On the Optimal Placement of Web Proxies in the Internet: The Linear Topology

Bo Li, Xin Deng and Mordecai J. Golin
Department of Computer Science, Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong.
Tel: +852 2358 6976, Fax: +852 2358 1477
E-Mail: {bli, dengxin, golin }@cs.ust.hk

Kazem Sohraby
Bell Laboratories, Lucent Technologies
101 Crawfords Corner Road, Holmdel, NJ 07733, USA.
E-Mail: sohraby@lucent.com

Abstract

Web caching or web proxy has been considered as the prime vehicle to cope with the ever-increasing demand for information retrieval over the Internet, WWW being a typical example. The existing work on web proxy has primarily focused on content based caching; relatively less attention has been given to the development of proper placement strategies for the potential web proxies in the Internet. This paper investigates the optimal placement policy of web proxies for a target web server in the Internet. The objective is to minimize the overall latency of searching the target web server subject to the network resources and traffic pattern. Specifically, we are interested in finding the optimal placement of multiple web proxies (m) among the potential sites (n) under a given traffic pattern. We model the problem as a *Dynamic Programming* problem, and we obtain an optimal solution for a linear array topology using $O(n^2m)$ time.

Keywords

Web caching, Proxy server, Dynamic Programming

1 INTRODUCTION

We have witnessed an explosive growth in the use of World Wide Web (or web) in the past few years; there are many reasons behind this success, in particular, ease of use, the availability of standard tools for creating web documents and

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35388-3_42

H. R. van As (ed.), *High Performance Networking*
© IFIP International Federation for Information Processing 1998

for navigating the web, timely dissemination of information, and the increased popularity of the Internet [1]. At the same time, this quick adoption also leads to its poor performance, as web clients often have to tolerate long response times. There are a number of factors contributing to this inefficiency including server congestion during peak time, links with limited inadequate bandwidth, and long propagation delay. Caching has been considered as the prime vehicle to cope with this inefficiency.

The caching technique has been successfully used in the memory hierarchy [12] and distributed file system, AFS being one of such examples [10]. The basic principle behind caching is that it allows the retrieved documents to be kept close to the clients, this is essential in bringing down the access latency. There are several ways that documents can be cached for a web server including, web browser (client), web server itself and web proxy [15]. Caching at the clients' side has been implemented by most existing web browsers [2]. This can prevent a client from generating traffic to the same location repeatedly; for example both NCSA Mosaic and Netscape can save images and documents. Caching can also be deployed at the server side when a server contains pointers to other servers [15], this allows a web server to use a local copy fetching in advance to serve clients' requests, instead of having to forward the requests to remote server(s) each time. Unfortunately, both do little towards reducing the overall latency on the network [13]. Client side caching only saves one single client from fetching this document. In other words, each client has to cache the document, even if multiple clients accessing the same remote web document belong to the same local area network. Server caching only mitigates the problem of not forwarding requests further, but does nothing to alleviate the long access delay to the sever experienced by clients.

The most effective way in reducing the overall latency is the use of web proxy, or proxy server (or simply proxy). Web proxy is an intermediate server acting as an caching agent between clients and server. If properly designed, proxy can eliminate the possibly long propagation delay, and alleviate the potential inadequate link bandwidth path(s). Additionally, it also can reduce the server load, which may be critical during peak time. There has been considerable work on various aspect of web proxy, for example, traffic characterization [1], cache replacement algorithms [13] and server design [4, 9].

The effectiveness of proxy is primarily determined by ***locality***, the same as for any cache. This locality depends a number of factors such as access patterns and configurations. The unique characteristics of web caching, ***different*** from conventional caching used in memory and distributed systems, is that locality is also largely influenced by the location of the web proxies. Simply put, placing a web proxy in the "wrong" place is not only costly, but also does little to improve the performance. In addition, it has also been shown that multiple web proxies are sometime needed in order to increase this locality, e.g., the hierarchical caching proposed in [4, 6].

Finding the optimal placement of web proxies in a network like the Internet

is a challenging task, as there is relatively little data on how well web proxy works. The decentralized and dynamic nature of the web adds extra complexity to this task [15]. Most existing proxies are placed in fairly "obvious" spots, e.g., gateway for a LAN, or some "strategic" locations [11]. To the best of our knowledge, there has been no systematical study on the proper placement of web proxies, which is the aim of this paper.

In this paper, we focus on two factors: the overall traffic and latency as described in [15]. The objective is to minimize the overall latency of searching the target web server subject to the network resources and traffic pattern. Specifically, we are interested in finding the optimal placement of multiple web proxies (m) among the potential sites (n) under a given traffic pattern. This turns out to be a very difficult problem, mainly caused by the dependency among the potential sites. This is because a potential site, say i, can be in place between another potential site (j) and the web server. We define site i to be upstream of site j and j to be downstream of site i. The caching at any downstream site (j) in general modifies the traffic pattern of the upstream site (i). Unless the paths from all sites to the server are ***disjoint***, in which the finding the optimal location becomes trivial, these dependencies significantly complicate the problem. In this paper, we consider a simple linear array topology. We show that this can be modeled as a ***dynamic programming*** problem, we further obtain the optimal solution for the linear array topology using $O(n^2m)$ time.

The rest of the paper is organized as follows. We present the problem formulation in the next Section. Results are discussed in Section 3. We conclude the paper in Section 4 with discussions of on-going work.

2 PROBLEM FORMULATION

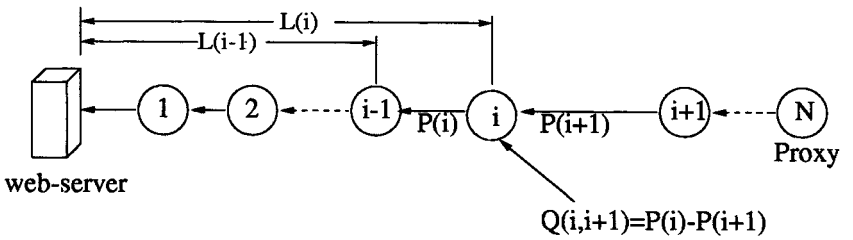

Figure 1 The Linear Configuration

We consider a one-dimensional array illustrated in Figure 1. Denote the potential web proxy locations by $n = 1, 2, \ldots, N$. Without loss of generality, we assume the locations starting from the web server to be labeled as $1, 2, \ldots, N$, i.e., i and $i+1$ are neighbors and i is closer to the web server than $i+1$. Let

$P(i)$ be the percentage of the overall traffic accessing the web that has to pass through node i. Since requests passing through node i must also pass through node $i-1$ we have $P(1) \geq P(2) \geq \ \ldots \ \geq P(N)$. Let the propagation delay (distance) from node i to the server be $L(i)$; If caching is done at the node i, we define the Gain to be $G(i) = L(i) \times P(i)$ *. This makes intuitive sense in that the percentage of the traffic ($P(i)$) would not need to traverse the distance from the node i to the web server, i.e., $L(i)$. We are interested in finding M ($M \leq N$) web locations $K_1, K_2, \ldots, K_M$, i.e., $K_1 < K_2 < \cdots < K_M$, that maximize the following value

$$(P(K_1) - P(K_2))L(K_1) + (P(K_2) - P(K_3))L(K_2) + \cdots \\ +(P(K_{M-1}) - P(K_M))L(K_{M-1}) + P(K_M)L(K_M)$$

The main complication is dependency between the potential web proxies, specifically the caching at a node i will affect the up-stream node, $1, 2, \cdots, i-1$. To simplify the notation, It will help us to slightly rewrite this problem. Set $P(N+1) = 0$ and now, for $1 \leq i \leq j \leq N+1$ define

$$Q(i,j) = P(i) - P(j) \quad i < j$$

Note that $P_i = P_i - P_{N+1} = Q(i, N+1)$. $Q(i,j)$ is the amount of traffic coming to node i when node j has been chosen as one of the proxies and no nodes between i and j (i.e., $i+1, \cdots, j-1$) are proxies; $Q(i, i+1)$ is simply the traffic arriving at node i that did not pass through node j as illustrated in Figure 1b. Using this notation the expression that we wish to maximize becomes

$$\begin{aligned} L(K_1)Q(K_1,K_2) \ &+ \ L(K_2)Q(K_2,K_3) + \cdots \\ &+ \ L(K_{M-1})Q(K_{M-1},K_M) + L(K_M)Q(K_M,N+1) \end{aligned}$$

To efficiently solve this optimization problem we will first generalize it

Definition Let n, m be such that $2 \leq n \leq N+1$ and $n-1 \leq m \leq M$. Let $K_1, K_2, \ldots, K_m$ be such that $1 \leq K_1 < K_2 < \cdots < K_m < n$. Set

$$\begin{aligned} cost(m, n: K_1, \ldots, K_m) \ &= \ L(K_1)Q(K_1,K_2) + L(K_2)Q(K_2,K_3) + \cdots \\ &+ \ L(K_{m-1})Q(K_{m-1},K_m) + L(K_m)Q(K_m,n) \end{aligned}$$

The (m,n)-*optimization problem* is to find $K_1 < K_2 < \cdots < K_m$ that maximizes $cost(m, n: K_1, K_2, \ldots, K_m)$.

Note: the reason for restricting $n-1 \leq m \leq M$ is that in the (m,n) problem

*The calculation derived in this paper also applies to other cost functions.

we must have $1 \leq K_1 < K_2 < K_3 < \cdots < K_m < n$. If $m < n-1$ this is obviously impossible.

The original problem becomes the problem of maximizing $cost(M, N+1 : K_1, K_2, \ldots, K_M)$, i.e., solving the $(M, N+1)$ optimization problem. We will now develop a dynamic programming method that permits solving all of the (m,n)-optimization problems with $2 \leq n \leq N+1$ and $n-1 \leq m \leq M$. Solution of the $(M, N+1)$ problem yields the solution to the original problem.

Our main observation is that solutions to the (m,n) problem must contain optimal solutions to certain subproblems.

Lemma 1 *Let $2 \leq n \leq N+1$ and $n-1 < m \leq M$. Further suppose that $m > 1$. $K_1, K_2, \ldots, K_m$ is an optimal solution to the (m,n) problem then $K_1, K_2, \ldots, K_{m-1}$ is an optimal solution to the $(m-1, K_m)$ problem.*

Proof: Suppose, by contradiction that the lemma is incorrect. Then there exist m, n, $K_1, K_2, \ldots, K_m$ and $K'_1, K'_2, \ldots, K'_{m-1}$ such that $K_1, K_2, \ldots, K_m$ solves the (m,n) optimization problem but $K_1, K_2, \ldots, K_{m-1}$ does not solve the $(m-1, K_m)$ one because

$$cost(m-1, K_m : K'_1, K'_2, \ldots, K'_{m-1}) < cost(m-1, K_m : K_1, K_2, \ldots, K_{m-1}).$$

But then

$$\begin{aligned}
& cost(m, n : K'_1, \ldots, K'_{m-1}, K_m) \\
= \; & L(K'_1)Q(K'_1, K'_2) + L(K'_2)Q(K'_2, K'_3) + \cdots \\
& \quad + L(K'_{m-1})Q(K'_{m-1}, K_m) + L(K_m)Q(K_m, n) \\
= \; & cost(m-1, K_m : K'_1, \ldots, K'_{m-1}) + L(K_m)Q(K_m, n) \\
< \; & cost(m-1, K_m : K_1, \ldots, K_{m-1}) + L(K_m)Q(K_m, n) \\
= \; & L(K_1)Q(K_1, K_2) + L(K_2)Q(K_2, K_3) + \cdots \\
& \quad + L(K_{m-1})Q(K_{m-1}, K_m) + L(K_m)Q(K_m, n) \\
= \; & cost(m, n : K_1, \ldots, K_{m-1}, K_m)
\end{aligned}$$

contradicting the optimality of $K_1, K_2, \ldots, K_m$ for the (m,n) problem.

This leads immediately to the following corollary:

Corollary 2 *Let $K_1, K_2, \ldots, K_m$ be an optimal solution to the (m,n) problem and $K'_1, K'_2, \ldots, K'_{m-1}$ be an optimal solution to the $(m-1, K_m)$ problem. Then $K'_1, K'_2, \ldots, K'_{m-1}, K_m$ is also an optimal solution to the (m,n) problem.*

Proof: From the Lemma we already know that $K_1, K_2, \ldots, K_{m-1}$ is also an optimal solution to the $(m-1, K_m)$ problem and thus

$$cost(m-1, K_m : K_1, \ldots, K_{m-1}) = cost(m-1, K_m : K'_1, \ldots, K'_{m-1}).$$

Therefore

$$\begin{aligned} & cost(m, n : K_1, \ldots, K_m) \\ = \; & cost(m, K_m : K_1, \ldots, K_{m-1}) + L(K_m)Q(K_m, n) \\ = \; & cost(m, K_m : K'_1, \ldots, K'_{m-1}) + L(K_m)Q(K_m, n) \\ = \; & cost(m, n : K'_1, \ldots, K'_{m-1}, K_m) \end{aligned}$$

Since $K_1, \ldots, K_{m-1}, K_m$ is optimal for (m, n) and $K'_1, \ldots, K'_{m-1}, K_m$ has the same cost as
$K_1, \ldots, K_{m-1}, K_m$ this implies that $K'_1, \ldots, K'_{m-1}, K_m$ is optimal as well.

Now define the following

Definition Let $2 \leq n \leq N + 1$ and $n - 1 \leq m \leq M$. Then

$$OPT(m, n) = \text{ maximal value solution for the } (m, n) \text{ problem.}$$

If $m = 1$ then, by definition,

$$OPT(1, n) = \max_{1 \leq k < n} L(k)Q(k, n). \tag{1}$$

If $m > 1$ and $K_1, K_2, \ldots, K_m$ is a solution to the (m, n) problem then, from Lemma 1, $K_1, K_2, \ldots, K_{m-1}$ is a solution to the $(m - 1, K_m)$ subproblem so

$$\begin{aligned} OPT(m, n) &= cost(m, n : K_1, \ldots, K_{m-1}, K_m) \\ &= cost(m - 1, K_m : K_1, \ldots, K_{m-1}) + L(K_m)Q(K_m, n) \\ &= OPT(m - 1, K_m) + L(K_m)Q(K_m, n) \end{aligned}$$

Thus, for $m > 1$,

$$OPT(m, n) = \max_{1 \leq k < n} [OPT(m - 1, k) + L(k)Q(k, n)]. \tag{2}$$

Equations (1) and (2) can together be used to calculate all of the $OPT(m, n)$ values. To also calculate the actual solution locations we define another array $K(m, n)$ that satisfies $K(1, n) = k$ such that

$$k < n \text{ and } L(K)Q(k, n) = OPT(1, n)$$

i.e., $K(1, n)$ is a solution location for the one-cache problem and $K(m, n) = k$ such that

$$k < n \text{ and } OPT(m - 1, K_m) + L(K_m)Q(K_m, n) = OPT(m, n)$$

i.e., $K(m, n)$ is a rightmost cache location in some solution to the (m, n)

problem. Notice that if there are many solutions, $K(m,n)$ might not necessarily be uniquely defined. Given this $K(m,n)$ array we can calculate a solution by setting $K_m = K(M, N+1)$ and, iteratively, for all $i < m$, setting $K_i = K(i, K_{i+1})$. Repeated applications of Corollary 2 shows that

$$cost(m,n : K_1, \ldots, K_m) = OPT(M, N+1)$$

and is thus the solution we are looking for.

Pseudocode for constructing the $OPT()$ and $K()$ arrays is given below. The first, initialization, section, has two $O(n)$ size nested for loops and therefore uses $O(n^2)$ time in total. The second section contains an outer $O(m)$ size for loop each iteration of which calls two nested $O(n)$ size loops. Thus the entire section, and therefore the entire algorithm, runs in $O(n^2 m)$ time.

1. **Initialization.**
   for $n := 2$ **to** $N+1$ **do**
   $k := 1$;
   for $j := 2$ **to** $n-1$ **do**
   if $L(j)Q(j,n) > L(k)Q(k,n)$ **then** $k := j$
   $OPT(1,n) := L(k)Q(k,n)$; $K(1,n) := k$;

2. **Filling in the array**
   for $m := 2$ **to** M **do**
   for $n := m+1$ **to** $N+1$ **do**
   $k = m$;
   for $j := m+1$ **to** $n-1$ **do**
   if $(OPT(m-1,j)+L(j)Q(j,n)) > (OPT(m-1,k)+L(k)Q(k,n))$
   then $k := j$
   $OPT(m,n) := OPT(m-1,k) + L(k)Q(k,n)$; $K(m,n) := k$;

3 RESULTS

In this section, we present an example to illustrate how the algorithm works. The example considers a configuration with $N = 10$ and $M = 5$, shown in Figure 3. The delay time $L(i)$ and traffic probabilities $P(i)$ are given in Table 1. This example shows a case with balanced spread-out traffic, specifically, each of the 10 nodes contributes 10% traffic (i.e., $Q(i, i+1)$). The distance is what dictates the measurement $P(i)L(i)$.

The algorithm essentially needs to fill in the matrix $K(M, N+1)$, whose entry $K(m,n)$ ($m = 1, 2, \cdots, M$, $n = 2, 3, \cdots, N+1$) denotes the rightmost (farthest) proxy location, i.e., $K_m = K(m,n)$, as defined earlier. Recall from Lemma 1, that the next element K_{m-1} must be the rightmost optimal solution

	1	2	3	4	5	6	7	8	9	10
$L(i)$	0.1	0.6	1.6	3.1	5.1	7.6	10.6	14.1	18.1	22.6
$P(i)$	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

Table 1 Example: The delay time $L(i)$ and traffic percentage $P(i)$

	1	2	3	4	5
10	(7)	(6, 9)	(5, 7, 9)	(4, 6, 8,10)	(4, 6, 8, 9,10)

Table 2 Example: The optimal solution (M, N)

for $K(m-1, K_m)$, i.e., the next element is $K_{m-1} = K(m-1, K_m)$. Accordingly, we can obtain all the optimal proxy locations : $K_1 = K(1, K_2)$, $K_2 = K(2, K_3)$, $\cdots$, $K_{M-1} = K(M-1, K_M), K_M = K(M, N+1)$.

The algorithm is divided into two parts: the first part is ***initialization***, which calculates the $K(1, N+1)$. For example, in the above example, $K(1, 11) = 7$, simply because $L(7)P(7) = 4.24$ is the maximum over all $L(i)P(i)$ $i = 1, 2, \cdots, 10$.

The next step is to fill in the rest of the elements in $K(m, n)$. For the above example, referring to the Figure 2. That $K(5, 11) = 10$ means that $K_5 = 10$ is the rightmost optimal proxy location. The next one is therefore $K(5-1, 10) = K(4, 10) = 9$, and so on. We have the optimal solution is $(4, 6, 8, 9, 10)$ shown in Table 2 and Figure 3.

Notice, as defined in Section 2, that the notation $K(m, n)$ in Figure 2 assumes the last element (n) has no traffic. For the rightmost proxy location (K_M), this is obtained by calculating $K(M, N+1)$. For others $K_i = K(i, K_{i+1})$, since the K_{i+1} has been chosen as one of the proxy locations, Hence the traffic generated from location K_{i+1} to location K_i is also zero. This is precisely what the term $Q(i, j)$ does.

4 CONCLUSION

In this paper, we investigate the optimal placement policy of web proxies for a target web server in the Internet. The objective is to minimize the overall latency of searching the target web server subject to the network resources and traffic pattern. Our contributions are 1) formulating the problem into a *dynamic programming* problem; 2) obtaining an optimal solution for *a linear array topology* using polynomial time.

N \ M	1	2	3	4	5
1					
2	1				
3	2	2			
4	3	3	3		
5	3	4	4	4	
6	4	5	5	5	5
7	5	6	6	6	6
8	5	6	7	7	7
9	6	7	8	8	8
10	7	8	8	9	9
11	7	9	9	10	10

Figure 2 Example: The optimal solution matrix I $K(M, N+1)$

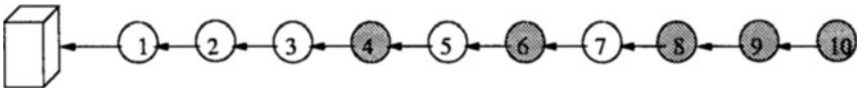

Figure 3 Example: The optimal proxy locations

The model proposed here can be easily extended to handle the following two cases:

- Hierarchical caching, in which the down-stream proxies only hold a subnet of the documents of the up-stream proxies. In such cases each down-stream proxy can only block a portion of the traffic. This can be handled by re-defining the notation $P(i)$ to be only the percentage of the traffic that the i^{th} node's cache can serve.
- Different link bandwidth. This can be dealt with by incorporating the link bandwidth into the distance $L(i)$, e.g., assigning larger value to slower links.

We are currently working on refining the model by relaxing two key assumptions in this paper, ***linear topology*** and ***static traffic pattern***. In particular, we are studying a ***tree topology***, which is considered to be more realistic topology for the Internet. The web server is the root of the tree. The complication, similar to the linear topology, is dependency among the potential web proxies. The result we obtained recently demonstrates that this can also be modeled as a dynamic programming problem with higher complexity $O(n^3m^2)$ [8]. In

addition, we are also working on reducing this complexity, and the preliminary result indicates that this can be brought down to $O(n^2m^2)$ [5].

The second issue concerns the dynamic nature of the traffic. The placement policy ideally should be distributed and adaptive. We are investigating this issue in the content of *active networking* [14]. The model assumes the potential web proxy sites can somehow monitor the traffic periodically, which is one of properties within an active network environment, and then make the decision about whether caching or not based on a threshold. Specifically, if the observed traffic volume within an observed period is above the threshold, the potential site will cache the web content. Notice that the threshold is determined by a number of factors, in particular the distance $L(i)$. In other words, different sites have different thresholds. This makes intuitive sense in that the closer the potential site is to the target web server, the higher the threshold should be. The results show that such distributed decisions can potentially lead to convergence to the static solution proposed in this paper by properly selecting the threshold [7].

The future work will consider more realistic web traffic distribution, for example capturing the actual workload characterization [1], or considering the Zipf distribution used by Bestarov [3].

REFERENCES

[1] M. F. Arlitt and C. L. Williamson, "Internet Web Servers: Wordload Characterization and Performance Implications," *IEEE Transactions on Networking*, Vol. 5, No. 5, October 1997.

[2] M. Baentsch, L. Baum, G. Molters. S. Rothkugel and P. Sturm, "World Wide Web Caching: The Application-Level View of the Internet," *IEEE Communications Magazine*, Vol. 35, No. 6, June 1997.

[3] A. Bestavros, "WWW Traffic Reduction and Load Balancing Through Server-based Caching," *IEEE Concurrency*, Vol. No. , January 1997.

[4] S. Glassman, "A Caching Relay for World Wide Web," *Computer Networks and ISDN Systems*, Vol. 27, No. 2, November 1994.

[5] M. Golin, G. Italiano and A. Vigneron, "The p-median problem on directed trees," To be submitted for publication.

[6] C. Bowman, P. Danzig, D. Hardy, U. Manber and M. Schwartz, "The Harvest Information Discovery and Access System," *Computer Networks and ISDN Systems*, Vol. 28, No. 1-2, December 1997.

[7] B. Li, X Deng, M. J. Golin and K. Sohraby, "Dynamic and Distributed Web Caching in Active Networks," Submitted to *APWeb'98*, April 1998. Also Li's presentation at Bell Lab, Lucent Technology, June 1997.

[8] B. Li, M. J. Golin, G. Italiano, X. Deng and K. Sohraby, "On The Optimal Placement of Web Proxies in the Internet," Submit to *IEEE Transactions on Knowledge and Data Engineering: Special Issue on Web Technologies*, May 1998.

[9] A. Luotonen and K. Altis, "World Wide Web Proxies," *Computer Networks and ISDN Systems*, Vol. 27, No. 2, November 1994.

[10] J. Morris, "Andrew: A Distributed Personal Computing Environment," *Communications of ACM*, Vol. 29, No. 3, March 1986.

[11] M. Nabeshima, "The Japan Cache Project: An Experiment on Domain Cache," *Computer Networks and ISDN Systems*, Vol. 29, No. 8-13, September 1997.

[12] D. Patterson and J. Hennessy, Computer Organization and Design: the Hardware/Software Interface, 2nd edition, *Morgan Kaufman*, 1997.

[13] P. Scheuermann, J. Shim and R. Vingralek, "A Case for Delay-Conscious Caching of Web Documents", *Computer Networks and ISDN Systems*, Vol. 29, No. 8-13, September 1997.

[14] D. Tennenhouse, J. Smith, W. Sinncoskie, D. Wetheral and G. Minden, "A Survey of Active Network Research," *IEEE Communications Magazine*, Vol. 35, , No. 1, January 1997.

[15] N. Yeager and R. McGrath, Web Server Technology, *Morgan Kaufman*, 1996.