IEICE TRANS. COMMUN., VOL.E87–B, NO.7 JULY 2004

PAPER

Active Anycast Technique that Achieves Capacity-Aware Load Balancing for Heterogeneous IP Networks

Satoru OHTA[†] *and* **Toshiaki MIYAZAKI**[†]**,** *Members*

SUMMARY Real-world IP networks are heterogeneous in terms of server and link capacities. A sophisticated and comprehensive load balancing method is essential if we are to avoid congestion in the servers and links of heterogeneous networks. If such a method is not available, network throughput is limited by bottleneck servers or links. This paper proposes an anycast technique that achieves load balancing under heterogeneity. The proposed method well suits implementation on active networks. By taking advantage of the processing ability provided by active nodes, the method can decide packet routes flexibly on the basis of various criteria to realize a variety of load balancing schemes. Some of these schemes can successfully prevent the congestion of heterogeneous networks by tackling bottlenecks in both server and link capacities. The method is also advantageous given its light control load even when using many mirrored servers. Computer simulations confirm the effectiveness of these features.

key words: *anycast, Internet protocol, load balancing, active network, simulation*

1. Introduction

Server congestion must be avoided if IP (Internet Protocol) networks are to provide good service quality. Load balancing by setting many mirrored servers is a well-known way of decreasing the possibility of server congestion. Besides server congestion, link or network congestion also degrades the service quality provided by networks. Therefore, the best load balancing technique should prevent traffic from concentrating on particular servers or network resources.

Given the importance of load balancing, a considerable number of methods have been proposed and introduced to networks. The simplest method is to use round-robin address search in the DNS (Domain Name System) [1], [2]. It is also possible to introduce proxy servers that pass a service request to an appropriate mirror server [3]. Reference [4] evaluates a method that uses a device called the HTTP scheduler, which balances server load by rewriting IP packet headers, for distributing World Wide Web load. One technique that can be used to balance loads over mirror servers is known as "anycast." The standardized version of anycast is specified in Ref. [5]. Several variations of the anycast technique have been also proposed [7]–[9].

Real world IP networks are heterogeneous in terms of server and link capacities. In such networks, existing load balancing techniques may not work successfully. If the servers have different capacities and the load balancing mechanism employed offers the same load to each server, the servers with smaller capacities will be congested. Some existing methods can offset this problem to a certain degree. For example, the round-robin search in the DNS can adjust server load by appropriately ordering the address list [1]. Reference [2] presents an improvement on the load balancing method based on the DNS that achieves more sophisticated control. However, it is difficult for such methods to distribute load perfectly in the face of heterogeneous server capacities. Similarly, the network may contain links with a mixture of high and low capacities. In such a situation, heavy loads should not be routed through low capacity links. A flexible and sophisticated load balancing mechanism is needed to deal with the heterogeneity expected of server and link capacities.

Anycast is more powerful in a realistic heterogeneous network. However, several additional difficulties must be addressed if we are to use anycast to realize an advanced load balancing technique. For example, some mechanism is needed to apply anycast to the stateful higher layer protocols such as TCP (Transmission Control Protocol) because anycast may send the packets of the same connection to different destinations [6]. In addition, Ref. [6] points out the possibility of packet looping if anycast is implemented with existing routing tables and if multiple paths can be used for packet routing. Moreover, when the technique is introduced into an operating IP network, the protocol stack implemented in clients and servers should not be changed for smooth deployment of the technique. This is because it is not realistic to renew the protocol stack for a large number of existing clients or servers.

The active network technology was applied to the load balancing problem in Ref. [9]. This approach is interesting because it distributes loads according to server performance through the power of active network technology. With this approach, however, each active node must measure and compute the performance of every duplicate server. This is not easy when the network has many servers.

This paper presents a new anycast protocol that well suits implementation on an active network. The protocol can distribute loads across servers according to various criteria by executing different computational procedures at active nodes. By exploiting an appropriate computational procedure, the method is able to balance the load among heterogeneous server and link capacities. The active network technology well supports the proposed method through its node computation function and ease of activating the computational procedure. Additionally, the control loads placed

Manuscript received September 26, 2003.
Manuscript revised March 5, 2004.

[†]The authors are with the NTT Network Innovation Laboratories, NTT Corporation, Yokosuka-shi, 239-0847 Japan.

on active nodes do not increase greatly even if many mirror servers are used. Another feature is that server addition or removal is automatically managed by table manipulation at active nodes. These features are confirmed by computer simulations.

The paper first describes the technical problems in Sect. 2. Section 3 provides full details of the protocol. Next, the simulation model is explained in Sect. 4. Section 5 presents the simulation results. Some technical issues concerned with the proposed method are discussed in Sect. 6. Finally, Sect. 7 concludes the paper.

2. Technical Requirements

Anycast is a communication style that sends an IP datagram from the source to any one of a group of multiple hosts. This communication style is useful to balance the load offered by clients among multiple servers that offer the same service. Reference [5] specifies the standardized version of anycast. However, this version and the conventional variants are not ideal.

Real world IP networks are almost always heterogeneous in terms of server and link capacities. If demand for some service grows and if mirror servers are added, the hardware and capacity of new servers may well be different from those of the old servers. Similarly, link capacities may be mixed in a network. Traffic demand may also be geographically heterogeneous.

As specified in [5], the standardized anycast sends packets from a client to the nearest server. However, it is obvious that this packet forwarding may not provide successful load balancing in a heterogeneous network. Suppose that there are two servers; one has a small capacity while the other has a large capacity. If the small capacity server is closer to many clients, and if the large capacity server is closer to a few clients, the small capacity server is very likely to be congested by the heavy service request load from the many clients. Meanwhile, the large capacity server is likely to be underutilized. This mixture of congestion and underutilization must be avoided from the viewpoint of achieving service quality as well as reducing unused resources. A similar situation happens for mixed link capacities. Namely, the link capacities on the route to the closest server may be too small, while the link capacities on the route to the distant server may be sufficiently large. For this case, congestion and underutilization occur on the respective links. For effective load balancing in heterogeneous networks, therefore, anycast should distribute loads on the basis of server and link capacities instead of the hop distance between a client and the server.

To solve the above problem, the anycast load balancing technique may have to send packets to a server that is not closest from the client. However, this causes another problem for the standardized anycast. Namely, multi-path anycast routing may generate packet looping if a longer path is selected from multiple paths. This is described in [6]. Therefore, care must be taken to avoid packet looping for the development of an anycast protocol, which may necessitate routing traffic to the longer path to ideally balance load and utilize resource capacities.

Stateful higher layer protocols support many of the services currently offered in the IP networks. The best known stateful protocol is TCP, which is indispensable for many attractive services. A stateful protocol utilizes a connection or session set up between two communicating hosts. For the correct operation of the protocol, the packets that belong to a connection must be exchanged exclusively between these two hosts; it is forbidden to send them to any other hosts. Meanwhile, anycast may send packets to any of multiple hosts. Thus, a stateful protocol may not operate correctly with anycast because packets may be sent to an inappropriate host. Therefore, if anycast is to be applied to load balancing for services based on the stateful protocol, some mechanism is needed to ensure that the packets of the same connection are delivered to the correct server. Fortunately, such mechanisms have been studied comprehensively [6]. This means that we can employ the most appropriate of those mechanisms when a newly developed anycast technique is applied to stateful protocols.

This paper focuses on an extended anycast protocol that satisfies the above technical requirements: load balancing for network heterogeneity, packet looping avoidance, and stateful protocol support.

3. Proposed Protocol

3.1 Overview

The proposed protocol assumes a network composed of active nodes, server nodes, client nodes, and optional legacy routers as shown in Fig. 1. Servers are categorized into groups, each of which is a set of servers. Servers that belong to the same group offer the same services. This paper studies the protocol that sends the packet from a client to one server in the group so as to maximize the throughput (or minimize the response time) for the given link and server ca-

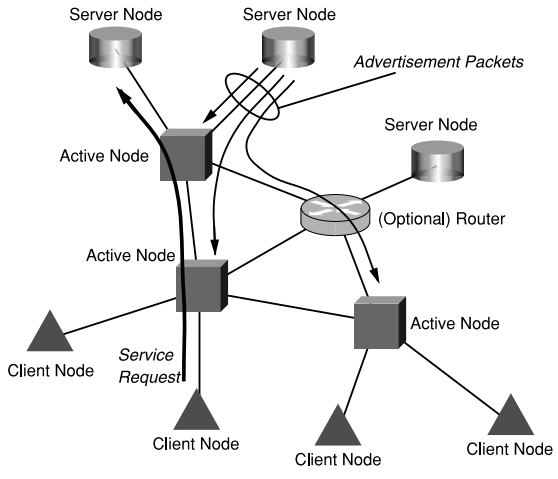

Fig. 1 Network configuration for the proposed anycast protocol.

pacities. Each group is called an "anycast group" hereafter.

The study assumes that each server has a unique unicast IP address. To identify the group of functionally equivalent servers, the unicast address of one server is used. The unicast address selected for group identification is called the "anycast address" hereafter. Note that the addressing scheme in the study differs from the standardized "anycast address [5]." Each active node maintains the relationship between the anycast address and the addresses of the other servers in the group.

Each active node acts as a transit node and maintains routing tables. A routing table has entries, each of which includes an element called "weight" as well as the address of the next hop. When a service request from a client arrives at an active node, the next hop is determined by selecting a table entry through some computation on the weight element values. In the proposed protocol, the next hop is another active node or a server node.

To manage routing tables and to set weight element values appropriately, the protocol uses "advertisement" packets. A server sends advertisement packets to each active node periodically. An advertisement packet carries a field called "measure," which active nodes use to compute the weight element values stored in the routing table entries. By putting server or link capacity information into the measure field of an advertisement packet, it becomes possible for an active node to determine the weight element so as to reflect server and link capacities.

When an active node receives an advertisement packet, it updates the associated routing table entry. First, the node that sent the packet is recorded as the next hop in the routing table entry. When an active node forwards an advertisement packet, it replaces the source address of the packet with its own address. Thus, the chain of the next hops written in the tables from the advertisement destination to the source forms a route of the advertisement packets in the reverse direction. Simultaneously, the node calculates the weight element value from the measure field in the received advertisement packet. The weight value computation depends on the load balancing scheme used as detailed in Sect. 3.2. An active node forwards the received advertisement packet to its destination if necessary. This forwarding process is detailed in Sect. 3.5.

3.2 Load Balancing Schemes for Heterogeneous Networks

3.2.1 New Schemes

The proposed protocol can achieve sophisticated load balancing schemes by putting capacity information into the measure field of the advertisement packets and the weight elements of routing table entries. This approach is expected to successfully handle the heterogeneity of real world networks. Such schemes include the following.

(1) Smaller Capacity Scheme

This scheme sends a service request to a server considering the capacities of servers and links. Initially, a server sets measure value m of an advertisement packet to its own capacity. It is assumed that the capacity is represented by the bit rate that the server can handle. Upon the arrival of an advertisement packet, the active node compares value m with the capacity of the link from which the packet came. Next, the node sets the weight element value of the associated table entry to the smaller of value m and the link capacity. When forwarding an advertisement packet, the active node sets m to the sum of the weight element values in the table. The active node randomly routes a service request packet to the node indicated in an entry that is selected with probability proportional to its weight element value. Namely, if w_i is the weight element value of the i-th entry, the node shown in the entry is selected with probability $w_i / \sum_j w_j$.

If server capacity is small and becomes a bottleneck, the scheme assigns less load to the server. This is because the entry selection probability is proportional to the smaller of the link capacity and the server capacity indicated in m. Similarly, if a link is a bottleneck, the scheme assigns less traffic to the link.

(2) Processing Time Scheme

This scheme is the same as the Smaller Capacity Scheme except for the computation of the weight element in the table entry. If an active node receives an advertisement packet, the weight element in the routing table entry is set to value, w.

$$w = \frac{1}{1/m + 1/C_l}, \tag{1}$$

where C_l is the capacity of the link on which the advertisement packet arrived and m is the measure field value of the advertisement packet.

In Eq. (1), $1/C_l$ is the time needed to send 1 bit over the link, while $1/m$ is the time needed to process 1 bit at the descendants of the next hop. Thus, the denominator of the equation roughly approximates the processing time per bit if the entry is selected. Value $1/m$ is large if the server has very small capacity, and thus weight w becomes small. Thus, a light load is given to the server. A small capacity link is also unlikely to be congested with the same mechanism.

3.2.2 Emulation of Existing Schemes

The proposed protocol can provide various load balancing schemes by altering the measure field setting for advertisement packets, the weight computing procedure, and the next hop selection for service request packets. Thus, the protocol can emulate existing load balancing schemes as follows.

(1) Minimum Hop Scheme

With this scheme, a service request is sent to the nearest mirror server as with the standardized anycast specified in

[5]. To do this, a server sets the measure field of an advertisement packet to 0. Let m denote the measure field value of a packet. When an advertisement packet arrives at an active node, the weight of the associated table entry is set to $m + 1$. Before the active node forwards the advertisement packet, value m is replaced by the minimum weight value found among all entries in the table. The service request packet is sent to the next node specified in one of the table entries that share the minimum weight value.

(2) Equal Load Scheme

This scheme sends a service request to an arbitrary server with equal probability. Thus, the scheme acts like round-robin table look-up in DNS. In this scheme, the measure field of an advertisement packet is set to 1 at a server and the weight element sum in the table at an active node. Then, the weight element value in the table is set to the measure value of the received advertisement packet. For these measure field and weight element settings, the next hop of service request packet is selected randomly from the table entries with probability proportional to the weight element value.

3.3 Loop Avoidance

As described in Sect. 2, packet looping must be avoided to develop an effective anycast technique. The proposed method achieves this goal by means of the routing table structure.

The protocol creates and manages a distinct routing table for each pair of advertisement packet destination and anycast group. Thus, the set of routes associated with a table forms a rooted and directed tree because the table includes advertisement routes destined for one particular active node. The root of the tree is the destination of advertisement packets while the leaves are the sources of advertisement packets. Since a tree can not contain loops, looping is prevented if a packet route is selected from the tree [11].

To route a service request along a path in the tree, an active node must identify which tree should be used. For this purpose, a service request packet must identify the tree root. A simple way of indicating the root address in the packet is to encapsulate the original packet and the root address into a new packet.

Specifically, the first active node to receive a service request from a client, encapsulates the packet and its own address into a new packet. Since this active node becomes the root of the tree, its address is the root address. The node then finds the table associated with the pair of the anycast group and the root address. The next hop is selected from the entries in the table, and the destination of the new packet is set to the address of the next hop. The source address of the packet is set to be the same as the original packet. The new packet is then forwarded. An active node that receives the packet looks for the table associated with the pair of the anycast group and the root address. This is executed by checking the encapsulated root address and original packet. Then, the packet is forwarded and its destination is rewritten in the same way as above. This forwarding process is repeated until the next hop is found to be a server. If the next hop is a server, the active node recovers the original packet and its destination address is replaced with the server address. In this way, a service request packet is forwarded from the root to a leaf (i.e., server) of a particular tree.

For the response from the server to the client, an active node replaces the source address of the packet with the anycast address. With this header translation, the client can communicate with the selected server as if it is communicating with the server that has the anycast address.

3.4 Support of Stateful Protocols

Load balancing with the anycast technique requires some mechanism for the stateful upper-layer protocol, for example, TCP. Fortunately, Ref. [6] comprehensively studied a method that forwards the packets of a connection to a fixed server. The methods shown in [6] include the hash-based approach as well as the approach of applying anycast to the TCP SYN packet and using the Source Routing IP option. Among these approaches, the latter approach has a shortcoming in that the protocol stack of every TCP connection receiver (typically server) must be modified. Meanwhile, the hash-based approach is more advantageous in that no changes are needed in the protocol stack for senders and receivers.

The hash-based approach described in Ref. [6] uses a hash function that takes the source address/port number as its input. The output of the function is used as a key to select the next hop from multiple candidates. This approach must be modified to be applied to the proposed protocol. First, it is not adequate to employ the source port number as hash input because some applications (for example, FTP) may use multiple ports. More importantly, an active node must select the next hop of a service request randomly with a specified probability in the schemes described in the previous section. Therefore, an appropriate mechanism should be established to achieve the specified selection probability. Thus, this study alters the hash-based approach to support probabilistic next hop selection as follows.

Assume that an active node has I entries $0, 1, \cdots, I - 1$ in the table associated with a received service request packet. Let s denote the source address of the received service request packet. Let us define a hash function $h_j(s)$ for active node j. Hash function $h_j(s)$ takes source address s and outputs integer value $x \in [0, X - 1]\,(X \gg I)$. Let p_i denote the probability assigned to entry $i \in \{0, 1, \cdots, I - 1\}$. Additionally, let us define value P_i as follows.

$$P_i = \begin{cases} 0 & \text{for } i = 0 \\ \displaystyle\sum_{k=0}^{i-1} p_k & \text{for } 0 < i \leq I. \end{cases} \tag{2}$$

Then, suppose that active node j selects the next hop from entry i, since the hash function output satisfies the following inequality for the source of the received packet.

$$P_i \leq h_j(s)/X < P_{i+1}. \quad (3)$$

It is obvious that the length of interval $[P_i, P_{i+1})$ is p_i. Meanwhile, if there are many clients and if hash function $h_j(s)$ maps s of to $[0, X-1)$ uniformly, value $h(s)/X$ approximates a uniformly distributed random variable in $[0, 1)$. Thus, $h_j(s)/X$ falls to interval $[P_i, P_{i+1})$ with probability p_i. As a result, entry i is selected with probability p_i. Simultaneously, the selected entry is constant for a particular source destination pair because value $h_j(s)$ does not change. Therefore, packets that belong to the same connection are routed to the fixed next hop as long as p_i does not change.

The computational complexity of the above process must be small because it is executed on a per-packet basis. However, this process can be executed in $O(1)$ time, and thus the complexity is not a serious problem. For $O(1)$ time computation, it is necessary to define integer $a_x (x = 0, 1, \cdots, X-1, 0 \leq a_x < I)$ for each table. Value a_x maps from hash output x to entry index $i (0 \leq i < I)$. Namely, a_x is set to satisfy the following.

$$P_{a_x} \leq x/X < P_{a_x+1}. \quad (4)$$

If active node j calculates $x = h_j(s)$ for a received packet, entry i is immediately obtained by setting $i = a_x$. It is not necessary to calculate value a_x frequently. The calculation, shown in Eq. (4) need be executed only at some appropriate interval T_{check}. Interval T_{check} can be sufficiently long within the limit of losing weight value adaptability and thus making probability p_i inappropriate.

Another problem with the hash-based approach is that modifying the weight values changes the packet route of a connection. An easy way of avoiding this difficulty is to introduce a time stamp, $t_x (x = 0, 1, \cdots, X-1)$, for each table. When an active node receives a service request packet, time stamp t_x is set to the current time for hash value $x = h_j(s)$. Then, when value a_x is updated and changed from the previous value, t_x is checked. If t_x is very close to the current time t_{now}, it is considered that there exists a connection between the source and destination pair corresponding to hash value x. Thus, value a_x should not be changed which fixes the route of the existing connection. Value a_x should be updated only if difference $t_{\text{now}} - t_x$ is greater than specified threshold t_{th}. This simple rule prevents packets of a connection from having their destinations changed even if the weight value is updated.

3.5 Routing Table Management

As outlined in Sect. 3.1, routing tables are managed through the advertisement packets. Table entries are also created and deleted through the advertisement packets. When an active node receives an advertisement packet, it searches for the routing table associated with the destination (i.e., root) and anycast addresses specified in the advertisement packets. Then, for the discovered table, the node looks for the entry such whose next hop element coincides with the source of the packet. If such an entry is not found, the node creates a new entry whose next hop element is the source of the advertisement packet. Thus, the hop element of a table entry exactly points to the previous node on the advertisement route.

To delete an unnecessary entries adequately, each table entry has a time stamp element. Every time the active node receives an advertisement packet, the time stamp is updated for the table entry associated with the packet source. If the time stamp for a table entry is not updated within a specified timeout period T_{del}, the table entry is deleted from the table. Thus, if a server stops sending advertisement packets, the table entry that indicates the server as the next hop is deleted. The deletion is executed by setting p_i at 0 if entry i is to be deleted. After the existing connections are cleared, a_x's are updated such that $a_x \neq i$ for any hash output x. This means that entry i is not selected as the next hop. As a result, the distribution of the service requests to the server also stops.

The timeout period T_{del} must be determined so as not to delete the entry associated with the source that sent the advertisement packet. The appropriate T_{del} value depends on the mechanism used to forward advertisement packets.

The time stamp of a table entry is checked periodically with an appropriate interval T_{check} to determine whether the entry is to be deleted or not. The interval T_{check} should be sufficiently large to reduce the processing load of the node.

In the above forwarding process, if the node forwards every received packet, the destination active node may receive a unacceptably large number of advertisement packets. This is because the packets are sent from possibly many servers to the active node. Since this advertisement packet concentration may overload the node, an active node should select the advertisement packets to be forwarded. This selection is carried out by putting another time stamp on each routing table. This time stamp is updated every time the node sends an advertisement packet for the anycast and root addresses associated with the table. When a transit active node receives an advertisement packet, it compares the current time and the time stamp value. Let T_{diff} denote this difference. If value T_{diff} is not less than a specified value D, the node forwards the packet. Otherwise, the packet is discarded. Clearly, this rule prevents the rate of forwarded advertisement packets for a pair of anycast address and root address from exceeding $1/D$.

Suppose that active nodes execute the above procedure, and a server sends advertisement packets with interval D for one pair of root address and anycast address. The following theorem is then obtained.

Theorem 1: Let r_a denote the rate of the advertisement packets sent by an active node for one pair of anycast address and root address. In addition, let N denote the number of active nodes. If an active node forwards an advertisement packet for the case of $T_{\text{diff}} \geq D$,

$$1/ND < r_a \leq 1/D. \quad (5)$$

Proof It is obvious that $r_a \leq 1/D$ because the interval between two successive advertisement packets is not less than D. For $1/ND < r_a$, suppose that the active node receives

advertisement packets with interval T from another node, denoted A. The interval between two transmitted advertisement packets is not larger than $T + D$. To confirm this, assume that the time stamp value is t_{last}. From the above procedure, no advertisement packet is forwarded for time t such that $t_{\text{last}} < t < t_{\text{last}} + D$. However, the node receives at least one advertisement packet from node A for time t such that $t_{\text{last}} + D \leq t < t_{\text{last}} + D + T$. The active node forwards the first advertisement packet that arrives within this range of t. Therefore, the interval is less than $(t_{\text{last}}+T+D)-t_{\text{last}} = D+T$. If node A is a server, T is clearly equal to D. Thus, the interval between advertisement packets is less than $2D$. By repeating this observation, it is easy to see that the interval between packets is not larger than $(h + 1)D$, where h is the number of active nodes on the path from the server to the node. Since h is less than $N - 1$, the interval is less than ND. This proves $1/ND < r_a$. (*End of Proof*)

Consider that $T_{\text{del}} = ND$ and no advertisement packets are lost. Then, from the above theorem, a child node is not deleted as long as it transmits advertisement packets. It is safe to set T_{del} to several times ND if frequent packet loss is expected.

In the above forwarding procedure, an active node forwards only some of the advertisement packets. This means that some of the information carried by the advertisement packets is dropped at the node. Such information dropping is acceptable because the active node only needs to know the capacity information of its direct children for routing and does not need to know any information about the children's descendants. Nevertheless, Theorem 1 shows that the interval between two advertisement packets can be as large as ND. This implies that routing table updating for server addition or removal may become slow if N is large. Therefore, it should be noted that the proposed method is not adequate for a network that has an excessive number of active nodes.

4. Evaluation Model

The effectiveness of the proposed method was evaluated through computer simulation. The evaluation was done by simulating the file downloads in a heterogeneous IP network.

The simulation model was built on a commercial simulation platform [12]. The protocol model was built so as to support a stateful higher layer protocol by the method stated in Sect. 3.4. As a result, the TCP (Transmission Control Protocol) model was successfully executed over the proposed anycast model. Note that the FTP (File Transfer Protocol) model provided by the platform also works correctly with the proposed method.

A block diagram of a server node is outlined in Fig. 2. The model has a mechanism that issues advertisement packets periodically. It also has a buffer queue to simulate a limited processing capacity. By setting the service rate for the queue, it becomes possible to set an arbitrary server processing rate.

Figure 3 outlines the active node model. The active node model features the IP routing unit, output queues, and the processing unit that executes the protocol.

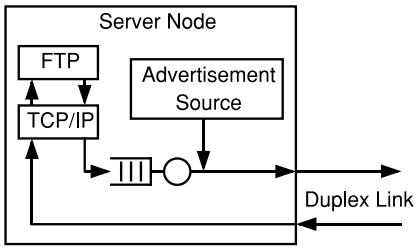

Fig. 2 Conceptual structure of the server node model.

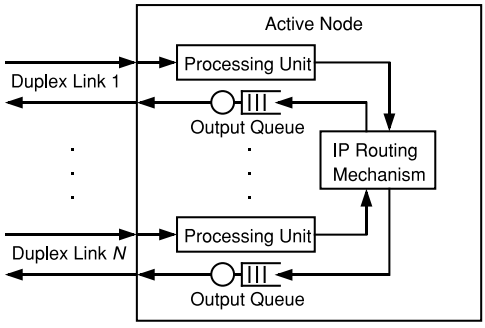

Fig. 3 Conceptual structure of the active node model.

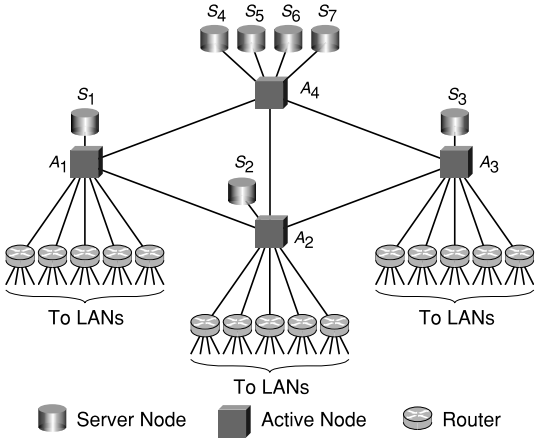

Fig. 4 Network model.

For client nodes, the model provided by the simulation tool is used without modification. The model simulates a node that randomly requests file downloads over FTP.

The network model was built by connecting these node models with links. Figure 4 shows the configuration of the network model. As shown in the figure, the model consists of 4 active nodes, 7 server nodes and 15 routers. Each router is connected to 4 LANs (Local Area Networks), each of which is composed of a hub and 16 client nodes. Thus, the network model has 960 client nodes. 10 Mb/s 10 BaseT cables were assumed as the connections between a router and hubs as well as between a hub and client nodes. The capacities of other point-to-point links, which connect servers, active nodes and routers, were randomly determined to model

a heterogeneous situation. Server processing capacity was also random. The capacity of each server and link was randomly selected from $1, 2, \cdots, 10$ (Mb/s) with equal probability.

For the above network model, the client nodes randomly requested file download. The file size was randomly determined according to an exponential distribution where the average size was 5×10^5 bytes. The interval between the client requests was also random according to the exponential distribution. Modifying the average of this interval varied the total traffic load. For the probabilistic next hop selection described in Sect. 3.4, parameter X was set at 997. The advertisement packet interval D was set at 1 (second). In addition, parameter T_{check} was 120 (seconds).

When using the above model, the protocol was evaluated in two simulation scenarios. The first simulation scenario compared the load balancing schemes mentioned in Sect. 3.2. For comparison, the average download time was measured while varying the total traffic load. For the load balancing schemes that perform well, the average download time will increase less for the larger traffic load than for schemes with poor performance. This is because congestion is likely to occur in servers and links if the load balancing scheme performs poorly. Thus, the download time is considered to be a good measure for assessing the effectiveness of the load balancing schemes.

The second simulation scenario focused on the control load versus changes in the number of servers; the validity of the table entry management technique described in Sect. 3.5 was also confirmed. The purpose of this scenario is to confirm that the control load does not increase greatly with server number as well as ensuring that the protocol allows servers to be smoothly added or deleted. In the scenario, servers S_1, S_3 and S_5 shown in Fig. 4 do not send advertisement packets and thus do not receive download requests at the start of simulation. Then, these servers start sending advertisement packets at $t = 10$ (min), $t = 20$ (min) and $t = 30$ (min) respectively, where t is the simulation time. Server S_5 stops sending advertisement packets at $t = 40$ (min). For this server addition and deletion, the rates of advertisement packets received by active nodes A_1 and A_3 were recorded. Since the control load depends on the number of advertisement packets received, the scenario uses the packet rates as the control load measure. Additionally, the download bit rate was measured for server S_5 to confirm that the delivery of download request was correctly controlled through advertisement packets. Throughout the simulation, the interval between download requests was fixed to 900 seconds.

5. Evaluation Results

Figure 5 shows the results for the first simulation scenario, namely, the performances of the load balancing schemes mentioned in Sect. 3.2. The x-axis shows the average bit rate of traffic received by the clients of the whole network, while the y-axis shows the average download time for each scheme.

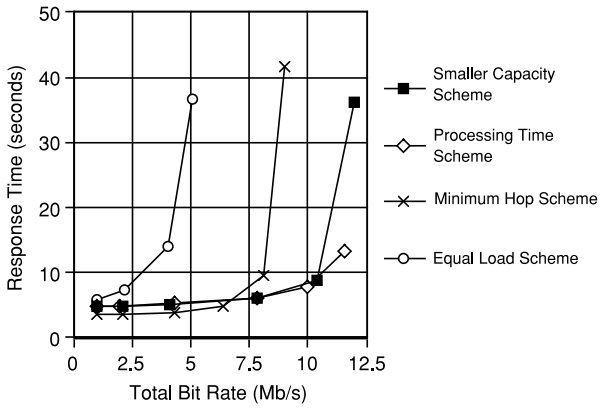

Fig. 5 Download response time versus average load for load balancing schemes.

The graph clearly shows that the Minimum Hop Scheme and the Equal Load Scheme do not perform successfully in a heterogeneous environment. With the Minimum Hop Scheme, the server closest to a client node does not always have sufficient capacity to meet the demand. Thus, a server with small capacity is very likely to be congested. In addition, even if a server has sufficient remaining capacity, the server capacity is not used by the client nodes that are not closest. This means that some server nodes are underutilized and thus the achievable network throughput is not large given the total server capacity. The Equal Load Scheme tries to offer the same load to each server. Apparently, this function places excessive loads on small capacity servers/links and underutilizes large capacity servers/links which yields strong congestion. Therefore, the Equal Load Scheme has worse performance than the Minimum Hop Scheme.

The Smaller Capacity Scheme and the Processing Time Scheme achieve larger throughput for a fixed response time than the other schemes. These schemes select the destination of a download considering the capacities of links and servers. Thus, links and servers are offered loads proportional to their capacities. With this operation, every link or server is unlikely to be congested. Therefore, the Smaller Capacity Scheme and the Processing Time Scheme perform better in heterogeneous networks. The proposed protocol must be used to execute these load balancing schemes because the capacity information exchanged by advertisement packets and the routing based on the weight values are essential for the schemes. Therefore, the characteristics shown in Fig. 5 prove the advantage of the proposed protocol in heterogeneous situations.

Figure 5 shows that the Processing Time Scheme has smaller response time than the Smaller Capacity Scheme for heavy traffic, i.e. the total bit rate is larger than 10 Mb/s. This result is probably due to the characteristic that the Processing Time Scheme routes more traffic to a closer server (See Appendix). Namely, the average number of hops between clients and servers tends to be smaller with

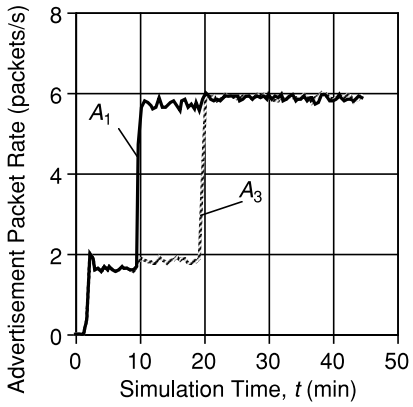

Fig. 6 Advertisement packet rates received by active nodes A_1 and A_3.

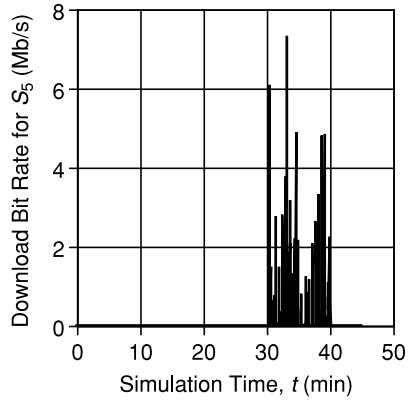

Fig. 7 Download bit rate from S_5 for the second scenario.

the Processing Time Scheme than with the Smaller Capacity Scheme. This average hop reduction may result in the smaller response time.

Figure 6 depicts the advertisement packet load measured in the second simulation scenario. The figure plots the rates of advertisement packets received by active nodes A_1 and A_3 versus simulation time t. The advertisement packet rate for A_1 increases only once at $t = 10$ (min): the time that server S_1 started sending advertisement packets. Meanwhile, the rate for A_1 does not change at the moments that servers S_3 and S_5 started sending advertisement packets. Similarly, the rate for A_3 increases only once when server S_3 became active. The rate for A_3 is insensitive for the start of S_1 and S_5. Thus, the advertisement packet load does not always increase with server addition. This is because the advertisement packet rate is bounded by the mechanism described in Sect. 3.5. This confirms that the proposed method is advantageous for the case of using many mirror servers from the control load viewpoint because control load does not become very large with server addition.

Other results obtained from the second scenario are shown in Fig. 7, which plots the download bit rate for S_5 against simulation time. As the figure shows, the download from S_5 started at $t = 30$ (min) and stops at $t = 40$ (min). In other words, the download was executed only while the server sent the advertisement packets. This suggests that the mechanism of using the advertisement packet to add and remove table entries works correctly. Thus, there is no need for setting any active nodes or routers when a server is started and stopped. This implies that the proposed method will decrease the operation costs for server addition and removal because no operations are needed within the network.

6. Discussion

Several technical issues must be resolved before the proposed method can be deployed in a network.

The first issue is the selection of the interval of advertisement packets, D. A too small value of D leads to excessively high rates of advertisement packets and thus unacceptably heavy processing loads. Meanwhile, if D is too large, the update of routing tables entries will become slow for server addition and removal. The analysis shown in Sect. 3.5 implies that the update time of routing table is the sum of T_{check} and ND in the worst case. From the viewpoint of network management, the update time of several minutes seems to be acceptable. This means that D can be set as large as 10 to 20 seconds. Obviously, this setting does not impose a serious control load on active nodes.

Second, it is necessary to assess the proposed method for the situation where active nodes are partially deployed in the network. For correct operation under this situation, the network must be constructed such that at least one active node exists on the route between a client and a server. If this condition is satisfied, the active node can route the service request from a client to a mirror server properly and the header of the response packet is translated successfully. On the other hand, if there is no active node between a client and a server, the service request may concentrate on a particular server with the anycast address and thus load is not balanced correctly. Moreover, the packet header may not be correctly written for the response packets. Therefore, if active nodes are partially introduced and coexist with legacy routers, the network must be planned carefully to locate active nodes between clients and the server. As long as such network planning is executed, the proposed method is applicable to a network composed of many legacy routers and a few active nodes.

7. Conclusion

This paper presented an extended anycast protocol that well implements load balancing in heterogeneous IP networks. By exchanging capacity information through advertisement packets and the use of weight based routing, the proposed anycast protocol achieves load balancing schemes that consider both server capacities and link capacities. These schemes perform successfully in a heterogeneous networks because the resources are fully utilized. This feature was confirmed through computer simulations. The proposed protocol is essential to execute these advanced load balancing schemes.

Another feature clarified by the simulations is that the control message load grows slowly with server number. This suggests that the proposed protocol offers excellent scalability in terms of the number of servers supported. The simulations also showed that the requests to a server can be started or stopped through advertisement packets without setting active nodes or routers. This characteristic is also favorable in terms of lowering the operational expenditure for mirror server addition or removal.

Acknowledgment

The authors thank Dr. Kimio Oguchi for his encouragement.

References

[1] P. Mockapetris, "Domain names—Concepts and facilities," IETF RFC 1034, Nov. 1987.
[2] T. Brisco, "DNS support for load balancing," IETF RFC 1794, April 1995.
[3] D. Andresen, T. Yang, V. Holmedahl, and O.H. Ibarra, "SWEB: Towards a scalable World Wide Web server on multicomputers," Proc. IPPS'96, pp.850–856, Honolulu, April 1996.
[4] H. Bryhni, E. Klovning, and Ø. Kure, "A comparison of load balancing techniques for scalable web servers," IEEE Network, vol.14, no.4, pp.58–64, July/Aug. 2000.
[5] C. Partridge, T. Mendez, and W. Milliken, "Host anycasting service," IETF RFC 1546, Nov. 1993.
[6] E. Basturk, R. Engel, R. Haas, V. Peris, and D. Saha, "Using network layer anycast for load distribution in the Internet," Proc. Third Global Internet Mini-Conference, Sydney, Nov. 1998. http://ec.eurecom.fr/~basturk/papers/gi98.anycast.ps.gz
[7] S. Bhattacharjee, M.H. Ammar, E.W. Zegura, V. Shah, and Z. Fei, "Application-layer anycasting," Proc. Infocom'97, pp.1388–1396, Kobe, 1997.
[8] D. Katabi and J. Wroclawski, "A framework for scalable global IP-anycast (GIA)," Proc. ACM SIGCOMM 2000, pp.3–15, Stockholm, April 2000.
[9] M. Yamamoto, H. Miura, K. Nishimura, and H. Ikeda, "A network-supported server load balancing method: Active anycast," IEICE Trans. Commun., vol.E84-B, no.6, pp.1561–1567, June 2001.
[10] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden, "A survey of active network research," IEEE Commun. Mag., vol.35, no.1, pp.80–86, Jan. 1997.
[11] S. Ohta and T. Miyazaki, "Active load balancing for heterogeneous IP networks," IEICE Technical Report, IN2002-66, Sept. 2002.
[12] http://www.opnet.com

Appendix: Difference between the Schemes Proposed in Sect. 3.2.1

To see the difference between the two schemes proposed in Sect. 3.2.1, let us consider the following example of a distribution tree. There is an active node with two children. One child is the directly connected server. The other child is an active node, which is the first of a series of connected m active nodes $a_1, a_2, \cdots, a_m$. Here, active node a_i $(1 \leq i < m)$ has only one child a_{i+1} and the child of a_m is a server. Let us assume that the capacity of every link and every server is the same, C. With the Smaller Capacity Scheme, the weight element stored in the table of the observed active node is C for both children. Thus, equal load will be offered to the two servers regardless of the hop count between the node and each server. The Processing Time Scheme, by contrast, sets the weight element at $C/2$ for the directly connected server. Meanwhile, by applying Eq. (1) repeatedly, it is seen that the weight element is $C/(m+2)$ for the server connected through $a_1, a_2, \cdots, a_m$. Thus, the more distant server has a lower weight value. Since the next hop of a service request is selected with a probability that is proportional to the weight value, the node routes more traffic to the directly connected server than to the distant server. In this way, the Processing Time Scheme can reflect the hop count to the server on the traffic load. If more hops exist between a server and a client, the response time tends to increase. Therefore, the Processing Time Scheme may yield better average response time than the Smaller Capacity Scheme because more traffic is routed to the closer servers.

Satoru Ohta received the B.E., M.E., and Dr. of Engineering degrees from the Tokyo Institute of Technology, Tokyo, Japan, in 1981, 1983, and 1996, respectively. Since 1983 he has been with NTT, where he worked on the research and development of cross-connect systems, broadband ISDN, network management, and telecommunication network planning. He is currently involved in research on network protocols and network performance evaluation at NTT Network Innovation Laboratories, NTT Corporation. Dr. Ohta is member of the IEEE. He received the Excellent Paper Award in 1991 from IEICE.

Toshiaki Miyazaki is a senior research engineer, supervisor (a research group leader) of NTT network innovation laboratories. He received the B.E. and M.E. degrees in applied electronic engineering from the University of Electro-Communications, Tokyo, Japan in 1981 and 1983, and the Dr. Eng. degree from the Tokyo Institute of Technology in 1994. Since joining NTT in 1983, he has been engaged in research on VLSI CAD systems including high-level synthesis, logic design tools, and CAD frameworks. Since 1993, he has been involved in developing telecommunications-oriented FPGAs and their applications, and next generation networking protocols. Since 2003, he has been engaged in research on peer-to-peer communications and ubiquitous network environments. His research interests are in programmable hardware systems, adaptive networking technologies, and autonomous systems. Dr. Miyazaki is a member of IEEE and ISPJ.