Flexible Anycast Utilizing the Programmable Node Technique

Satoru Ohta and Toshiaki Miyazaki

NTT Network Innovation Laboratories, NTT Corporation,
1-1 Hikari-no-Oka, Yokosuka, 239-0847 Japan,
{ohta.satoru,miyazaki.toshiaki}@lab.ntt.co.jp

Abstract. The "anycast" communication technique is expected to be applied to many applications and so must be flexible enough to cope with different criteria. This paper presents such an anycast technique. The proposed technique achieves flexibility through node programmability, the key feature of the active network. The paper explores how the proposed technique can use the different computational procedures at the nodes to implement a variety of load balancing schemes. These include the conventional load balancing schemes as well as more sophisticated schemes that consider the capacities of both servers and links. The sophisticated schemes provide larger throughput in a heterogeneous network than the conventional schemes. This is confirmed through computer simulations. The proposed method makes such load-balancing schemes achievable because it provides a flexible and generic computation and information exchange mechanism. This fact confirms the advantage of the flexibility provided by the proposed method.

1 Introduction

In the communication style of "anycast [1]," one of a group of hosts is selected to supply a service. One popular application is load balancing among mirror servers [2]. In another application, multiple servers offer the same service at different prices and the network routes a request to the least-expensive server. Because of the many applications possible, it is important to establish a flexible anycast technique.

For appropriate host selection, a transit node must discover the next hop for a packet according to the appropriate criteria. To do this, the node must process the attributes associated with hosts and other network resources. The computation used greatly depends on the application. For the load balancing application, different computational procedures are required according to the policies of the network operators. Namely, if the operator wants to distribute load in proportion to server capacity, the route must be determined from server capacities. Alternatively, the operator may set server selection according to service quality. With this policy, the quality metric depends on the traffic characteristic of the service type and thus a distinct computational procedure must be designed for

each service type. If the anycast mechanism is used in applications other than load balancing, yet another procedure will be necessary.

As seen above, we need a variety of computational procedures for anycast. The quick service deployment provided by the active network [3] is an interesting approach to meeting this technical requirement. In particular, the node programmability provided by active networks enables us to modify the procedures easily. By means of this function, it becomes possible to develop a generic anycast protocol, that is flexible enough to support a wide range of applications.

This paper focuses on anycast from the viewpoint of the load balancing application. For load balancing, a considerable number of techniques are described in the literature and applied to networks. These techniques include approaches other than anycast. The simplest technique is to use round-robin address search in the DNS (Domain Name System) [4, 5]. It is also possible to introduce proxy servers that pass a service request to an appropriate mirror server [6]. Reference [7] evaluates a method that uses a device called the HTTP scheduler, which balances server load by rewriting IP (Internet Protocol) packet headers, for distributing World Wide Web load. Reference [2] comprehensively studied anycast as applied to load balancing. For the purpose of load balancing, several variations of the anycast technique have been proposed [8–10].

Real world IP networks are heterogeneous in terms of server and link capacities [11]. 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 its address list [4]. However, it is difficult for such a method to distribute load to perfectly match the 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.

This paper presents an anycast protocol that exploits node programmability. The protocol can flexibly support various applications. For the load-balancing application, the protocol can distribute loads across servers according to various criteria by executing different computational procedures at programmable nodes. By exploiting an appropriate computational procedure, the method is able to balance the load among heterogeneous server and link capacities. Namely, the method achieves larger throughput for a heterogeneous network than the existing load balancing schemes. The active network technology essentially supports the method through its node computation function and ease of deploying the computational procedures. This feature is confirmed by computer simulations.

The paper first describes the details of the proposed protocol in Sect. 2. Next, the simulation model is explained in Sect. 3. Section 4 presents the simulation results. Finally, Sect. 5 concludes the paper.

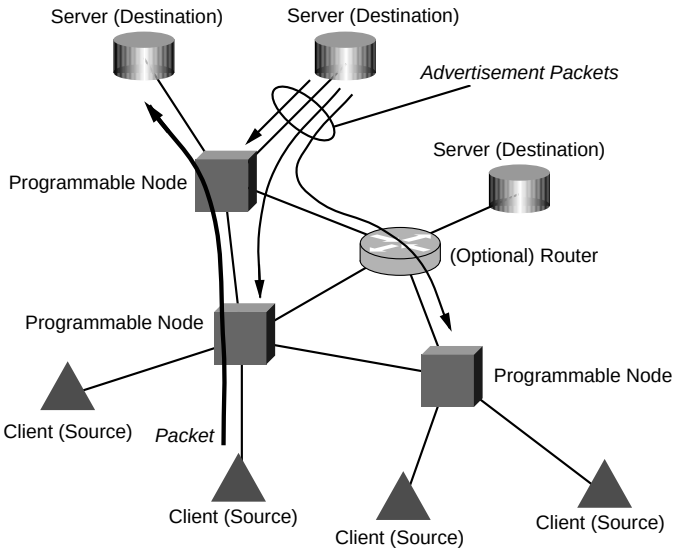

Fig. 1. Network configuration for the proposed anycast protocol.

2 Protocol

2.1 Overview

The proposed protocol assumes a network composed of programmable nodes, multiple candidate destinations, sources, and optional legacy routers as shown in Fig. 1. This paper details the application of the proposed anycast protocol for load-balancing among mirror servers. Namely, the source node is referred to as the client node while the destination nodes are referred to as the server nodes. Servers are categorized into groups. Every server in the same group offers the same services. The group considered hereafter is called the "anycast group."

The study assumes that each server has a unique unicast IP address. To identify a server group, the unicast address of one server in the group is used. This address is called the "anycast address" hereafter. Note that the addressing scheme in this study differs from the standardized "anycast address [1]."

Each programmable 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, which can be another programmable node or a server. When a service request from a client arrives at a programmable node, the next hop is determined by selecting a table entry through some computation on the weight element values.

To manage routing tables and to set weight element values appropriately, the protocol uses "advertisement" packets. A server sends advertisement packets to each programmable node periodically. An advertisement packet carries a field

called "measure," which the programmable 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 a programmable node to relate the weight element to server and link capacities. For other applications, it is possible to use different information (for example, the service price) as the field value and the weight elements.

When a programmable 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. Thus, the chain of next hops written in the tables from the advertisement destination to the source forms a route in the reverse direction. Simultaneously, the node calculates the weight element value from the measure field in the received advertisement packet. This weight value calculation depends on the load balancing scheme employed as detailed in Sect. 2.2. A programmable node forwards the received advertisement packet to its destination if necessary. This forwarding process is detailed in Sect. 2.5.

The proposed protocol is flexible since it can be modified by merely changing the computational procedures used: advertisement packet processing, weight value computation, and the next hop decision. Node programmability is essential for achieving this flexibility.

2.2 Load Balancing Schemes

Emulation of Existing Schemes. The proposed protocol can establish various load balancing schemes by altering the measure field setting in the advertisement packets, the weight computing procedure, and the next hop selection for service request packets. The proposed protocol can emulate existing load-balancing schemes as follows.

A. Minimum Hop Scheme. With this scheme, a service request is sent to the nearest mirror server as with the anycast specified in [1]. To do this, a server sets the measure field of an advertisement packet to 0. Let m denote the measure field value of an advertisement packet. When an advertisement packet arrives at a programmable node, the weight of the associated table entry is set to $m + 1$. Before the advertisement packet is forwarded by the programmable node, value m is replaced by the minimum weight value found among all entries in the table. The service request packet is sent to the node specified in one of the table entries that share the minimum weight value.

B. Equal Load Scheme. This scheme sends a service request to an arbitrary server with equal probability. Thus, the scheme performs in the same way as 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 is replaced by the sum of weight elements in the table at a programmable node. 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 the service request packet is selected randomly from the table entries with probability proportional to the weight element value.

Sophisticated Schemes. When an anycast technique is applied to load balancing, it is important to consider the heterogeneity of real world networks. That is, servers and links have diverse capacities. The load balancing mechanism must, therefore, avoid congestion in the face of such heterogeneity [11].

The proposed protocol can achieve more sophisticated load balancing schemes by putting capacity information into the measure field of advertisement packets. This approach is expected to handle the heterogeneity more successfully. Such schemes include the following.

C. Smaller Capacity Scheme. This scheme sends a service request to a server with 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 programmable 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 programmable node sets m to the sum of the weight element values in the table. The programmable 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 provides lighter 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 offers lighter traffic to the link.

D. 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. That is, if a programmable node receives an advertisement packet, the weight element in the distribution 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.

In (1), $1/C_l$ is the time needed to send 1 bit over the link, while $1/m$ is the time needed for processing 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.

2.3 Looping Avoidance

If we employ an anycast technique with the existing routing table structure and if we do not route a packet to the closest destination, packet looping becomes

possible. This mechanism is detailed in [2]. The proposed method avoids looping 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 programmable node. The root of the tree is the destination 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, a programmable node must identify which tree should be used. For this purpose, a service request packet must identify a 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. This allows each programmable node to identify the tree and the anycast address by checking the encapsulated root address and the original packet. This encapsulation must be executed at the first programmable node that the service request packet encounters. The destination address of the encapsulated packet is set to the next hop address indicated in the table at each programmable node. The packet is decapsulated at a programmable node if the next hop is found to be a server. The destination of the decapsulated packet is replaced by the address of the selected server.

As the response from the server to the client, a programmable 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.

2.4 Support of Stateful Protocols

Stateful upper-layer protocols, for example, TCP (Transmission Control Protocol), support many attractive network services. If such protocols are to operate successfully, the packets that belong to a connection must be exchanged exclusively between these two hosts and not sent to any other hosts. This suggests that anycast is incompatible with stateful protocols. Fortunately, Ref. [2] comprehensively studied the methods by which anycast can be made to forward the packets of a connection to a fixed server. The methods shown in [2] include the hash-based approach as well as the approach of applying anycast to TCP SYN packets and using the Source Routing IP option. Among these approaches, this study employs the hash-based approach because the protocol stack in servers and clients does not need to be modified.

The hash-based approach described in [2] uses a hash function that takes the source address/port number of a packet as its input. The output of the function is then used as a key to select the next hop for the packet. This approach must be modified if it is to be combined with the proposed protocol. First, it is not adequate to employ the source port number as the hash input because some applications, for example, FTP (File Transfer Protocol), may use multiple ports. More importantly, a programmable node must select the next hop randomly with a specified probability in the load balancing schemes described in Sect. 2.2.

Thus, the hash-based approach is altered for the probabilistic next hop selection as follows.

Assume that the programmable node has I entries $0, 1, \ldots, I-1$ in the table associated with the 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 each programmable node j. Function $h_j(s)$ takes the source address s and outputs an integer value $x \in [0, X-1]$ $(X \gg I)$. Let p_i denote the probability of entry $i \in \{0, 1, \ldots, I-1\}$ as the next hop. Additionally, let us define value P_i as follows.

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

The programmable node to selects the next hop from entry i, if the hash function output satisfies the following inequality with regard to the source of the received packet.

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

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 arriving packets to $[0, X-1)$ uniformly, value $h(s)/X$ approximates a uniformly distributed random value 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 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 may become a problem because it must be 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 $a_x (x = 0, 1, ..., X-1, 0 \leq a_x < I)$ 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.

$$P_{a_x} \leq x/X < P_{a_x+1} . \tag{4}$$

If programmable node j receives a packet and calculates $x = h_j(s)$, entry i is immediately obtained by setting $i = a_x$. It is not necessary to calculate value a_x frequently. Value a_x only has to be calculated at some appropriate interval T_{check}. Interval T_{check} should be sufficiently long within the limit of losing the adaptability to the weight value variations.

2.5 Routing Table Management

As outlined in Sect. 2.1, advertisement packets are used to manage the routing tables. They are also used in the creation and deletion of table entries. For

entry creation, a programmable node searches for the routing table associated with the destination (i.e., root) and anycast addresses specified in the received advertisement packet. Then, for the discovered table, the node looks for the entry such that its next hop element coincides with the source of the advertisement 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.

To delete unnecessary entries, each entry has a time stamp element. Every time the programmable 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, the table entry is deleted from the table. Thus, if a server stops sending advertisement packets, the entry associated with the server is deleted. The deletion is executed in accordance with the procedure described in Sect. 2.4 by setting p_i to 0 if entry i is to be deleted. This means that entry i is not selected for the next hop of a service request packet. As a result, the distribution of service requests to the server also stops.

3 Evaluation Model

The advantage of the proposed anycast protocol is that various load balancing schemes can be flexibly achievable. In particular, it is possible to develop load balancing schemes that consider the capacities of both links and servers. This advantage was evaluated by simulating the file downloads in a heterogeneous IP network. Namely, load balancing schemes based on the proposed method were applied to file download and their performances were compared.

The simulation model was built on a commercial simulation platform [12]. The protocol model was built so as to support the stateful higher layer protocol by the means stated in Sect. 2.4. Thus, the TCP model can be executed over the proposed anycast model. Accordingly, the FTP model provided by the platform works well. The evaluation used the FTP model to load the network and servers.

A block diagram of a server node is shown in Fig. 2. The model has a mechanism that issues advertisement packets periodically. Additionally, the model 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.

The programmable node model features an IP routing unit, output queues, and a processing unit that executes the protocol. For client nodes, the model provided by the simulation tool is used without modification. The client node model requests file downloads over FTP randomly.

The network model was built by connecting these node models as shown in Fig. 3. As shown in the figure, the network model consists of 4 programmable 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. 10Mb/s 10BaseT cables were assumed as router-hub connections as well as between hub and client nodes. The capacities of other point-to-point links, which connect servers, programmable nodes and

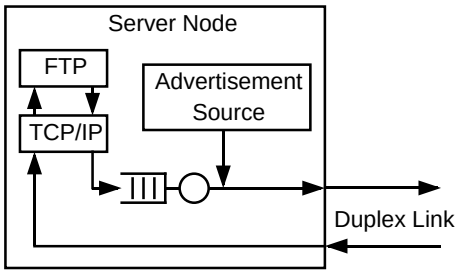

Fig. 2. Conceptual structure of the server node model.

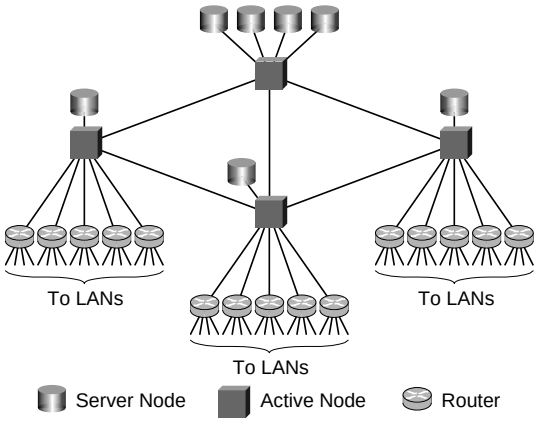

Fig. 3. Network model.

routers, were randomly determined to model a heterogeneous situation. Server processing capacity was also random. Namely, the capacity of each server and link was randomly selected from $1, 2, \ldots, 10$ Mb/s with equal probability.

The client nodes randomly requested file downloads. The file size was 5×10^5 Bytes on average and randomly determined according to an exponential distribution. The interval between the requests from a client node was also random according to an exponential distribution. The total traffic load was varied by modifying the average of the request interval. For the probabilistic next hop selection described in Sect. 2.4, parameter X was set at 997. The advertisement packet interval was set at 1 second. In addition, parameter T_{check} was 120 seconds.

The load balancing schemes mentioned in Sect. 2.2 were compared. In comparison, the average download time was measured for various total traffic loads. If a load balancing scheme effectively avoids congestion, the average download time will not grow greatly as the load increases. Thus, the download is a good way of assessing the effectiveness of the load-balancing schemes.

4 Evaluation Result

Figure 4 shows the simulation results for the load balancing schemes mentioned in Sect. 2.2. 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.

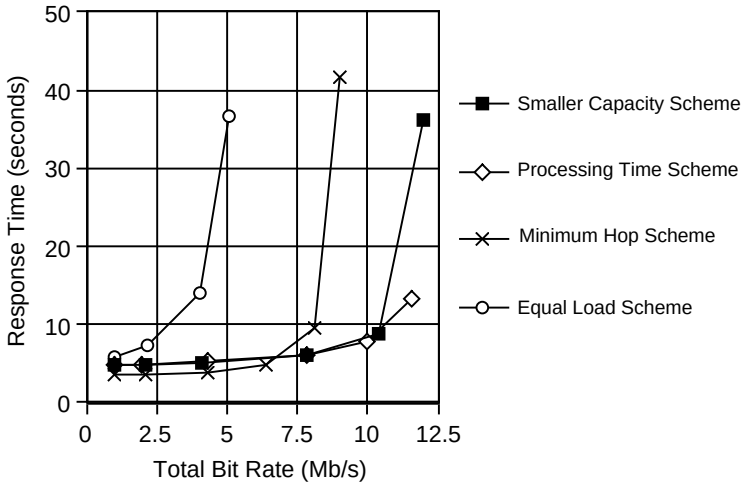

Fig. 4. Download response time versus average load for load-balancing schemes.

The graph clearly displays that the existing schemes called 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, servers with small capacity are 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 falls well under the total server capacity. The Equal Load Scheme tries to offer the same load to each server. This function appears to excessively load small capacity servers/links while under utilizing large capacity servers/links. Therefore, the Equal Load Scheme shows the worst performance due to congestion.

The Smaller Capacity Scheme and the Processing Time Scheme achieve larger throughput for a fixed response time than the other schemes. These schemes decide the server that is the destination of a download request after considering the capacities of links and servers. Thus, links and servers are offered loads proportional to their capacities. With this approach, few links or servers are likely to be congested. Therefore, the Smaller Capacity Scheme and the Processing

Time Scheme perform better in a heterogeneous network. 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 results shown on Fig. 4 prove the advantage of the proposed protocol for heterogeneous network situations.

5 Conclusion

This paper presented an extended anycast protocol based on node programmability. The proposed protocol employs weight based routing with the information exchanged through advertisement packets. By utilizing the computational ability and the flexibility provided by programmable nodes, various load balancing schemes can be implemented by the proposed method. In particular, it is possible to develop load balancing schemes that consider the capacities of both servers and links. These schemes perform more successfully in heterogeneous networks than existing schemes because the resources are more fully utilized. This feature was confirmed through computer simulations. The proposed protocol is attractive because its flexibility enables us to realize superior load balancing schemes.

References

1. Partridge, C., Mendez, T., Milliken, W., Host anycasting service. IETF RFC 1546 (1993)
2. Basturk, E., Engel, R., Haas, R., Peris V., Saha, D.: Using network layer anycast for load distribution in the Internet. In Proceedings of the Third Global Internet Mini-Conference, Sydney, (1998)
3. Tennenhouse, D.L., Smith,J.M., Sincoskie, W.D., Wetherall, D.J., Minden, G.J.: A survey of active network research. IEEE Communications Magazine, **35**, 1 (1997) 80–86
4. Mockapetris, P.: Domain names – concepts and facilities. IETF RFC 1034 (1987)
5. Brisco, T.: DNS support for load balancing. IETF RFC 1794 (1995)
6. Andresen, D., Yang, T., Holmedahl, V., Ibarra, O.H.: SWEB: towards a scalable World Wide Web server on multicomputers. in Proceedings of IPPS '96, Honolulu, (1996) 850–856
7. Bryhni, H., Klovning, E., Kure, Ø.: A comparison of load balancing techniques for scalable web servers. IEEE Network, **14**, 4, (2000) 58–64
8. Bhattacharjee, S., Ammar, M.H., Zegura, E.W., Shah, V., Fei, Z.: Application-layer anycasting. in Proceedings of Infocom'97, Kobe, (1997) 1388–1396
9. Katabi, D., Wroclawski, J.: A framework for scalable global IP-anycast (GIA). in Proceedings of ACM SIGCOMM 2000, Stockholm, (2000) 3–15
10. Yamamoto, M., Miura, H., Nishimura, K., Ikeda, H.: A network-supported server load balancing method: active anycast. IEICE Transactions on Communications, **E84-B**, (2001) 1561–1567
11. Ohta, S., Miyazaki T.: Active load balancing for heterogeneous IP networks. Technical Report of IEICE, **IN2002-66**, (2002) 79–84 (in Japanese)
12. http://www.opnet.com