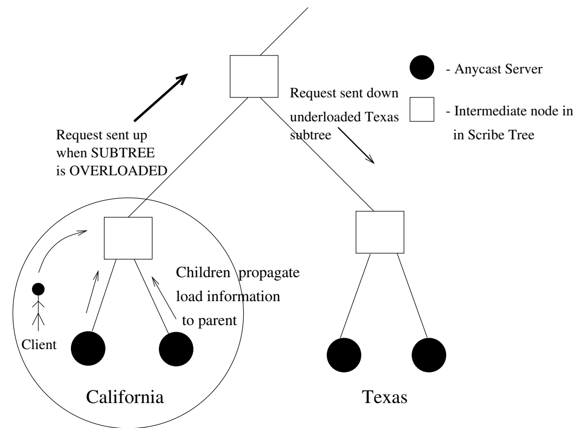


Figure 3.3 : Adjusting size of RRegions in Scribe

3.3 LALA Algorithm

Before we describe the design of LALA in the tree-based anycast system, we make the following observation. As we go higher up each level in the tree, the responsible regions of the lower level subtrees can now be collapsed to represent a larger geographic space. In Figure 3.2 when we go up one level in the tree, the RRegions of California and Texas can be collapsed to represent a larger region representing U.S.A.

We will now describe a way in which servers can update the size of their responsible regions. Note that the anycast request first hits an intermediate node in the tree and from there is sent down to the children. In the scenario when the anycast servers in a particular subtree are overloaded, the parent of the nodes in that subtree could be requested to send the anycast request up the tree instead. The anycast request would therefore go up one level in the tree and could then be sent down another sibling subtree. As shown in Figure 3.3, in the scenario when anycast servers in the California region are overloaded, a client request originating from California is sent up one level in the tree and then sent down to some anycast server in the Texas subtree (assuming that the Texas subtree of servers is underloaded). Effectively, California anycast servers will have shrunk their responsible region. On the contrary, the anycast servers in Texas subtree now receive anycast requests

from Texas as well as California, thereby effectively increasing the size of their responsible region.

Having given the intuition of the mechanism to adjust responsible regions in tree-based anycast systems, we are now in a position to describe how client requests are assigned to servers. Consider a request issued by client c, which hits a intermediate node in the tree. We have two cases:

1. If the subtree attached to this intermediate node is not overloaded, the request is sent down its children with probability $P(s)$, where $P(s)$ is proportional to the aggregate capacity of the anycast servers in the subtree hanging beneath the child. Thus within a subtree, high capacity children receive a greater proportional share of the anycast requests hitting the intermediate node, as compared to low capacity children. To achieve this effect, the anycast servers monitor their own load and run the AIMD style overload protection algorithm described in Section 2.3.2, where the k value in the algorithm now corresponds to the server's willingness to accept a request from its parent. Thus, within a subtree, low capacity members have lower k values than high capacity sibling members in the same subtree. Children propagate their k values to their parent. Aggregated load information in the form of k values is propagated up the tree, all the way up to the root.

2. If the subtree attached to this intermediate node is overloaded, the request is sent up one level in the tree. The request then arrives at the parent of this intermediate node where the same two cases are then checked for and corresponding action taken.

We however need to be able to determine if the subtree beneath an intermediate node is overloaded or not, to be able to take the corresponding action as per defined in the two cases above. An interior node in the tree calculates its own k value, which is representative of the load in its subtree, by averaging the k values of its children (In future, we plan to study the use of other strategies like using median etc.). When the k value at the parent node is less than a threshold (we use $0.05 * k_max$), the entire subtree is considered overloaded.

When all the children in a particular subtree are overloaded thereby having low k values, the parent node will also have a low calculated average k value.

We point out the following observations of the above algorithm: (1) Anycast servers that lie within the same subtree receive requests in proportion to their capacities, although they share the same responsible region. This differs from the LALA algorithm for DHT based anycast system where anycast servers always received requests in proportion to the size of their responsible regions. This anomaly arises from the fact that in the tree-based anycast system, the responsible regions are defined at a more granular level, the granularity depending on the number of levels in the tree. As a result of this we needed a mechanism where different capacity servers, inspite of having the same responsible region, could control the amount of traffic they received from that region. (2) Whenever an anycast message is forwarded up the tree due to overload, it tends to move toward groups members that are farther and farther away from the client in the network. However, this only happens if and when more nearby group members are overloaded. This property of the algorithm is in synergy with the goal of LALA's anycast functionality i.e., select the closest anycast server that is not overloaded.

3.4 LALA in Scribe (Scribe-lala)

The LALA algorithm describe above can be trivially implemented in the Scribe architecture. The propagation of load information up the tree can be piggy-backed on the periodic Scribe tree maintenance traffic as well as control traffic sent in the tree traversal during the delivery of the anycast request.

Chapter 4

Evaluation

In this chapter, we evaluate our design of LALA architecture on the DHT-based and tree-based anycast systems. In particular, we evaluate the implementations of our design on $i3$ (**i3-lala**) and Scribe (**scribe-lala**) and show that our design is successful in realizing LALA's anycast functionality: select the closest anycast server that is not overloaded.

We evaluate **i3-lala** and **scribe-lala** against the original mechanism of doing only location-aware anycast (without overload protection) in $i3$ and Scribe, referred to as **i3-simple** and **scribe-simple**. These approaches resolve each anycast request to a nearby server while ignoring load. *i3-simple* is the approach where we use only one trigger per anycast server, which encodes the location of the server. The anycast request which contains the location encoding of the client is resolved to the server whose trigger is numerically closest to the anycast request's encoding. The location encoding technique is the same as that of *i3-lala* (i.e using GNP and Hilbert curves). With *simple-scribe*, an anycast request is resolved to a nearby member in the anycast tree [CDKR02]. In this approach, the anycast request hits an intermediate node in the Scribe tree and is then sent down a randomly chosen child. Note that the *i3-simple* and the *scribe-simple* approaches could be modified with mechanisms of doing coarse grained overload protection in the form of removing the trigger when the server is overloaded in the $i3$ approach or the server unsubscribing from the anycast group in the case of Scribe. These mechanisms however, incur high network traffic overhead resulting from repeated trigger insertions/removals or group subscribe/unsubscribe operations even in the case where the anycast traffic follows a steady pattern over time. Moreover these coarse grained overload protection algorithms are stable only at low utilizations values. Since the *i3-simple* and the *scribe-simple* approaches ignore load and try to resolve the

anycast request to the closest anycast server, they serve as our baseline comparison with regard to the ability of selecting a nearby anycast server.

4.0.1 Simulation settings

We consider an overlay network consisting of $10,000$ nodes and an anycast group consisting of N $(= 256)$ randomly chosen nodes in the network. To generate the network, we used the GT-ITM topology generator [Geo]. The server capacities are uniformly distributed in the interval [MIN-CAP, 8*MIN-CAP], and the request sizes are uniformly distributed in the range [MIN-SIZE, 20*MIN-SIZE]. The client issuing the request is randomly selected from the nodes in the network and the arrival times of the requests follow a Poisson distribution.

We simulate a web like application, where the load of a server is measured as the average size of its request queue. Each server uses thresholds $L_{low} = 10$, and $L_{high} = 150$, respectively. As described in Section 2.3.2, each server does overload protection using an AIMD algorithm with the following parameters, $\beta = 2$, $\alpha = \alpha' * server_capacity/demand_rate$, where $\alpha' = 0.9$, and $demand_rate$ represents the service rate requested by the incoming requests. Note however that this AIMD algorithm is slightly different to that proposed in Section 2.3.2, since α is not a constant. In our simulations, we found this slightly modified version of AIMD to be significantly more stable. In future, we plan to theoretically study the stability of the system and derive the AIMD parameters.

4.0.2 Evaluation metrics

We use two metrics to evaluate our schemes:

- **Maximum waiting time** The maximum time a request waits in the server's queue across all the requests in the system. This metric evaluates the ability of a scheme to provide overload protection. Note again, that this time does NOT include the network delay associated with getting the anycast request packet delivered to the chosen anycast server.

- **Weighted rank**. This metric is computed as $\Sigma_{i=1}^{i=N} i * frac(Rank = i)$, where $frac(Rank = i)$ represents the fraction of anycast requests that are resolved to servers with rank i. The *rank* of a server is defined with respect to each request r, and represents the index of the server in the list of all servers ordered by their distance from the client that has issued r. Thus, given a client c that issues a request r, the closest anycast server from c has rank 1, the second closest anycast server from c has rank 2, and so on. This metric evaluates the accuracy of a scheme to select the closest anycast server. Ideally, ignoring the server overload, each request should be delivered to the closest server, in which case the weighted rank is 1.

As mentioned above, the *maximum waiting time* metric evaluates the ability to provide overload protection and the *weighted rank* metric evaluates the ability to select nearby anycast servers. Both metrics together, evaluate the effectiveness of our mechanisms in realizing LALA's anycast functionality (i.e. selecting the closest anycast server that is not overloaded). In the future, we also plan to evaluate our mechanisms with respect to the *total response time* of the anycast requests. The total response time of the request has two components - (1) network delay associated with sending and receiving of the anycast request and reply between the client and the selected anycast server; and (2) the waiting time of the anycast request in the selected anycast server's queue. Based on the assumptions stated in Section 1.2, realizing the anycast functionality would imply optimizing with regard to the total response time metric also. We plan to evaluate the total response time metric when we have the prototype of our system ready for deployment on the Planetlab testbed.

4.0.3 Simulation Methodology

To evaluate our schemes, we vary the utilization U of the system between 0.01 and 1. U is defined as the ratio of (1) the total amount of service requested by all clients to (2) the aggregate capacity of all servers in the anycast group. For each utilization value, we run the simulation long enough for the system to stabilize. The stability of the system at a

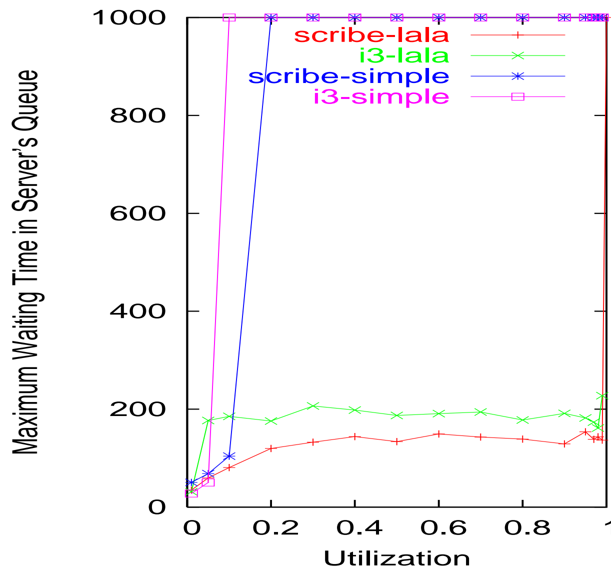


Figure 4.1 : Maximum waiting time vs. utilization

particular utilization value is judged on the basis of the average waiting time of the anycast requests with simulation time. A monotonic increase in the average waiting time of anycast requests with simulation time suggests that the system did not stabilize for that utilization value. Note that for values of U close to 1 the system does not stabilize. If the system does not stabilize, we report a maximum waiting time of $1,000$.

4.0.4 Simulation Results

Figures 4.1 plots the maximum waiting time versus the system utilization. Even for utilizations as low as 0.1, *i3-simple* and *scribe-simple* exhibit large maximum waiting times. This is to be expected since both these schemes ignore the server load when resolving the requests. The primary reason for this is that the anycast servers have heterogeneous capacities. Moreover, as we observed in some of our unreported experimental results, even in a group with homogeneous server capacities, these approaches result in server overloading since the servers and clients could potentially be scattered non-uniformly in the geographic space, thereby resulting in some servers requiring to handle a greater fraction of the anycast

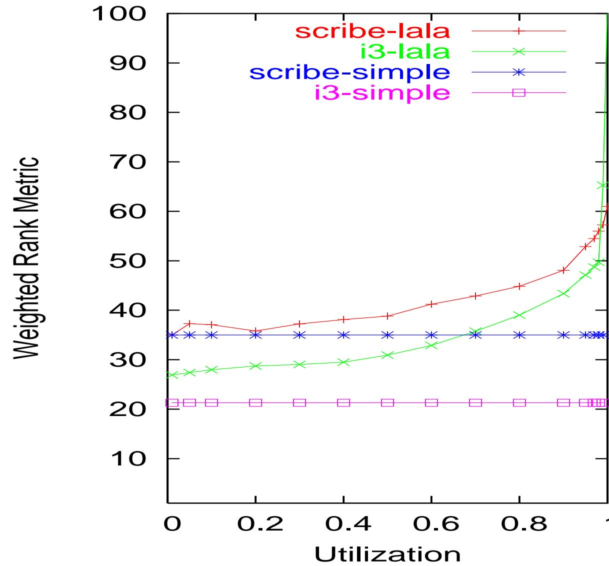


Figure 4.2 : Weighted rank metric vs. utilization

requests. Additionally, flashcrowd-like effects can severely result in server overloading. This motivates the need for overload protection in locality-aware anycast architectures.

As shown in Figure 4.1, *i3-lala* and *scribe-lala* are successful in preventing overload even for very high utilization values. Between the two schemes *scribe-lala* performs better than *i3-lala*. This is because the overload protection algorithm in *scribe-lala* is more localized, involving only members in the same anycast group subtree. In *i3-lala* however, the feedback mechanism essentially operates over the entire anycast group together and is therefore more susceptible to fluctuations resulting from over-reacting.

Figure 4.2 plots the average *weighted rank* metric as a function of the system utilization. Note that a weighted rank value of 1 corresponds to all requests being resolved to the closest server, a value of N $(= 256)$ corresponds to all requests being resolved at the farthest server, and a value of $N/2$ $(= 128)$ corresponds to selecting a random server to resolve each request. The weighted ranks of the baseline schemes, *scribe-simple* and *i3-simple*, are flat lines. Again, this is because none of these schemes take into account the server load.

As shown in Figure 4.2, for lower values of utilization, the weighted ranks of our schemes (*i3-lala* and *scribe-lala*) are relatively close to their baseline counterparts. This is because, at lower utilization values, the need for aggressive overload protection does not arise and therefore our schemes perform as good as their baseline counterparts. With increasing utilization, the weighted rank metric increases. Also observe that the rate of increase in the weighted rank metric of *i3-lala* with utilization is more than the rate of increase in the case of *scribe-lala* and this is because of the previously mentioned fact that distributed overload protection works less robustly in *i3-lala* as compared to *scribe-lala*. Even for utilizations as high as 0.9 the weighted rank of the *i3-lala* is only about two times larger than the weighted rank of *i3-simple*, while the weighted rank of *scribe-lala* is less than 40% larger than the weighted rank of *scribe-simple*.

Also observe in Figure 4.2, that as expected, the average weighted ranks of *scribe-lala* and *scribe-simple* are basically the same for very low utilizations, this is not the case for the *i3* schemes. This results from the fact that in our *i3-lala* implementation we partition the ID space in $M = 512$ intervals and the requests are approximated as originating from the center of those intervals. By increasing the number of intervals, the weighted rank of *i3-lala* approaches the weighted rank of *i3-simple* for small system utilizations. In summary, Figures 4.1 and 4.2 show that LALA successfully prevents server overload, while at the same time resolving requests to nearby servers.

We also experimented with a more skewed request distribution to simulate flash-crowd scenarios. That is, instead of generating requests from randomly chosen clients in the underlying network topology, we generated 50% of the requests from clients lying in a hot-spot zone which represents only 20% of the total network space. Both our schemes *i3-lala* and *scribe-lala* were successful in preventing server overloading, but, as expected, they exhibited slightly higher weighted ranks. This is depicted in Figures 4.3 and 4.4.

We also evaluated the system for varying sizes of the anycast group, while keeping the utilization U to be constant at 0.5 and generating anycast requests from a randomly chosen client. Note that the earlier experiments were performed for N=256. In this experiment,

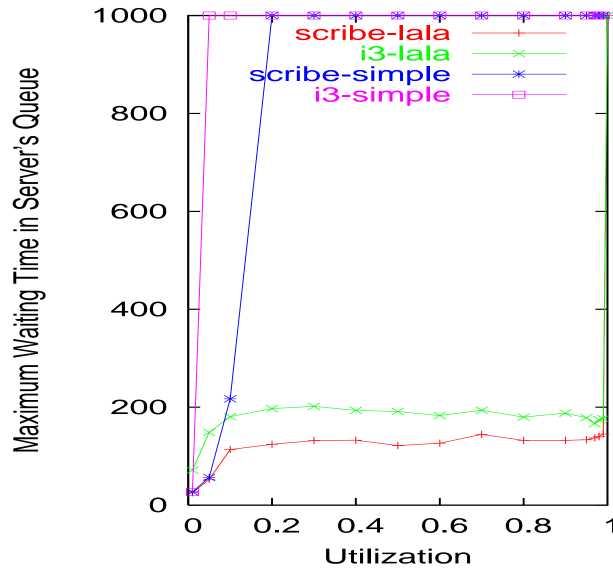


Figure 4.3 : Maximum waiting time vs. utilization : Flashcrowd scenario

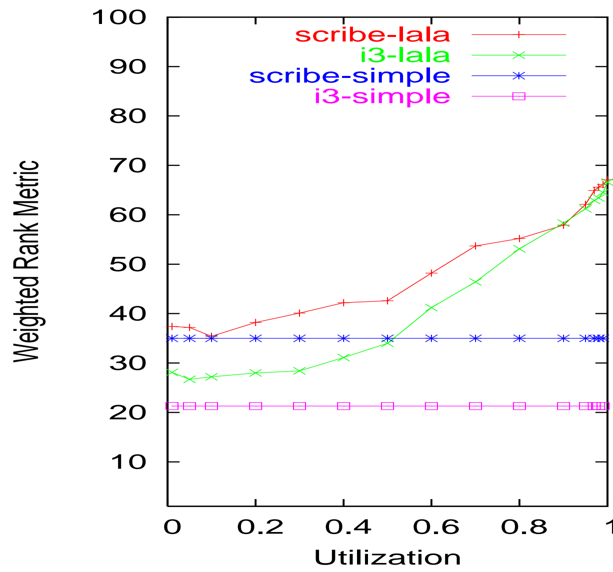


Figure 4.4 : Weighted rank metric vs. utilization : Flashcrowd scenario

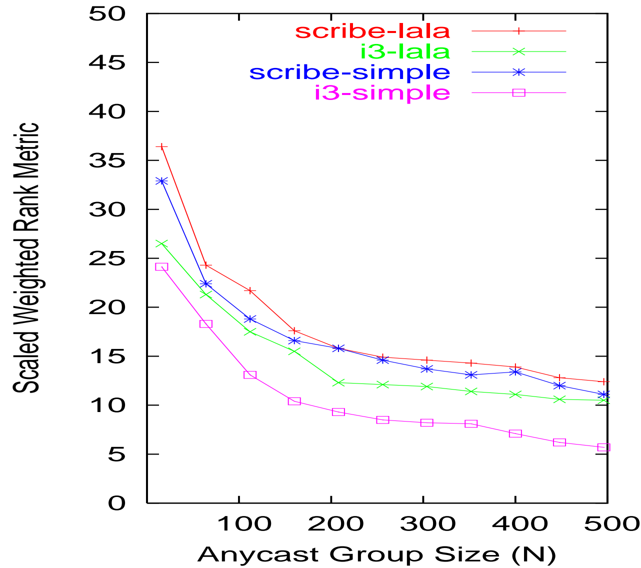


Figure 4.5 : Scaled weighted rank metric vs. Anycast Group Size

we vary N between 16 to 496. For the purpose of this experiment we propose a *Scaled Weighted Rank* metric, which is defined as $(WeightedRankMetric/N) * 100$. Basically, this metric evaluates the effect of varying anycast group sizes on the ability of different approaches in locating a close anycast server. For instance, in the experiments performed earlier with N=256, the weighted rank metric for the *i3-simple* and *scribe-simple* approach were 21.3 and 35.0 respectively. This translates to respective *scaled-weighted-rank* metrics of 8.3 and 13.7 respectively when the group size is 256.

As shown in Figure 4.5, we observed that the *scaled-weighted-rank* metric of both the *scribe-simple* and *i3-simple* approach improve with increasing size of the anycast group for the following reasons. In *scribe-simple*, increasing group sizes cause the Scribe tree to be built in a more locality-aware fashion, which result in more effectively locating a close anycast server. In *i3-simple*, the ability to locate close anycast servers using the GNP/Hilbert technique improves when the mean geographic distance between a randomly chosen client and its closest anycast server decreases. This is because, when two nodes are relatively close to each other in the underlying topology, the relative error in mapping their

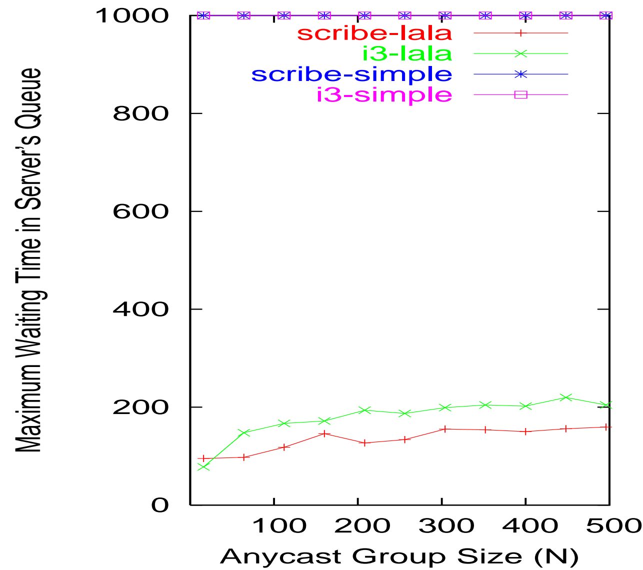


Figure 4.6 : Maximum waiting time vs Anycast Group Size

physical location to their corresponding location encoding using the GNP/Hilbert technique is less. Figure 4.5 also shows in corroboration to our earlier experiments that the scaled weighted rank metrics of *i3-lala* and *scribe-lala* are only slightly greater than the values for their corresponding baseline counterparts *i3-simple* and *scribe-simple* when the utilization U is low or medium (U=0.5 in this case).

Figure 4.6 shows the effect of increasing group sizes on the ability to do server overload protection. Figure 4.6 corroborates our earlier results that *i3-simple* and *scribe-simple* approaches are not able to prevent server overloading even at medium values of utilization. (Observe the reported maximum waiting time as $1,000$ for both these approaches for all group sizes). The *i3-lala* and the *scribe-lala* approach are however able to bound the maximum waiting time approximately within the desired threshold. However, as observed and mentioned in our earlier experiments, *scribe-lala* performs better than *i3-lala* in the ability to prevent server overloading. This is clear from the fact that the maximum waiting time at server's queue increases at a greater rate with group size for the *i3-lala* approach as compared to the *scribe-lala* approach.

Before concluding this chapter, we would like to mention that the results in this chapter only serve the purpose of showing that the anycast functionality can be realized in the DHT-based system like $i3$ as well as the tree-based anycast system like Scribe. The results of the $i3$ approach and the Scribe approach should not be compared as of now. The experimentation done so far is in no way comprehensive to allow interpreting one type of anycast system to be superior over the other. See Chapter 6 for the different experiments that still need to be performed.

Chapter 5

Background and Related Work

Resource discovery has been the subject of extensive research for more than a decade. The problem is to find the desired resource over the network by addressing the resource or service with a name or property that characterizes the type of resource. Several initial solutions were provided like broadcasting the client's request to all possible locations where the resource might reside (e.g [BAA95, OD83]), in the hope that one of the servers having the resource would respond. An alternative to using broadcast was to send the client request to a name server [Moc87, GS88] in order to lookup the location of the resource.

There are a number of situations in networking where a host or user wishes to locate a host that supports a particular type of service [Gut99], but does not particularly care which server amongst the group of servers supporting the service is used. Researchers came up with the notion of anycast to simplify the process of resource discovery or service discovery. Anycast was first proposed in RFC 1546 [PMM93]. The document proposes anycast as a means of service discovery and describes a way of accomplishing it at the network IP level. In particular, a host transmits a datagram to an anycast address and the internetwork is responsible for providing best effort delivery of the datagram to at least one, and preferably only one of the servers that accept datagrams for the anycast address. It also points out the challenges faced in deploying IP anycast. The alleged unscalability of IP anycast has limited its acceptance by the community. GIA [KW00] is an architecture for doing global IP anycast in a more scalable fashion but it however retains some of the drawbacks of network level approaches: i.e., it requires router modifications and it cannot exploit application-level metrics to guide the process of server selection.

To alleviate these problems, several proposals for providing an anycast service at the

application layer [FBAZ98, FBAZ98, FJP+99, MDZ99, SSK97] were made. Most of these approaches attempt to build a directory system which when queried with a client address and the service name, returns the address of the nearest server to the client that provides the service. These directory based application-layer anycast systems are not scalable for the reasons mentioned below. Two types of information needs to be collected in this type of directory system : (1) Information of which servers provide a particular service and which of them are currently up; and (2) the network distance between the client and these available servers in order to be able to determine the closest server. Collecting this type of information incurs repeated probing of the servers for their availability and collecting up-to-date estimates of the network distance between hosts. This makes the directory the bottleneck and limits the scalability of such a system. Although not scalable, the application layer anycast systems however made the fundamental observation that it was not sufficient to only locate the closest server. In particular, the server selection procedure should at least be guided by both the information about the proximity of the servers from the client as well as the server load. This is because, under scenarios of a huge demand of resources from a particular region of the network, resolving the anycast request to the closest anycast server would result in overloading that anycast server.

Several proposals [CC95, SBSV98, CC97, GECZ99] for doing efficient server selection were made. The algorithms for server selection in these systems use a combination of metrics to determine the best server to serve the client's anycast request. These systems employ measurements from clients. In the work by Carter et al. [CC95, CC97], the selection is based primarily on the characteristics of the path leading from the client to the server. The authors acknowledge the desirability of using server load information as a guide to server selection. In the work of Sayal et al. [SBSV98] and Karaul et al. [KKO97], server load is incorporated into the client-side selection.

Active anycast [MYNI00] proposes to handle IP anycast in a location-aware and load-aware manner. The solution relies on active networking technology [TW96, TSS+97]. In particular, it relies on active routers maintaining load information about the anycast servers

lying in their network proximity. This proposal however suffers from the drawback that the anycast is implemented at the IP level and thus requires router modifications.

Recently, scalable overlay anycast services like $i3$ [SAZ+02], Scribe [CDKR03] and Tapestry [ZKJ01] have been built on structured peer-to-peer(p2p) overlays [RFH+01, SMK+01, RD01, ZKJ01]. The scalability of these anycast services is attributed to the scalability of their underlying p2p substrates, which have recently gained popularity as a platform for building scalable, self-organizing, decentralized applications. The anycasting paradigm henceforth, started being viewed as a powerful building block for managing and locating resources in large-scale decentralized systems.

Section 3.1 briefly describes the anycast architecture for Scribe [CDKR03] and reasons why it is location-aware but not load-aware. Like Scribe, Bayeux [ZZJ+01] and CAN-Multicast [RHKS01] are group communication systems built on top of structured p2p overlays of Tapestry [ZKJ01] and CAN [RFH+01] respectively. Although these currently do not support the anycast primitive, they could be modified to support the anycast primitive. Like Scribe, they belong to the class of tree-based anycast systems and are not location-aware and load-aware for the same reasons that were applicable to Scribe. The object location approach used in Tapestry [ZKJ01] and Plaxton's work [PRR97] can be viewed as a special case of the anycast mechanism used in Scribe. In these systems, pointers to the nearest replica holder is maintained at every intermediate node of the the tree that is formed by the overlay routes from each replica holder of an object to the object's root. Lookup messages which, when routed to the object's root, intercept the tree at an intermediate node and use the replica pointer at that node to fetch a copy of the object. These anycast services, although scalable have their own limitations. In particular, they were either location-aware or load-aware but not both.

The Internet Indirection Infrastructure ($i3$) [SAZ+02] proposes a generic indirection mechanism supporting unicast, multicast, anycast and mobility. $i3$ uses triggers to represent indirection points between senders and receivers. A trigger is represented by a unique Id. Receivers subscribe to triggers and senders publish to triggers. Section 2.1 describes how

anycast is done in $i3$ and also points out its limitations in being location-aware or load-aware but not both.

Rao et al. [RLS$^+$03] motivate the need for load balancing mechanisms in heterogeneous p2p systems. It explores some load balancing algorithms using the notion of virtual servers. The algorithms revolve around transferring virtual servers from a heavily loaded physical node to a lightly loaded physical node. Their schemes rely on nodes doing probing to find out other heavily or lightly loaded nodes, or by using directories that store load information about a set of lightly loaded nodes in the system. In our approach however, we advocate that doing overload protection instead of fine-grained load balancing (which was the goal of Rao et al.) is sufficient. The benefit of this relaxed constraint is that the server can do overload protection by simply monitoring its own load and controlling the number of virtual servers it has without requiring any form of probing of load on other servers. Adjusting the number of virtual servers based on a server's own load, is fundamentally a feedback-driven approach to overload management.

The notion of feedback-driven approach to overload management has been prevalent in several systems. Welsh et al. [WC02] make the case that overload management should be a critical design goal for Internet-based systems and services. They argue that the right approach to overload management is to explicitly signal overload conditions using feedback-driven control. Based on this observation, they sketch the design of their SEDA architecture [WCB01] which is designed to provide adequate primitives for managing load in busy Internet services. The feedback-control approach to overload management can also be seen in most of the congestion avoidance algorithms [Jac88, CJ89, FJ93] in computer networks.

In our thesis, the first step was towards identifying the goal of a location-aware load-aware (LALA) anycast architecture. The question here is how to go about optimizing both the metrics of location as well as load in our server selection process. Choosing lightly loaded servers could result in resolving client requests to a server located far away from the client. Similarly, resolving client requests to nearby servers would very likely result in server overloading in scenarios of client requests being generated unevenly in the

geographic space. Amongst the various possible strategies that could be devised to strike the correct balance between the two metrics, we take a stance similar in nature to that taken by LARD [PAB+98]. LARD, a framework for doing content based request distribution in cluster-based network servers also faced a similar problem in which the front-end of network servers had to direct the incoming requests to one of the back-ends in a way to optimize performance with regard to load on the back-ends and locality in the back-ends main memory caches. LARD resolves this problem by resolving requests by observing locality and ensuring only that the back-ends are not overloaded instead of doing perfect load balancing. Motivated by LARD's way of optimizing between two metrics, we decided to resolve anycast requests to closeby anycast servers as long as they were not overloaded. We thereby defined LALA's anycast functionality as resolving the client request to the closest anycast server that is not overloaded.

Chapter 6

Conclusions and Future Work

In this work, we make the following contributions. First, we motivate the need for a location-aware and load-aware (LALA) anycast architecture and then define the goal of such an architecture. In particular, the LALA architecture aims to achieve the following anycast functionality: given a client request, select the closest anycast server to the client that is not overloaded. We also develop the concept of self-adjusting reponsible regions as being the central idea in designing the LALA architecture. Second, we investigate the current overlay anycast systems and identify two classes of anycast systems which differ radically in their mechanism of doing scalable anycast. The two classes of anycast systems are : DHT-based anycast systems like $i3$ and tree-based anycast systems like Scribe. Thirdly, we investigate mechanisms in which we could make these types of anycast systems location-aware and load-aware by employing different manifestations of LALA's central idea of self-adjusting responsible regions. Finally, we evaluate the performance of LALA on $i3$ and Scribe. Simulation results employing GATech's Transit-Stub topology model showed the effectiveness of our proposed architecture in realizing the anycast functionality in both DHT based anycast systems like $i3$ and tree based anycast systems like Scribe. To the best of our knowledge this is the first proposal and design for a scalable, location-aware, load-aware anycast architecture for decentralized distributed systems.

We are currently investigating the overhead associated in implementing LALA on $i3$ and Scribe. For instance, in the $i3$ approach, we want to quantify the overhead associated with trigger insertions/removal. We also want to quantify the overhead resulting from the network traffic resulting from the fact that the overlay nodes need to compute their Global Network Positioning (GNP) coordinates. In the Scribe approach, we want to quantify the

overhead in propagating load information up the tree. We also want to quantify the network traffic that would result from the fact that we keep the underlying overlay substrate locality-aware while building the routing tables of the overlay nodes using proximity neighbour selection (PNS).

We are also investigating ways in which we could make the overload protection algorithm more stable. Our current overload protection algorithm is not stable in the presence of long-tailed distributions of the sizes of anycast requests. We plan to theoretically study the stability of the system and derive the AIMD parameters for the overload protection algorithm. We are also investigating the use of other overload protection algorithms which need not be of the AIMD type. We also intend to study how good our overload protection algorithm works in more dynamic settings. For instance, when the load on a server rises above the threshold, it decreases the number of virtual servers. Depending on how long the virtual server adjustment takes, the load on the server could potentially rise beyond its desired value. In particular, how agile should an anycast server be in doing feedback-driven control in the overload protection algorithm.

We are currently working on evaluating our architecture on the Planetlab testbed. We expect the testbed results to quantify the overhead costs of the implementing LALA on $i3$ and Scribe. We also hope to get insights on how agile the anycast servers need to be with regard to preventing overload but the same time keeping the overhead traffic to acceptable limits.

We started looking into potential applications which could be built on this type of architecture. We have identified two such applications : (1) Imagine that you have a content distribution network like Akamai, where the client request is dynamically redirected to an appropriate server. The chosen server in such an environment is usually located close to the client. However, in the event of a flashcrowd of requests from a particular region, the servers could potentially become overloaded. In such an environment, we could employ LALA to ensure that the chosen server is close to the client as well as is not overloaded. (2) Imagine, an Internet scale Grid Computing platform. The servers in this platform share

their resources to perform compute-intensive tasks initiated by other clients. When a client submits a job request, the architecture tries to locate a server which has resources (CPU cycles) and is reasonably close to the client. The requirement of the server in being close to the client is that when the task is completed, potentially large amounts of trace files might need to be shipped to the client. In order to keep the underlying network usage to a minimum we should be able to locate compute-servers which are close to the client and which at the same time have sufficient resources to serve the client.

Bibliography

[BAA95] J. Bernabeu, M. Ammar, and M. Ahamad. Optimizing a generalized polling protocol for resource finding over a multiple access channel. In *Computer Networks and ISDN Systems, vol. 27, pp. 1429-1445*, 1995.

[BAZ+97] S. Bhattacharjee, M. Ammar, E. Zegura, N. Shah, and Z. Dei. Application layer anycasting. In *In Proceedings of IEEE Infocom'97*, 1997.

[CC95] M. Crovella and R. Carter. Dynamic server selection in the internet. In *Proceedings of Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95)*, August 1995.

[CC97] R.L. Carter and M.E. Crovella. Server selection using dynamic path characterization in wide-area networks. In *Proceedings of IEEE Infocom 97*, 1997.

[CDHR02] M. Castro, P. Druschel, Y.C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks, 2002. Technical report MSR-TR-2002-82.

[CDKR02] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), October 2002.

[CDKR03] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. Scalable application level anycast for highly dynamic groups. In *Proc. NGC'2003*, Munich, Germany, September 2003.

[CJ89] D. Chiu and R. Jain. Analysis of increase/decrease algorithms for congestion

avoidance in computer networks. *Journal of Computer Networks*, 17(1), June 1989.

[DKK+01] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. ACM SOSP'01*, Banff, Canada, October 2001.

[DM78] Y.K. Dalal and R. Metcalfe. Reverse path forwarding of broadcast packets. *CACM*, 21(12):1040–1048, 1978.

[DZD+03] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proceedings for the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, California, February 2003.

[FBAZ98] Z. Fei, S. Bhattacharjee, M. Ammar, and E. Zegura. A novel server technique for improving the response time of a replicated service. In *Proc. IEEE Infocom'98*, 1998.

[FJ93] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. In *IEEE/ACM Transactions on Networking, Vol 1 No.4, p. 397-413*, August 1993.

[FJP+99] P. Francis, S. Jamin, V. Paxson, L. Zhang, D.F. Gryniewicz, and Y. Jin. An architecture for global internet host distance estimation service. In *Proc. INFOCOM 1999*, New York, NY, USA, March 1999.

[GECZ99] M.L. Gullickson, C.E. Eichholz, A.L. Chervenak, and E.W. Zegura. Using experience to guide web server selection. In *Multimedia Computing and Networking*, January 1999.

[Geo] Geogia Tech Internet topology model. `http://www.cc.gatech/fac/Ellen.Zegura/graphs.html/`.

[Gnu00] The Gnutella protocol specification, 2000. `http://dss.clip2.com/GnutellaProtocol04.pdf`.

[GS88] I. Gopal and A. Segall. Directories for networks with casually connected users. In *Proceedings of IEEE Infocomm 88 pp. 1060-1064*, 1988.

[Gut99] E. Guttman. Service location protocol: Automatic discovery of ip network services. In *IEEE Internet Computing, 1999 4(4)p. 71-80*, 1999.

[Jac88] V. Jacobson. Congestion avoidance and control. In *Proc. ACM SIGCOMM'88*, September 1988.

[KKO97] M. Karaul, Y. Korilis, and A. Orda. Webseal: Web server allocation. Technical Report TR1997-752, New York University, 1997.

[KW00] D. Katabi and J. Wroclawski. A framework for scalable global ip-anycast (gia). In *Proc. ACM SIGCOMM'00*, 2000.

[MDZ99] A. Myers, P. Dinda, and H. Zhang. Performance characteristics of mirror servers on the internet. In *Proc. IEEE Infocom'99*, 1999.

[Moc87] P. Mockapetris. Domain names: Concepts and facilities. In *RFC 1034*, November 1987.

[MYNI00] H. Miura, M. Yamamoto, K. Nishimura, and H. Ikeda. Server load balancing with network support: Active anycast. In *Second International Working Conference on Active Networks (IWAN 2000)*, Toyko, Japan, October 2000.

[NZ02] T.S.E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proc. INFOCOM 2002*, New York, NY, USA, June 2002.

[OD83] D. Oppen and Y. Dalal. The clearinghouse: A decentralized agent for locating named objects in a distributed environment. In *ACM Transactions on Office Information Systems, vol. 3, pp. 230-253*, July 1983.

[PAB+98] V.S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proc. ASPLOS'1998*, San Jose, California, USA, October 1998.

[PMM93] C. Partridge, T. Mendez, and W. Milliken. RFC 1546: Host anycasting service, November 1993.

[PRR97] C.G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. 9th ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, June 1997. Newport, Rhode Island, USA.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM'01*, San Diego, CA, August 2001.

[RHKS01] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *NGC*, November 2001.

[RLS+03] A. Rao, K. Lakshminarayan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *Proceedings for the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, California, February 2003.

[SAZ+02] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM'2002*, Pittsburgh, PA, USA, August 2002.

[SBSV98] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek. Selection algorithms for replicated web servers. In *Workshop on Internet Server Performance (WISP '98)*, 1998.

[SMK+01] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM'01*, San Diego, CA, August 2001.

[SSK97] S. Seshan, M. Stemm, and R. Katz. Spand: Shared passive network performance discovery. In *Proceedings of USITS' 97*, 1997.

[SSK02] T. Skopal, V. Snasel, and M. Kratky. Properties of space filling curves and usage with ub-trees. In *Workshop on Information Technologies - Applications and Theory, Proc. ITAT 2002*, Malino Brdo, High Fatra, Slovakia, September 2002.

[TSS+97] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. In *IEEE Commun. Magazine, vol. 35, no 1, pp. 80-86*, January 1997.

[TW96] D. Tennenhouse and D. Wetherall. Towards an active network architecture. In *ACM Computer Communication Review, vol.26, pp 5-18*, April 1996.

[WC02] M. Welsh and D. Culler. Overload management as a fundamental service design primitive. In *Tenth ACM SIGOPS European Workshop*, September 2002.

[WCB01] M. Welsh, D. Culler, and E. Brewer. Seda: An architecture for well-conditioned, scalable internet services. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, October 2001.

[ZKJ01] B.Y. Zhao, J.D. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.

[ZZJ+01] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV*, June 2001.