A Probe-Based Server Selection Protocol for Differentiated Service Networks

Meng Guo, Mostafa H. Ammar, Ellen W. Zegura
Networking and Telecommunications Group
College of Computing, Georgia Tech, Atlanta, GA 30332

Fang Hao
Bell Laboratories
Holmdel, NJ 07733

Abstract—**Quality-of-service (QoS) techniques and server replication are two complementary approaches that can improve the performance observed by users. QoS techniques provide differentiated service to meet the diverse needs of applications; server replication enables load balancing across a set of servers. To combine the two approaches, we need to address one important question: how to select a server among a replicated set that satisfies the user QoS requirement. In this paper, we propose a probing-based protocol to discover network resources and make reservations for requests in differentiated service networks. This approach optimizes network resource utilization with reasonable signaling overhead. Furthermore, we compare five heuristics that can be used for server selection. We also investigate two methods that can reduce probing overhead, namely caching and the technique of probing from a subset of servers.**

I. Introduction

Deploying a commercial-quality web or streaming service on the Internet is challenging for several reasons. First, scalability is a major concern due to the large network size and potential user population. Second, customers have higher expectations for the quality of paid services than for free services. For instance, on-line video on demand users may expect VHS or DVD video quality. They may also require the service to be reliable and have high availability.

Quality of service (QoS) techniques and server replication are two complementary approaches that can be used to improve user-perceived QoS. Networks deploying QoS mechanisms, such as those proposed in the differentiated service (diff-serv) context [4], can generally provide QoS assurance to user applications. Server replication reduces bandwidth consumption by moving instances of the service closer to users. Server replication also improves availability since the service is available as long as at least one server is available.

When servers are replicated, an important problem is how to select a server from the replicated set such that both the server and the path can satisfy the user QoS requirement. Currently, most research in server selection has focused on best-effort networks such as the Internet[10], [11], [19], [6]. Server selection under this scenario tries to find the server which provides the best performance, eg delay, throughput. PNNI is a QoS-based routing and signalling protocol in ATM networks [1], [18] which supports server selection with QoS guarantee. There are two major constraints for PNNI anycasting: First, the server selection operations are supported on a per-flow basis, which lead to scalability concerns with the protocol. Second, PNNI does not consider server QoS information, hence the selected server may not sustain the user QoS requirement, especially when server capacity is less adequate than network capacity.

In prior work, we have proposed a QoS-based server selection architecture to operate within the diff-serv framework [12]. In this architecture, clients in a domain that request the same anycast service are aggregated together for the purpose of resource reservation. Resources are discovered and reserved by using signaling between the bandwidth brokers (BBs) of the domains on the paths from servers to the clients. The architecture is scalable in the sense that server selection and resource reservation are done in an aggregated fashion and operate in the background. However, this mechanism has limited ability to optimize overall network resource utilization, and hence affects server selection performance. The signaling process combines resource discovery and reservation in one step, so that the server selection decisions are made based on partial path information rather than a more complete view of the availability of network resources.

In this paper, we present a probe-based server selection protocol as an alternative to the approach proposed in [12]. We separate the resource discovery and reservation into two steps. First, probe messages are sent from the server domains to the client domain to discover available resources on the paths. Second, a server (and path) is selected based on the probing result, and then resources are reserved along the path. We show that the new protocol significantly improves network resource utilization with reasonable increase in signaling overhead. We also investigate two methods that may reduce probing overhead: (1) caching probing results at the client side for later use; and (2) probing a subset of servers instead of all servers. We show that caching can significantly reduce probing overhead with little adverse impact on selection performance. We also demonstrate the trade-off between performance and probing overhead under different sizes of probed server sets.

This paper is organized as follows. We provide background in Section II. We then describe our protocol in Section III. Simulation configuration and result analysis are presented in Sections IV. Finally, we conclude our work in Section V.

II. Background

This section provides background regarding the diff-serv network model and the server selection architecture that our protocol is based upon.

∗ This work is supported by NSF grant ANI-9973115, and the Georgia Tech Broadband Institute

Differentiated service networks. Diff-serv networks are intended to provide scalable QoS assurance to user applications. Diff-serv achieves scalability by pushing the processing complexity to the edge routers so that core routers only deal with aggregates of flows. The network 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.

Details regarding realization of a diff-serv network remain open and are the subject of debate within IETF and elsewhere. We make the following assumptions about the basic operation. (1) The resources in each diff-serv domain are managed by a bandwidth broker (BB) [17]. The BB is responsible for negotiation with BBs in the neighboring domains, and resource management in its own domain. They can signal each other to reserve resources [15]. (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. (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.

Server selection architecture. We have proposed a server selection architecture to operate in the diff-serv framework [12]. The architecture consists of two major types of entities: resolvers and server agents. Each resolver is designated to serve a client domain. One server agent serves an entire group of replicated servers. The server agent collects server load information periodically from all servers in the group. Each resolver predicts the client demands in its domain, and then sends the aggregated resource request to the appropriate server agent. Upon receiving the request, the server agent first selects a list of candidate server domains and then initiates signaling to reserve resources between the server domains and the client domain. A client contacts its resolver when it needs to access the service. The resolver will then return a server address, from which the client can get the service with QoS assurance. The resolver periodically adjusts its predictions to approximate the changing demand of clients, and then renegotiates with the server agent and BBs to change its reservation.

The architecture reported in [12] combines the resource discovery and reservation processes together, so that each BB reserves resources without first knowing whether the other domains on the path also have available resource. Resources have to be released if the reservation fails at other domains on the path. Furthermore, since no resource information is available before reservation, the server agent does not have a "global picture" of the paths from all candidate servers, and hence it is very difficult to optimize the server selection decision. In this paper, we take the existing architecture and signalling protocol as a baseline, and improve the performance by adopting a new server selection protocol.

III. Probe-based Protocol

Our alternative protocol decouples resource discovery and reservation. In this section we describe the protocol operation, explain several different methods for selecting a server based on the resource discovery results, and outline a caching mechanism that reduces the overhead for resource discovery.

Resource discovery and allocation. In our probe-based protocol, resource reservation is done in two phases: discovery and allocation. In the discovery phase, probe messages are sent from the server domains to the client domain to obtain resource availability information on the paths. Probing results are sent to the resolver in the client domain. In the allocation phase, the resolver first selects the "best" server that satisfies the resource requirement, and then sends an allocation request upstream towards the selected server domain to reserve resources along the path. The selection algorithm attempts to optimize network resource utilization and maximize the amount of anycast user demand that can be accommodated.

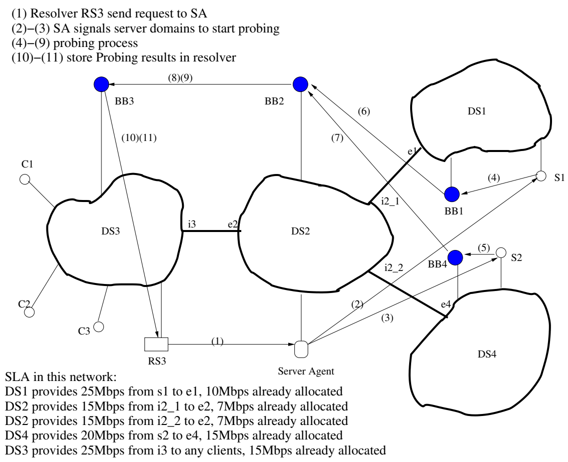


Fig. 1. Server side probing

Figure 1 shows the entities in the server selection architecture and illustrates the sequence of steps in the probing process. The resolver (RS3) predicts user demand in its domain, and initiates probing and reservation. The server agent (SA) collects server load information and selects an initial list of candidate servers that have available resources. BBs relay the probing messages and provide the resource availability information in their domains; they also respond to resource reservation requests and allocate bandwidth in their domains.

We describe the protocol by way of an example in which we assume a virtual wire service [17] is set up between neighboring domains. The total bandwidth of SLAs and the bandwidth that is already in use are stated in the figure at the lower left. Assume the resolver RS3 in client domain DS3 needs to request 1 Mbps from any of the servers s_1 and s_2. RS3 sends a request to the server agent. The server agent first selects a candidate list of server domains in which some servers have available capacity, say, DS1 and DS4; and then it signals BB1 and BB4 to start the probing process. Signalling messages are relayed through BB1-BB2-BB3-RS3, and also BB4-BB2-BB3-RS3. Resolver RS3 then knows the bottleneck bandwidth on path BB1-BB2-BB3 is min(25-10, 15-7, 25-15) = 8Mbps; and the bottleneck bandwidth on path BB4-BB2-BB3 is min(20-15, 15-7, 25-15) = 5Mbps. RS3 also knows the domain-level distance (number of intermediate domains) for both paths is 2. Note that during the probing process, if some intermediate domain does not have enough bandwidth available to support a resolver's request, a failure message will be directly sent to the resolver.

After RS3 selects a server, say S1, it then sends a reservation signal through BB3, BB2 and BB1, so that resources can be reserved on the path BB1-BB2-BB3. Unlike [12], the reservation should nearly always succeed since probing has already confirmed that the bandwidth is available[1].

Server and path selection methods. Probing generally returns multiple servers and paths that satisfy the resolver QoS requirement. We consider several methods to select from the set of feasible servers and paths:

i Random: randomly select a candidate server. This method is used as a baseline for comparison with other methods.

ii Least loaded server first: select the server with maximum available capacity. This approach attempts to balance the load among different servers [14].

iii Tightest path first: select the server with the smallest bottleneck bandwidth on its path to the client. This approach intends to pack small requests together so that paths with larger bandwidth can be used to accommodate larger future requests.

iv Widest path first: the server with the largest bottleneck bandwidth on its path to the client is selected. This approach intends to balance the traffic load in the network.

v Shortest path first: Choose the server whose path to the client has the smallest domain-level distance. If multiple such servers exist, choose the one with the maximum bottleneck bandwidth on its path.

Caching the probing results. To reduce the overhead for probing, we cache the probing results in a bandwidth availability table at the resolver. The table is indexed by the server address and contains the path and bottleneck bandwidth from the last probe. When the resolver is ready to make a new request, it first examines its bandwidth availability table. If possible, a path selection decision will be made based on the cached values. Path caching reduces the number of probing operations by reusing already calculated paths.

The cached information will be updated whenever the resolver allocates resource on a server-client path or releases previously allocated bandwidth. However, the information will become inaccurate over time as reservations are made and released to support other traffic. Several cases will make cached bandwidth information incorrect: First, whenever a successful allocation is made, only the available resource value over this path will be modified. Not modifying values of paths that share common links will cause bandwidth over-estimation. Second, when the resource along the server-client path is released, the corresponding resource availability information will be updated. Not updating the values of paths that share common links will cause bandwidth under-estimate. Other reasons such as adding/removing the links, server joining/leaving the service group, and background unicast traffic will also cause the cached values to be incorrect.

Generally speaking, there are three methods to update the cached information: *On-demand cache update:* The update of the cache table is triggered by cache error signals (e.g., a failed reservation attempt) and will result in a number of on-demand probes. *Periodic update:* All cached paths for all destinations are updated periodically. *Individual invalidation:* Each cached path has a life time associated with it. After the path's lifetime has expired, a probe along this path is initiated. We focus on *on-demand cache update* policy. This policy requires no state information, and hence is easy to implement. Two types of reservation failure may occur. First, a reservation may fail because the resolver over-estimates the resources, i.e., less resources are actually available in some network domains. Hence the resource allocation signal is rejected by the domain that lacks resource and a failure message is sent back. Second, the cached values on the resolver may indicate that no sufficient resources are available, which may be caused by under-estimation. New probing messages are triggered in both situations.

IV. Performance Evaluation

A. Simulation Configuration

We use the GT-ITM[5] transit-stub model to generate the network topology. This network is composed of 410 domains, including 10 transit domains and 400 stub domains. The average degree of this graph is 2.5, and transit domains have a much higher degree than stub domains. We randomly select 25 stub domains as anycast server domains, and 250 domains as anycast client domains. We assign static SLAs between neighboring domains. The SLAs are defined as follows: each stub domain without an anycast server provides 100Mbps for its neighbor to all other neighbors. For those domains with anycast servers, the bandwidth is 300Mbps. During this experiment, we assume an infinite server capacity.

Two types of traffic exist in the network: point-to-point background traffic, where both source and destination address are known, and server selection traffic where the client specifies an address that corresponds to a group of replicated servers. Background traffic exists between each pair of stub domains. The BBs in these domains predict their total user demand and make resource reservations based on the prediction. Server selection traffic exists between anycast server domains and client domains. The resolver in each client domain predicts the total

[1] The reservation will only fail if the bandwidth becomes unavailable in the short time between the probe and the reservation. We expect this to be a rare case.

user demand and cooperates with BBs to make resource reservations. We model only one anycast group, however would generally expect multiple simultaneous groups to exist, for the purposes of evaluating performance, it suffices to focus on one group and abstract all other traffic as background unicast connections. The duration of each client request is exponentially distributed with a mean of 1000s.

We use two metrics in the simulations: reservation performance and probing overhead. Reservation performance is measured by client demand acceptance ratio, i.e. the total rate of accepted reservation requests over the total rate of all reservation requests. Probing overhead is measured by the average number of probing signals that each server sends.

B. Analysis of Results

We simulate our server selection architecture, and compare the performance of different alternatives. Our experimentation is composed of three parts. First, we evaluate the performance of the five server selection mechanisms as described in section III. We also explore the effect of caching in reducing probing overhead and improving scalability. Finally, we evaluate the effect of probing from a subset of servers as opposed to the entire set of servers on the overall performance.

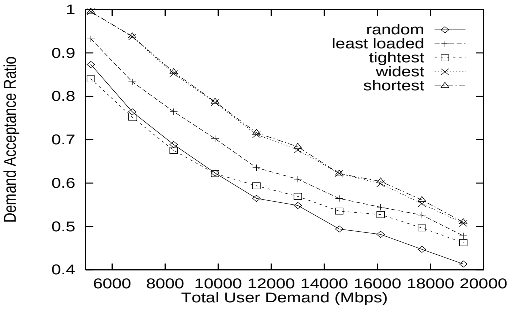


Fig. 2. Demand Acceptance Ratio

Demand Acceptance Ratio Comparison. Figure 2 shows the demand acceptance ratio as a function of total user demand under the five selection mechanisms. The total user demand is all the client demand averaged over time. We find that the shortest path selection has best acceptance ratio. The performance of different algorithms becomes slightly closer when the traffic load is higher.

Several reasons contribute to this trend.(1) The Shortest-Path first policy select the feasible server with a shortest domain distance to the clients. Thus, this mechanism tends to consume the fewest resources. In this way, there will be more available resources for future selection (2)The performance difference among different schemes becomes smaller when the traffic load becomes heavier, because there are fewer feasible paths, and different server selection policies have a higher probability to choose the same server.

Caching. Figure 3 shows the demand acceptance ratio under different user demand levels for both with and without caching. We find that the acceptance ratios are very close to each other. This means that even though the cached value is inaccurate sometimes, the performance does not drop much. The main reason is that we use a very strict on-demand caching update policy, i.e., whenever there is a chance that the resolver may claim a reservation failure, we will start a new server side probe, and make server selection decisions based on this most updated information. The caching mechanism causes a small performance loss, since occasionally, the *best* server will not be picked based on the cached value. Figure 4 shows the probing overhead. The probing overhead is significantly smaller when using the caching architecture. Since each server side probing pass will typically generate bandwidth availability information for multiple paths, this information can be reused for future client requests, thus reducing the number of server side probes. We also find that the probing overhead increases when the traffic load becomes heavier. This occurs because when the traffic load gets heavier, there will be more cache failures which cause more server side probes.

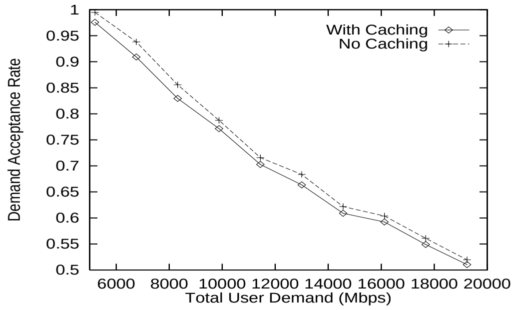


Fig. 3. Demand Acceptance Ratio

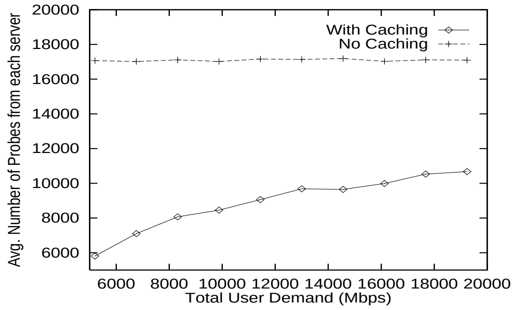
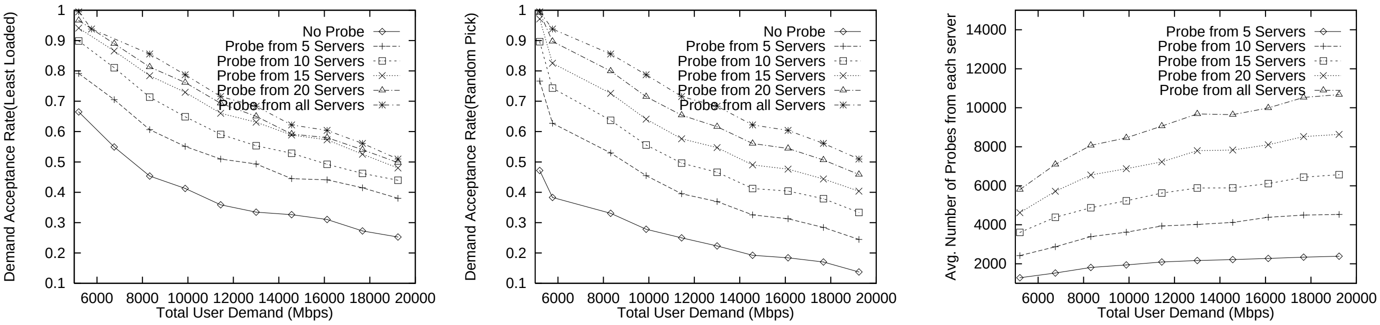


Fig. 4. Probing Overhead

Probing from a subset of Servers. To further reduce the probing overhead, we explore the influence of probing from a subset of all the replicated servers. Based on the philosophy of balancing the network load, we choose to probe from a subset of least loaded servers. In comparison, the experiment of choosing a random subset of servers is carried out as the baseline.

Figure 5 (a),(b) shows the result of demand acceptance ratio when choosing different subset of servers. Choosing the least loaded servers outperforms choosing random set of servers. This advantage is more apparent when the subset contains fewer



(a)Probe from least loaded servers (b) Probe from random servers (c) Probing overhead comparison

Fig. 5. Demand Acceptance Ratio Comparison, and Probing Overhead

servers. This indicates that choosing the least loaded servers has a better chance to include the globally best servers than random selection. Figure 5(b) shows the demand acceptance ratio if we choose to probe from different number of servers. The demand acceptance ratio increases as the number of servers participating server side probing increases. When the number of servers that participate probing is small, increasing the number of probing servers can significantly increase the demand acceptance ratio. As the number of probing servers increases, the improvement in demand acceptance ratio becomes slower. When the subset is small, increasing the size of this subset may include *better* servers participating in the probing. As the subset grows to a certain size, there is a good chance that the *globally best* servers are already included, thus adding servers to the subset makes little difference. Figure 5(c) shows the probing overhead when we start probing from a different number of servers. The caching mechanism as described in the previous section is used in this experimentation. The ratio of probing overhead between a smaller subset and a larger subset is more than the ratio of the size of these two subsets. If we start probing from fewer servers, the client demand satisfaction rate will be lower, thus triggering more server side probing.

V. Conclusion

In this paper, we have presented a probe-based protocol to support server selection in a differentiated service network. We have evaluated several related techniques, such as path selection algorithms, caching architecture, and probing from a subset of servers. We draw following conclusions to finish this paper.

- Among all the server selection policies, the shortest path first heuristic gives the best demand acceptance ratio and resource usage. The reason is that choosing the shortest feasible path tends to use less network resources, leaving more available resources for future use.
- Effective caching structure can greatly reduce the probing overhead with little degradation in performance. Even though cached path information and actual path information may be occasionally inconsistent, the performance will not be harmed, as long as we trigger the cache update properly.
- We find that probing from an appropriate subset of server domains can significantly reduce the probing overhead, while incurring a tolerable performance degradation.

References

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

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

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

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

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

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

[7] C.Chuah, R.H.Katz, and A.D. Joseph. QoS provisioning using a clearing house architecture. *International Workshop on Quality of Service*, 2000

[8] 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.

[9] 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.

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

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

[12] F.Hao, E.W.Zegura, M.Ammar. Supporting server selection in differentiated service networks. *Proceedings of IEEE INFOCOM*, 2000.

[13] E.D.Katz, M. Butler, and R.McGrath. A scalable HTTP server: The NCSA prototype In *Computer Networks and ISDN Systems*, vol 27, 1994

[14] Q.Ma, P.Steenkiste. On path Selection for Traffic with Bandwidth Guarantees In *Proc. of IEEE ICNP* 1997

[15] 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.

[16] 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.

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

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

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

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

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

[22] 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.