Supporting Server Selection in Differentiated Service Networks*

Fang Hao
Bell Laboratories
Murray Hill, NJ 07974
fangh@research.bell-labs.com

Ellen W. Zegura *Mostafa H. Ammar*
Networking and Telecommunications Group
College of Computing, Georgia Tech, Atlanta, GA 30332
{ewz,ammar}@cc.gatech.edu

Abstract—**As the Internet has grown in size and diversity of applications, two trends have emerged to provide good end-user perceived performance. First, servers are often replicated for better scalability of the service. Second, QoS approaches, such as the differentiated services framework, have been proposed as enhancement to the best-effort IP service. We are interested in the combination of these two trends; that is, replicated servers in QoS-based networks. In this paper, we focus on the problem of selecting amongst replicated servers in the context of differentiated service networks. Our contributions are twofold. First, we design a QoS-based server selection architecture. The architecture is scalable in the sense that server selection and resource reservation are done in an aggregated fashion and operate in the background, rather than being driven by individual client demand. At the same time, the architecture offers fast response time to client requests for server selection. Second, we explore the design space implied by the architecture and evaluate various design options including signalling protocols, server selection/sorting algorithms and resource reservation granularity.**

I. Introduction

As the Internet has grown in size and diversity of applications, two trends have emerged to provide good end-user perceived performance. First, to achieve scalability, Internet services frequently use *replication*, in which multiple instances of a single service are distributed either locally (in a server farm) or globally (using wide-area replication). Effective use of replicated services requires solutions to problems including naming, group management, and server selection (i.e., selecting a server to meet a client request).

Second, to achieve quality of service (QoS), additions to best-effort IP service in the form of *integrated* [27], [29], [30] and *differentiated services* [5], [19], [20] are under development. Integrated services (int-serv) can potentially offer guaranteed end-to-end QoS on a per-flow basis, at the cost of complexity in protocol processing and state requirements. Differentiated services (diff-serv) achieve scalability by dealing with aggregates of flows and offering a small set of different service categories.

We are interested in the combination of these two trends; that is, replicated servers in QoS-based networks. In this paper, we focus on the problem of selecting amongst replicated servers in the context of differentiated service networks. Our goal is to provide the following:

> *Given a client request for service, select a server and server-client path that satisfies the client QoS constraint. The client request will consist of (at least) a name for the desired service and a QoS requirement. The server will be selected from amongst a set of servers that offer the desired service.*

∗ This work was supported by the DARPA ActiveCast project F30602-99-1-0514, NSF grant ANI-9973115, Sprint, and BellSouth research funding. The work was done during the first author's graduate study at Georgia Tech.

Our focus in this paper is on the architecture, mechanisms and signalling protocols to make the selection. We do not consider issues such as replicated server group membership and details of the selection API, since they are largely orthogonal to the selection system.

Best-effort server selection can leverage the "best-effort" nature of the service to provide a scalable solution that usually (but generally not always) provides good performance (e.g., [11], [13], [28], [7], [4], [10], [26]). QoS-based server selection faces additional challenges, since the service inherently makes a guarantee regarding performance. Unlike in best-effort networks, offering a server selection mechanism in QoS networks may require explicit support from the network layer since we have to consider the path QoS constraints. Hence we believe it is important to take the server selection mechanism into consideration in the design and implementation of diff-serv networks, so that server selection can be more easily and efficiently supported. Furthermore, there is always a scalability concern associated with QoS networks. Although diff-serv is designed to be more scalable than other alternatives, supporting end-to-end QoS remains a challenging issue. Any new mechanisms proposed in this context should not incur much additional overhead. For instance, it is not desirable to make server selection and resource reservation on a per-flow basis.

Our contributions are twofold. First, we design a QoS-based server selection architecture. The architecture is scalable in the sense that server selection and resource reservation are done in an aggregated fashion and operate in the background, rather than being driven by individual client demand. At the same time, the architecture offers fast response time to the client requests for server selection. Second, we explore the design space implied by the architecture and evaluate various design options including signalling protocols, server selection/sorting algorithms and resource reservation granularity. We find, for example, that "backward" signalling, from clients towards servers, is much more efficient than "forward" signalling, from servers to clients. However, backward signalling requires additional support from bandwidth brokers placed in the network, and hence adds complexity to the bandwidth broker signalling protocols.

The paper is organized as follows. We next review related work. We introduce the differentiated service network model in Section III, and present the architecture design in Section IV. We then explain the simulation configuration in Section V, and present our performance evaluation results in Section VI. Finally, we summarize our results in Section VII.

II. Related Work

Most research in server selection and *anycasting*[1] has been done in the context of best-effort networks such as the current Internet. Very little attention has been paid to server selection in QoS networks, with the exception of the ATM PNNI specification [2]. In this section, we first review the related research in best-effort networks, and then explain the anycasting mechanism defined in PNNI.

A. Server selection in best-effort networks

The simplest server selection approach is to assign servers statically, i.e., each client uses a *pre-determined* server. For instance, Domain Name Service (DNS) servers [17] and Usenet news servers [15] for clients are statically configured. DNS can also be enhanced to support mapping a service to a server inside a domain [12]. It can approximately balance the load among servers by making selections randomly or in a round robin fashion. More sophisticated dynamic server selection techniques have also been proposed [13], [28], [7], [26]. Such techniques typically select servers based on server and/or path characteristics that are dynamically measured.

Server selection can be supported at either the network layer or the application layer. RFC 1546 proposes a network layer anycast service that can be used for service location and autoconfiguration [22]. Recently, Katabi and Wroclawski present a framework for Global IP-Anycast (GIA) [16], intended to provide global and scalable support for network layer anycast. Network layer anycasting typically does not consider server characteristics such as the server load.

Fei et al. have proposed a detailed application-layer anycasting architecture, including a novel server selection mechanism that can be used for Web servers [4], [10], [31]. Their architecture considers both server and path metrics. Designated "resolvers" are used to collect both path and server information and to select an appropriate server for clients. We will use the resolver-based architecture as a starting point for our architecture.

All the above studies have been conducted in the context of best-effort networks. The major goal is to select a server (together with the path) with the best "performance" (e.g., with shortest expected response time). Our study differs fundamentally from the earlier work due to the focus on QoS networks. Our primary goal is to select a server and path that satisfy the client QoS constraint and optimize network resource usage. Furthermore, scalability is a more challenging issue in our study because of the overhead in QoS network operations.

B. PNNI anycasting

PNNI is a QoS-based routing and signalling protocol for ATM networks [2], used to set up a virtual circuit that meets the user end-to-end QoS requirements. We describe the anycast service provided in PNNI in detail here, given its relevance to our study. PNNI defines anycast service at the network/link layer, as part of its routing architecture. Each server that provides the same service participates in the same anycast group by declaring that it can *reach* the anycast group address. The *reachability information* is flooded throughout the network. When a user needs to request a service, it simply sends a request across the user-network interface (UNI) to a directly connected switch, indicating its QoS requirements and the anycast group address. The switch then selects a server based on reachability information. The rest of the operations are similar to unicast requests: the switch first selects a path to the server based on its routing database and the user QoS requirement, then sends out a connection setup request along the path. The connection is successfully set up if every network component on the path and the server can indeed satisfy the QoS requirement. Note that PNNI does not specify how to select a server or path.

PNNI anycasting has several major constraints: First, PNNI supports routing and anycasting based on flooding of network state and reachability information, which consumes significant network resources. Furthermore, the operations are supported on a per-flow (or connection) basis, and hence the overhead increases with the number of flows. Both reasons lead to scalability concerns with the protocol. Second, PNNI does not consider server QoS information. A switch can only select a server based on reachability information, without knowing whether the server can satisfy the client QoS constraints or not. Hence the selected server may not sustain the user QoS requirement, especially when server capacity is less adequate than network capacity.

Our work considers server selection in a diff-serv-capable Internet, which does not lend itself to much of the ATM-style mechanisms used in PNNI anycasting. Furthermore, we consider both path and server QoS information in our server selection architecture.

III. Differentiated Service Networks

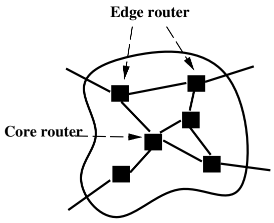


Fig. 1. Edge and core routers

This section provides a brief overview of differentiated services, and highlights the most important assumptions with respect to our work. Diff-serv networks distinguish between *edge* routers and *core* routers of a domain (Figure 1). Edge routers are connected to end hosts or routers of other domains. A core router is solely connected to routers in its own domain. Since core routers generally see many more flows than edge routers, a diff-serv network implements only simple operations in the core routers and pushes more complex operations to the edge routers. When data packets enter a diff-serv domain, they are classified, marked, shaped, and policed in the edge routers, typically on a per-user-flow basis. The packets that pass through the edge routers are marked as certain flow aggregates, each corresponding to a *per-hop behavior* (PHB). The PHB defines how the packets should be forwarded. The core routers treat the packets on a PHB basis, i.e., the packets sharing the same PHBs

[1] Anycast is a communication paradigm involving selection of one destination from a set of destinations. We use the terms anycast and server selection interchangeably.

are treated in the same way. Only a small number of PHBs are to be supported in the core routers.

QoS can be guaranteed in a diff-serv network domain if the resources are appropriately provisioned. The resources are managed based on contracts between the neighboring diff-serv domains, called *service level agreements* (SLAs). The diff-serv domains configure their edge and core routers based on the SLAs to make sure enough resources are provisioned for the diff-serv traffic. End-to-end QoS can be ensured by concatenation of the SLAs between all neighboring domains on the path.

Many details regarding the realization of a diff-serv network remain open and are the subject of debate within the IETF and elsewhere. Therefore, we make a set of assumptions about the basic operation, which are consistent with the current proposals for diff-serv operation. Specifically, we assume:

(1) The resources in each diff-serv domain are managed by an entity called a *bandwidth broker* (BB) [20]. The BB is responsible for negotiation with BBs in the neighboring domains, and resource management in its own domain. BBs in different domains typically do not share resource availability information, but they can communicate with each other to reserve resources by using signalling. One example of such a signalling protocol is the Qbone BB protocol defined by the Internet2 QBone Bandwidth Broker Advisory Council [18].

(2) SLAs define the maximum amount of resources that can be allocated by one domain on behalf of another domain. SLAs are relatively static, i.e., they change on a long time-scale (weeks or months). A particular portion of the resources within the range of the SLA must be explicitly allocated before any data can actually use the service. Such resource allocation can occur dynamically, e.g., on a time-scale of several minutes or hours.

(3) Resource allocation is typically done end-to-end between all diff-serv domains along the data path to insure end-to-end QoS for the aggregate flow between two end domains.

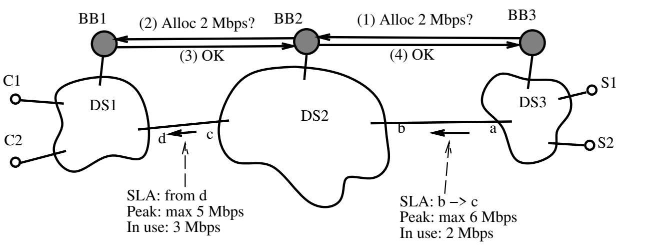


Fig. 2. Resource allocation in diff-serv networks

Figure 2 shows an example of the resource allocation process. Suppose diff-serv domains DS3 and DS2 have the following SLA: DS2 provides a virtual wire (VW) service[2] for DS3 from ingress point b to egress point c with peak rate of 6 Mbps. Also assume that 2 Mbps has already been allocated. Similarly, assume DS1 provides VW for DS2 from ingress point d to the hosts in DS1 with peak rate of 5 Mbps; assume 3 Mbps has already been allocated. Suppose that it is determined that some collection of hosts in DS3 need to send data to hosts in DS1 at a rate of 2 Mbps. The bandwidth brokers in each domain are responsible for allocating the resources, as indicated by the ordered message exchange between BBs. BB3 will send a request message along the data path to BB2, and then to BB1. If resources are available in each domain, a confirmation will be sent back, and the corresponding resources will be allocated; otherwise, a rejection message will be returned to BB3. Note the allocated resources can be adjusted by a similar signalling process if the user demand changes in the future.

IV. Architecture Design

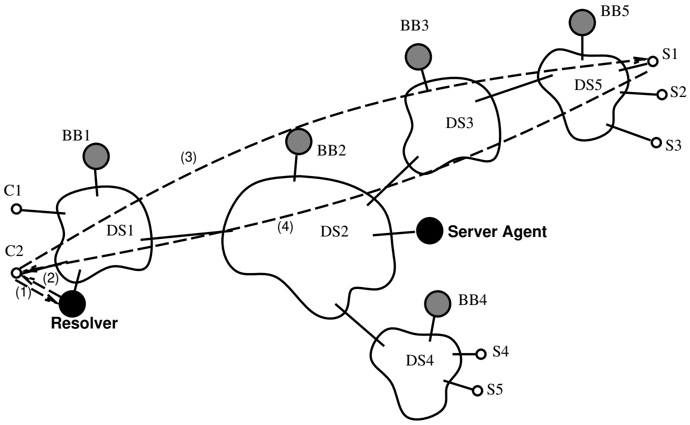


Fig. 3. Architecture: Entities and client-resolver interaction

A. Overview

Our goal is to design an architecture to support QoS-based server selection in diff-serv networks. We intend to provide end-to-end QoS guarantees to anycast clients. Hence we need to discover the available resources on the paths from the servers to the clients and make a reservation accordingly. However, the BBs in different domains typically do not share resource availability information; and no entity in the network maintains global network state information. This implies that we have to rely on signalling between BBs for resource discovery and reservation. However, it is not desirable to send reservation messages to BBs on per-flow basis, since that approach will dramatically increase the signalling overhead and thus compromise the scalability of the diff-serv network. Instead, we propose to reserve resources in an aggregated fashion. More details of the architecture are described next. Note that we focus on bandwidth requirement in this design; but it should be possible to extend the architecture to also accommodate other QoS metrics.

Our design is based on two types of entities: *resolvers* and *server agents*, illustrated in Figure 3. There is a resolver in each domain that has anycast clients, following a similar approach to the best-effort service architecture of Fei et al. [4], [10], [31]. An entire anycast group shares a single server agent; one server agent may act on behalf of many anycast groups. The server selection process is fairly simple from a client's perspective, as also illustrated in Figure 3: When a client (C2) needs to access the service, it contacts the resolver in its own domain and asks for a server address. The resolver replies to the client with a server address (S1) that can meet the client QoS requirements. Then Client C2 sends a request to server S1, and S1 starts delivering data to C2.

To enable the resolver to make appropriate server selection decision, additional background operations are necessary. In the

[2]VW [21] intends to emulate a conventional leased line service in diff-serv networks, by using the expedited forwarding (EF) PHB [14]. It guarantees to deliver data with a specified peak rate, low loss, low latency, and low jitter. Here we use VW as an example, but our anycast architecture design does not depend on any particular service.

background, the resolver predicts the client demand in the domain, and contacts the server agent and/or BBs to make server selection decisions and resource reservation. The server agent collects dynamic QoS information from all individual servers. This can be done by letting the servers "push" the resource update information to the server agent whenever there is a significant change in the server load. Based on the server resource information and the resolver demand, the server agent can select a list of candidate server domains that have servers with available capacity, and then assist the resolver in server selection and resource reservation. Although the resolver selects one server for each individual client request, it may reserve resources from multiple servers (potentially in multiple server domains) for all its clients in the anycast group, so that different clients may be assigned to different servers. Multiple server domain reservation is more flexible than single domain reservation, but is also more expensive in terms of signalling cost, as we will show in Section VI-B.

In summary, the server selection is accomplished in two steps: first, the server agent selects servers that have available capacity; and second, among the selected servers, the server agent, resolver, and BBs work jointly to select the servers with enough resources on their paths to the client[3].

After the reservation process, the resolver has a list of selected servers, including both server addresses and the amount of resources that are available from each server. The resolver monitors the utilization of the reserved resources and adjusts the reservation accordingly by sending new signalling messages.

There are four major issues in this architecture design: (1) client demand prediction; (2) signalling protocol between the resolver, server agent, and BB; (3) server selection / sorting algorithms; and (4) resource reservation granularity. We discuss the issues in further detail in the next four sections.

B. Determining client demand

The resolver needs to determine the resource demands of its clients before it contacts the server agent and/or BBs to make reservations. This can potentially be done in one of three ways: (1) the resolver initiates a new reservation whenever it receives a request from a client; (2) the resolver collects the client requests for a certain period of time, and then makes reservations for the aggregation of all the requests; or (3) the resolver tries to anticipate the aggregated demand of all its clients and make reservations correspondingly, before the clients actually request for the service.

Approach (1) can lead to accurate reservation for each individual client, but it is not scalable because a separate reservation is triggered by each client request. Approach (2) is more scalable than (1). However, the resolver has to wait for a certain period of time in order to have reasonable aggregation, which causes delay in response to the client requests. We believe that the third approach is the best choice because it is scalable and offers fast response time for client requests.

Under the third approach, either the clients have to tell the resolver their future demand in advance, or the resolver has to predict their future demand. Prediction algorithms that have been used in the context of resource provisioning, such as the local maximum or local Gaussian predictor discussed in [9], are also applicable here. The prediction algorithms are relatively orthogonal to the other three design issues in the architecture, and we focus on the other issues in the rest of this study.

C. Signalling protocol

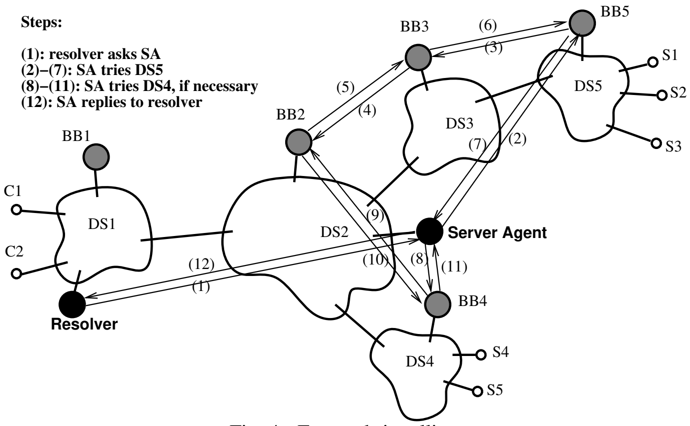


Fig. 4. Forward signalling

Based on its demand prediction algorithm, the resolver is triggered to reserve resources from the server domains to the client domain. Resource reservation is done by signalling between resolver, server agent and BBs. We consider two alternative signalling directions: *forward* (from the server domains to the client domain) and *backward* (from the client domain to the server domains).

In *forward signalling*, the resolver asks the server agent to select server domains and reserve resources. The server agent first selects a list of candidate server domains where servers have resources available, and then tries each domain in the list sequentially by contacting the BB in each server domain.

Figure 4 shows an example of forward signalling. Suppose the server agent selects both DS5 and DS4 as candidate server domains. The server agent first contacts BB5, which triggers signalling messages between neighboring BBs on the path to the client domain (i.e., BB5-BB3-BB2). If all the domains have available resources, a confirmation message will come back to the server agent, and resources will be allocated along the path. Otherwise, the server agent can contact BB4, and try to reserve resources from DS4. Similarly, signalling occurs down the data path towards the client domain (i.e., BB4-BB2). The server agent replies back to the resolver after either resources are reserved, or all candidate server domains are tried. If reservation is successful, the reply message will contain the list of selected server domains and corresponding resources.

In *backward signalling*, the resolver asks the server agent for a list of candidate server domains where servers have resources available. The resolver then asks the BB in its own domain to reserve resources from one or more of the candidate server domains.

Figure 5 shows an example of backward signalling. Suppose the resolver receives the candidate server domain list (DS4, DS5) from the server agent resolver then contacts BB1 to reserve resources from DS4 or DS5, which triggers signalling messages to the neighboring bandwidth brokers in the upstream (BB1-BB2-

[3]In general, both server capacity and network resources can dynamically change and become bottleneck. Hence both steps are necessary.

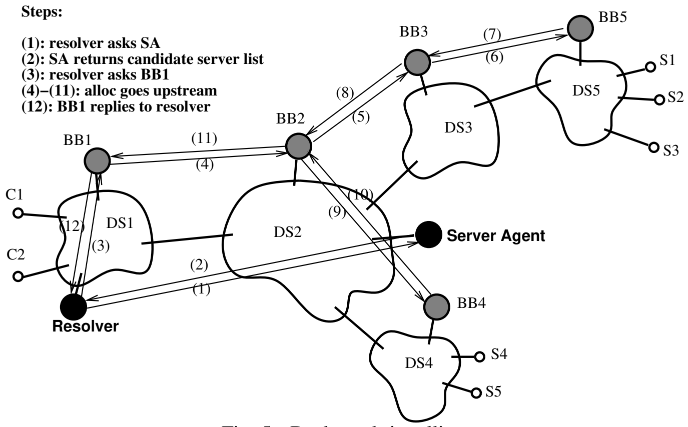


Fig. 5. Backward signalling

BB3-BB5). Note that the BBs on the path may participate in server selection. For instance, BB2 may decide which direction (to BB3 or BB4) to try first. BB2 may also decide to try another direction if the previous reservation attempt fails.

We can view the paths from all the candidate server domains to a particular client domain as a tree structure, where the client domain is the root and the server domains are leaves (Figure 6(a)). Forward signalling starts from a leaf and traverses to the root, as shown in Figure 6(b). Multiple leaves may need to be tried before a successful reservation can be made, since some of the domains on the path may not have enough resources available. Note that the server agent makes the decision on which server domain to try, and the BBs determine if resources are available. The BBs do not have to distinguish between anycast and unicast signalling; hence no significant change in the current BB signalling proposals (e.g., [18]) is required.

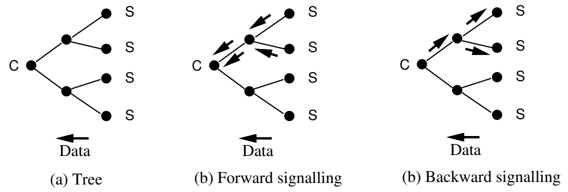


Fig. 6. Comparison of forward and backward signalling

On the other hand, backward signalling starts from the client domain (root), and traverses to the server domains (leaves), as shown in Figure 6(c). Unlike forward signalling, each BB on the path may decide which directions (or branches) to try. Backward signalling is more efficient than forward signalling for two reasons. First, the signalling messages are combined towards the root. At most one message needs to be sent on each link. Hence the total number of signalling messages is reduced. Second, backward signalling can prune infeasible links (i.e., the links with insufficient resources) faster than forward signalling. In forward signalling, pruning one infeasible link leads to pruning one path and leaf. For instance, in Figure 4, if we find link DS5-DS3 is not feasible after step (2), we do not need to do steps (3)-(6), i.e., we can prune path DS5-DS3-DS2-DS1 together with leaf DS5. But in backward signalling, pruning one infeasible link causes the whole subtree associated with the link to be pruned. For instance, in Figure 5, if we find link DS2-DS1 is not feasible after step (3), all steps (4)-(11) can be avoided, i.e., both branches DS2-DS3-DS5 and DS2-DS4 can be pruned. Faster pruning can narrow down the searching space more quickly, and hence reduce the time to reach a successful reservation.

However, backward signalling requires additional support from the BBs. The BBs need to signal upstream instead of downstream. Since the path from a sender to a receiver is, in general, different from that from the receiver to the sender (i.e., the paths are asymmetric), we also need to discover the upstream paths so that the BBs know where to send signalling messages[4]. Hence backward signalling is somewhat more complex.

D. Server pre-selection and sorting algorithms

Server selection is accomplished in two steps. First, the server agent selects a list of candidate server domains based on the server information. Second, signalling is initiated either in the forward or the backward direction to reserve resources from one or more server domains to the client domain.

We refer to the first step as *pre-selection*. Pre-selection can be based on metrics such as available server capacity and the domain-level distance (i.e., number of domains on the path). Server capacity information is "pushed" from each individual server to the server agent. A server domain is *feasible* if it has at least one server with available capacity to support the resolver request. We study two pre-selection algorithms: (1) *unrestricted selection*, which selects all feasible server domains; and (2) *closest-only selection*, which selects all feasible server domains that are closest (i.e., with minimum domain-level hop-count) to the client domain.

In the second step, we need to decide the order in which the candidate server domains should be tried. In forward signalling, the decision is made by the server agent. In backward signalling, the BBs on the path need to decide which direction (i.e., ingress point) to try first. Note that several server domains may correspond to the same ingress point. We study the following three sorting algorithms to determine order:

- *Random*: sort the server domains or ingress points in random order.
- *Widest-first*: sort the ingress points according to the amount of resources available from the ingress point to the egress point; ingress points with more resources go first.
- *Probabilistic balancing*: sort the ingress points based on the resources available from the ingress point to the egress point. However, the order is randomized such that the ingress points with more resources are more likely to be attempted earlier.

Random sorting is applicable for both forward and backward signalling; the other two can only be used in backward signalling.

E. Reservation granularity

In the simple case, resources are reserved from a single selected server domain to the client domain. However, for large demands, it may be better to allow the reservation to be divided between several server domains, so that the anycast clients in a

[4]Here is one possible way to discover the upstream paths. The BBs access the BGP routing information in their own domains. The server agent contacts the BBs in the server domains to obtain the path information from the server domains to the client domain, and then relay the path information to the resolver in the client domain. The resolver may then use the discovered upstream path information during the backward signalling process.

domain can get service from several server domains. We define *reservation granularity* as the minimum amount of bandwidth that can be reserved from each server domain. Intuitively, a decrease in the reservation granularity makes resource reservation more flexible since the server agent or BBs have more freedom in deciding how to split the reservation among different server domains. Consequently, it may lead to better reservation results. On the downside, splitting the reservation across multiple server domains may potentially cause extra signalling overhead. We explore this trade-off in the performance evaluation.

V. Simulation Configuration

We simulate our proposed QoS-based server selection architecture using a simulator written in TeD [3], [23], [24], [25], a network description language that supports discrete event simulation with parallel executions. The simulation configuration is described below.

A. Network topology

The simulation results presented in this paper are based on a 100-node topology[5] that models domain-level connectivity. There are 25 transit domains (ISP domains) and 75 stub domains (end-user domains). We use GT-ITM [6] to generate the 25 node transit domain topology, with the Waxman-1 algorithm for transit-transit connectivity; we then attach 75 stub domains and randomly connect each stub domain to one or two transit domains. We also randomly generate 30 edges between stub domains. The average degree of the graph is 3.4; the degree is much higher for nodes in transit domains and lower for stub domains.

We randomly select five stub domains as anycast server domains, and 50 stub domains as anycast client domains. Static SLAs exist between neighboring domains. Each domain agrees to provision a certain amount of bandwidth for its neighbors to all other neighbors (i.e., between each pair of ingress and egress points). This bandwidth is 25 Mbps for domains without anycast servers, and 75 Mbps for those with anycast servers.

We assume the servers have much higher capacity than the links can support in the experiments, and focus on the impact of network resource constraints on the reservation result.

B. Traffic model

We impose two types of traffic on the topology: unicast and anycast. Unicast traffic exists between each pair of end-user domains. The BBs in the end-domains predict their total user demand and make resource reservations based on the prediction. For simplicity, we only model the resource demand, intead of modeling the actual data traffic and the demand prediction algorithm. Each end-user domain generates the total rate for all out-going data flows, and splits the flow to all other end-user domains according to a uniform distribution. The total rate is uniformly distributed. A new rate and traffic distribution across destination domains are generated after an exponentially distributed holding time.

[5]We have also experimented with other topologies such as the ISP topology used in [1], with similar results.

Anycast demand exists between anycast server domains and client domains. The requests are generated by resolvers for aggregated anycast clients. Similar to the unicast demand, the requested rate and holding time are uniformly and exponentially distributed, respectively.

VI. Performance Evaluation

We evaluate both the signalling overhead and reservation performance of the design options implied by the server selection architecture. Signalling overhead is measured by the average number of hops that each signalling message traverses. Note that the number of signalling messages remains the same across all our experiments since the rate holding times are selected from the same distribution. Reservation performance is measured by the demand acceptance ratio, i.e., the total rate of the accommodated reservation requests over the total rate of all reservation requests.

A. Signalling direction

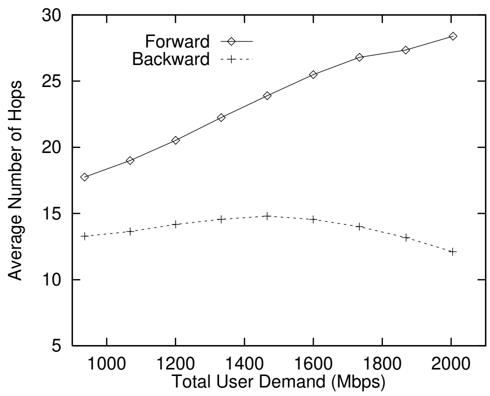
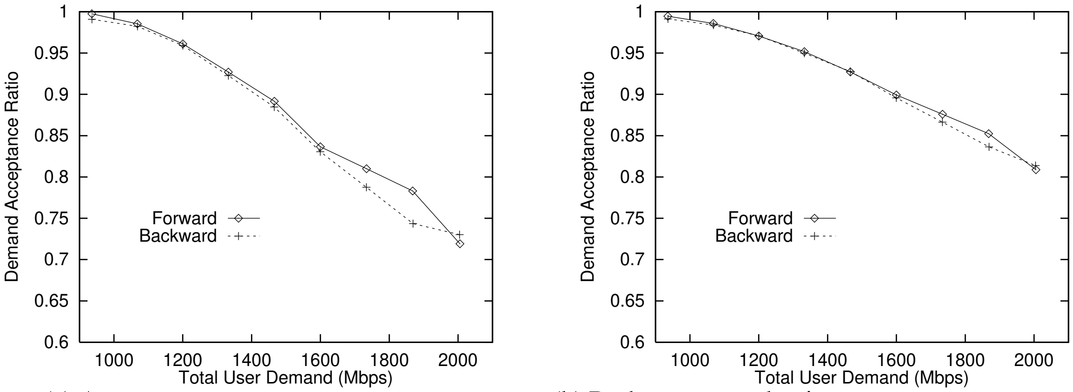


Fig. 7. Forward vs. backward signalling: overhead

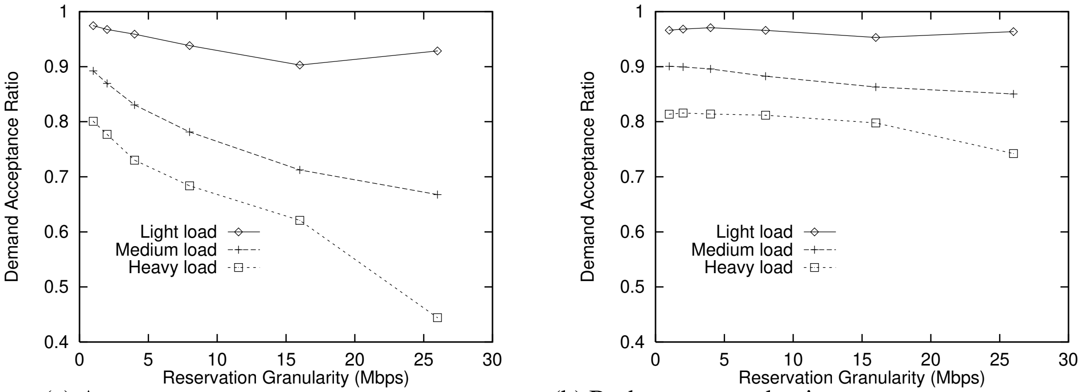
Figure 7 shows signalling overhead under both forward and backward signalling. Unrestricted server pre-selection and random sorting are used in both cases. The reservation granularity is 4 Mbps. We experiment with different user demand levels by changing the user bandwidth demand size while keeping the ratio of unicast and anycast demands the same (about 1.5). Total user demand (horizontal axis) refers to the total user demand (including both anycast and unicast demand) averaged over time. We observe that backward signalling has much less overhead, especially as total demand increases. The main reason is that backward signalling tends to combine the signalling messages towards the client domain (the root of the tree) and hence reduces the total number of signalling messages, as we explained in Section IV-C.

We also find that the difference between forward and backward signalling becomes larger as the total user demand increases. With the increase in user demand, forward signalling overhead monotonically increases, but backward signalling overhead first slightly increases and then decreases. This is because two factors contribute to the change in signalling overhead. On one hand, fewer network resources are available when the user demand increases, and hence the server agent or BBs need to try more times before they can find a feasible path, causing larger overhead. On the other hand, when fewer resources are available, we



(a) Anycast requests (b) Both anycast and unicast requests

Fig. 8. Forward vs. backward signalling: reservation performance



(a) Anycast requests (b) Both anycast and unicast requests

Fig. 9. Reservation granularity: reservation performance

can prune the infeasible paths faster since fewer links are likely to have sufficient resources, which reduces signalling overhead. Whether the signalling overhead actually increases or decreases depends on which factor is more significant. As explained in Section IV-C, the pruning effect is much more significant for backward signalling, especially when the user demand is relatively higher and fewer resources are available.

Figures 8(a) and (b) show the reservation performance for anycast requests and all requests (both anycast and unicast), respectively. In both cases, we find that forward and backward signalling have similar performance. Almost the same amount of user requests can be accommodated regardless of the signalling direction. This is because the same pre-selection and sorting algorithms are used; it does not matter whether we try from the server side or from the client side. Note that backward signalling is more flexible in the sense that more sorting algorithms can be used. We will show the effect of different sorting algorithms in Section VI-D.

The demand acceptance ratio for all requests (Figure 8(b)) is a little higher than for anycast requests (Figure 8(a)). This is because of the simulation configuration: since there are only five anycast servers, bottlenecks are formed near the server domains, which causes more anycast requests to be rejected.

B. Reservation granularity

In this section, we evaluate the impact of reservation granularity on reservation performance and signalling overhead. Backward signalling, unrestricted pre-selection and random sorting are used. We experiment with three different levels of traffic load: light, medium, and heavy [6]. Figure 9(a) shows the reservation performance under different reservation granularities for the anycast requests. We observe that the demand acceptance ratio decreases with the increase in reservation granularity; and the effect is more dramatic when the traffic load is relatively higher. This is because it is generally easier to satisfy an anycast request when the granularity is small, since resources can be reserved from multiple server domains (and paths) when a single domain (and path) does not have enough resources. Granularity affects performance more significantly when traffic load is higher and hence fewer resources are available, because there is a larger chance that multi-domain reservations can help.

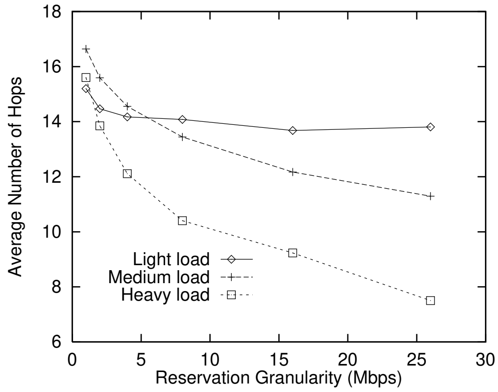
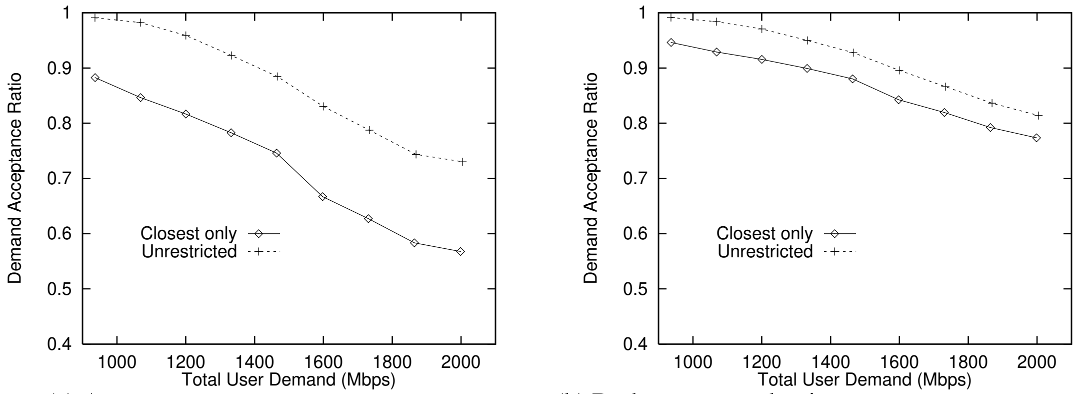


Fig. 10. Reservation granularity: overhead

[6]The total user demand under light, medium, and heavy load is 1200 Mbps, 1500 Mbps, and 2000 Mbps, respectively.



(a) Anycast requests (b) Both anycast and unicast requests

Fig. 11. Pre-selection algorithms: reservation performance

However, the total demand acceptance ratio for both anycast and unicast requests does not change significantly with the change in reservation granularity, as shown in Figure 9(b). In other words, about the same amount of user requests (counting both anycast and unicast) can be accommodated regardless of the reservation granularity; however, the reservation mechanism favors anycast requests when the granularity is small, and tends to grab more resources away from unicast applications.

Figure 10 shows the signalling overhead under different reservation granularities and different traffic load. We observe that the signalling overhead decreases with the increase in reservation granularity; the overhead decreases faster when the traffic load is higher. This result indicates that the better reservation results under smaller granularity that we observed in Figure 9(a) do not come for "free": the cost is higher signalling overhead.

We also observe that when granularity is smaller, the signalling overhead first increases and then decreases with the increase in the traffic load, so that the overhead under medium load is larger than both light and heavy load. This is consistent with our observation in Section VI-A — recall that Figure 7 corresponds to relatively smaller reservation granularity (4 Mbps). However, when the granularity is larger, the overhead monotonically decreases with the increase in the traffic load. This is because it is easier to prune a link under larger reservation granularity, so that the pruning effect is the dominant factor that decides signalling overhead.

C. Pre-selection algorithms

We compare two pre-selection algorithms in this section: unrestricted selection and closest-only selection. Unrestricted selection chooses all servers that are able to support the anycast resource requirement, regardless of the distance between the server and client domains. Closest-only selection imposes one additional restriction: only the servers with the minimum distance (in terms of the number of domains) are chosen. Unrestricted selection is in some sense more "greedy" since it tries every possibility to make a reservation. Closest-only selection tends to make efficient usage of network resources since it only tries to use the server domains with shortest distance to the client domain even though other server domains with longer distance may be available.

We use backward signalling and random sorting in this set of experiments. The reservation granularity is 4 Mbps. Figures 11(a)(b) show the reservation performance for anycast user requests and all user requests, respectively. Figure 12 shows the signalling overhead for anycast requests. We make the following observations across the experiments: (1) Unrestricted selection can accommodate more anycast requests than closest-only selection, because it tries to maximize the chance of making reservations for anycast requests. (2) Unrestricted selection also has better reservation performance when both anycast and unicast requests are considered, which suggests that unrestricted selection makes use of the resources that would otherwise be wasted. However, the difference between unrestricted and closest-only selections is smaller for all user requests than for only anycast requests. This implies that closest-only selection tends to accommodate more unicast requests. (3) Unrestricted selection has larger signalling overhead than closest-only selection because it is more aggressive in making reservations. This is the price that it pays for better reservation performance.

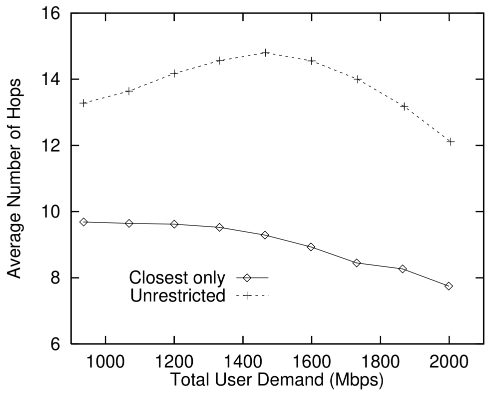


Fig. 12. Pre-selection algorithms: overhead

D. Sorting algorithms

The sorting algorithm determines which server domain is tried first during the signalling process. As we mentioned in Section IV, there are fewer alternatives in forward signalling since no path information is available. In this section, we evaluate three sorting algorithms for backward signalling: random, widest-first, and probabilistic balancing.

We use unrestricted selection in this set of experiments. The reservation granularity is 4 Mbps. We observe that the sorting algorithms make very little difference in both reservation performance and signalling overhead. Figure 13 shows the reservation

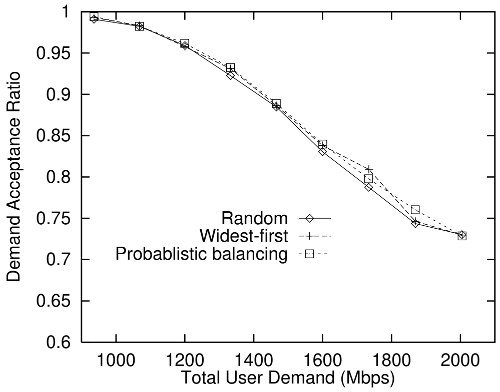


Fig. 13. Sorting algorithms: anycast reservation performance

performance for anycast requests. Results of signalling overhead are not shown due to space constraints, but they indicate that the overhead of three sorting algorithms is very close. This result suggests that it is not immediately clear whether one can improve reservation performance by taking the advantage of the local resource availability information in each domain, since a local optimal selection does not necessarily lead to a global optimal selection. However, we do not have enough evidence to conclude how the three sorting algorithms perform in general cases. For instance, widest-first or probabilistic balancing might perform better than random sorting when the distance between all server and client domains are very short, if we could assume the resource availability in two neighboring domains are positively correlated.

VII. Conclusions

In this paper, we have presented a QoS-based anycast architecture to support server selection in differentiated service networks. Resources for the clients in a domain that request the same anycast service are aggregated and reserved together. Resources are discovered and reserved by using signalling between the bandwidth brokers of the domains on the path. We have evaluated several design issues implied by the architecture, including signalling direction, server pre-selection and sorting algorithms, and resource reservation granularity. We find that: (1) Backward signalling is much more efficient than forward signalling. The advantage of backward signalling is more significant under higher traffic load. However, backward signalling requires additional support from the bandwidth brokers, and hence adds complexity to the bandwidth broker signalling protocols. (2) Smaller reservation granularity allows resources to be reserved from multiple servers, and hence leads to more flexible reservation. But smaller granularity also increases the signalling overhead. Hence it is possible to trade-off between anycast request acceptance ratio and bandwidth broker signalling overhead by adjusting the reservation granularity. (3) Unrestricted pre-selection can admit more anycast requests than closest-only pre-selection, but it also incurs more signalling overhead than closest-only. (4) The results from our experiments show that the difference between random, widest-first, and probabilistic balancing sorting algorithms is small. However, in general, it is not yet clear whether one can improve reservation performance by taking the advantage of the local resource availability information in each domain.

There are still many open issues for future research. First, we need to have a better understanding of our proposed pre-selection and server sorting algorithms and also explore new algorithms. The pre-selection and sorting algorithms are in some sense similar to conventional QoS routing algorithms [8], if we assume all the servers are connected by virtual links that have infinite resources and zero cost. One major difference is that the pre-selection and sorting algorithms are not based on global information, while most conventional QoS routing algorithms are based on routing tables that essentially reflect the global network state. It will be interesting to see whether some QoS routing algorithms can be adapted to the server selection problem.

Second, although our architecture can accommodate both server and path QoS characteristics in server selection, we have been focusing on the impact of path resource constraints in the performance evaluation. Further experiments are needed to understand the interaction between path and server resource constraints, and their impact on reservation performance.

Third, our architecture is based on signalling between bandwidth brokers. In doing so, we can make explicit reservations for anycast applications and make stringent end-to-end QoS guarantees. But the signalling overhead is still a concern for the scalability of the approach. Another possibility is to take a measurement-based approach without using signalling. For instance, it might be possible to measure the traffic patterns of each domain and predict the domain resource availability. Server selections can be done based on such predictions. However, the downside of this approach is that we will not be able to make hard QoS guarantees since we do not reserve resources.

References

[1] G. Apostolopoulos, R. Guerin, S. Kamat, and S. Tripathi. Quality of service based routing: a performance perspective. In *Proceedings of ACM SIGCOMM*, September 1998.

[2] ATM Forum. Private Network-Network Interface specification version 1.0. Technical Report af-pnni-0055.000, ATM Forum, March 1996.

[3] S. Bhatt, R. Fujimoto, A. Ogielski, and K. Perumalla. Parallel simulation techniques for large-scale networks. *IEEE Communications*, 36(8), August 1998.

[4] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei. Application-layer anycasting. In *Proceedings of IEEE INFOCOM*, 1997.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *RFC 2475*, December 1998.

[6] K. Calvert, M. Doar, and E. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, June 1997.

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

[8] S. Chen and K. Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Network, Special Issue on Transmission and Distribution of Digital Video*, November/December 1998.

[9] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe. A flexible model for resource management in virtual private networks. In *Proceedings of ACM SIGCOMM*, September 1999.

[10] Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proceedings of IEEE INFOCOM*, 1998.

[11] A. Gaulbrandsen and P. Vixie. A DNS RR for specifying the location of services (DNS SRV). *RFC 2052*, October 1996.

[12] A. Gaulbrandsen and P. Vixie. A DNS RR for specifying the location of services (DNS SRV). *RFC 2782*, Feburary 2000.

[13] J. Guyton and M. Schwartz. Locating nearby copies of replicated Internet servers. In *Proceedings of ACM SIGCOMM*, 1995.

[14] V. Jacobson, K. Nichols, and K. Poduri. An expedited forwarding PHB. *RFC 2598*, June 1999.

[15] B. Kantor and P. Lapsley. Network news transfer protocol - a proposed standard for the stream-based transmission of news. *RFC 977*, February 1986.

[16] D. Katabi and J. Wroclawski. A framework for Global IP-Anycast (GIA). In *Proceedings of ACM SIGCOMM*, 2000.

[17] P. Mockapetris. Domain names - concepts and facilities. *RFC 1034*, November 1987.

[18] R. Neilson, J. Wheeler, F. Reichmeyer, and S. Hares. A discussion of bandwidth broker requirement for Internet2 Qbone deployment (version 0.7). Technical report, Internet2 Qbone Bandwidth Broker Advisory Council, August 1999.

[19] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field) in IPv4 and IPv6 headers. *RFC 2474*, December 1998.

[20] K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the Internet. *RFC 2638*, July 1999.

[21] K. Nichols and K. Poduri. The 'virtual wire' behavior aggregate. Technical report, March 2000. IETF Internet draft `<draft-ietf-diffserv-ba-vw-00.txt>`.

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

[23] K. Perumalla and R. Fujimoto. A C^{++} instance of TeD. Technical Report GIT-CC-96-33, College of Computing, Georgia Institute of Technology, 1996.

[24] K. Perumalla, A. Ogielski, and R. Fujimoto. MetaTeD: A meta language for modeling telecommunication networks. Technical Report GIT-CC-96-32, College of Computing, Georgia Institute of Technology, 1996.

[25] K. Perumalla, A. Ogielski, and R. Fujimoto. TeD - a language for modeling telecommunication networks. *ACM Performance Evaluation Review*, 25(4), March 1998.

[26] J. Rosenberg and H. Schulzrinne. Internet telephony gateway location. In *Proceedings of IEEE INFOCOM*, 1998.

[27] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. *RFC 2212*, September 1997.

[28] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol. *RFC 2165*, June 1997.

[29] J. Wroclawski. Specification of the controlled-load network element service. *RFC 2211*, September 1997.

[30] J. Wroclawski. The use of RSVP with IETF integrated services. *RFC 2210*, September 1997.

[31] E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee. Application-layer anycasting: A server selection architecture and use in a replicated web service. *IEEE/ACM Transactions on Networking*, August 2000.