

An Efficient Scheme to Remove Crawler Traffic from the Internet

X. Yuan, M. H. MacGregor, J. Harms

Department of Computing Science

University of Alberta

Edmonton, Alberta, Canada

Email: {xiaoqin,macg,harms}@cs.ualberta.ca

Abstract—

We estimate that approximately 40% of current Internet traffic is due to Web crawlers retrieving pages for indexing. We address this problem by introducing an efficient indexing system based on active networks. Our approach employs strategically placed active routers that constantly monitor passing Internet traffic, analyze it, and then transmit the index data to a dedicated back-end repository. Our simulations have shown that active indexing is up to 30% more efficient than the current crawler-based techniques.

I. INTRODUCTION

Search engines such as *Yahoo!*, *Google*[1], or *Altavista* gather pages from all over the world with the aid of crawlers. Generally speaking, it takes the crawlers of a specific search engine anywhere from two weeks to one month to update its database of Web pages. However, studies show that a large percentage of pages remain unchanged for more than four months[2], which implies that an unchanged page could be fetched several times by crawlers of the same search engine between changes. Currently there are more than one thousand search engines in the world. Therefore, it is possible for a page to be retrieved a few thousand times by crawlers within a short time period.

In this paper, we propose the use of an active network to make crawlers unnecessary. There are two generic types of active networks. In the first, most commonly known, all packets are tagged with code to be executed at each node passed. The second type of active network, known more specifically as an *active switch network*, is more appropriate for our proposal. In these networks, only some packets contain code, and only some nodes accept the passing code for execution. In an active switch network, the routers or switches of the network can perform customized computations on the passive data flowing through them, whereas in traditional networks user programs execute only at the end systems and the network passively transports user data between these end systems[3]. In the following, the term “active network” should be understood to stand for the more specific “active switch network”.

Our proposal is to have active nodes in the network index passing Web pages. First, we would send the code for a packet monitor and indexer into the active network. These modules would be accepted by routers willing to participate in the indexing service. This mechanism allows for a graceful, gradual

upgrade of network capabilities in the highly complex world of the current Internet. It also allows for a variety of service scenarios, without having to code this into the monitor. For example, the indexing service described below could be distributed over the Internet just to corporate gateways that have licensed it, and hold a proper cryptographic key. Alternatively, an ISP might nominate certain of its gateways to initially test the service, and then later allow the application to lodge in all its gateways. Another alternative would be for a collocation provider to host the application in multiple sites.

We believe that an active network approach is necessary to make the large scale deployment of active indexing feasible. It is simply unrealistic to assume that all routers everywhere will suddenly be capable (and willing) of performing the computations described here. Rather, there will be early adopters eager to try something new, as well as very conservative network administrators who will enable active computation on their routers only when there is a demonstrated economic benefit. Certainly a one-time installation of code on a small set of routers does not require an active network, but the gradual deployment and upgrading of the application over a large portion of a continually changing Internet does.

This paper begins with the discussion of several problems caused by the current implementation of crawlers. After presenting a brief case study of the load imposed by crawlers on servers, we explain the architecture of our indexing system. The paper closes with our simulation results and conclusions.

II. ISSUES WITH CRAWLERS

A crawler typically starts with a base list of popular sites known as *seed URLs* and follows links within the seed pages[4]. However, not all Web pages are equally good choices as seed URLs. Broder, et al [5] show that one can represent the Web as a graph with five main components: a central strongly connected core(*SCC*), a component *OUT* consisting of a set of pages that can be reached from the core but do not connect back to it, a component *IN* composed of a set of pages that can reach the core but cannot be reached from it, the *TENDRILS* that contain pages that cannot reach the core and cannot be reached from the core and finally, other disconnected components.

A crawler that starts off with pages in *SCC* could reach all pages in *SCC* and *OUT*. However, it would never find any of the

pages in *TENDRILS*, *IN* or *DISCONNECTED*. Pages in these regions of the Web are effectively "hidden" from crawlers.

In contrast, active indexing will see all pages fetched by clients, no matter what the characteristics are of their degree or direction of connection to other pages in the Web.

In addition, current crawlers cannot generate queries or fill out forms, so they cannot visit dynamically generated pages. This problem will get worse over time, as more and more sites generate their Web pages dynamically from databases. It is an open question as to how dynamic content should be indexed, and we do not examine the issue further here. We note that active indexing provides a mechanism for indexing such pages, though, whereas crawlers cannot. The monitor in the active routers will capture all passing HTTP traffic.

Thus, the indexes generated by active indexing will be more complete than those that can be generated via crawlers. Web pages in tendrils and disconnected components, as well as dynamic content, will be indexed by the active network.

In addition, active indexing is capable of providing a "dynamic" view of the Web, by collecting statistics such as the number of times per hour a particular page has been fetched. Crawlers cannot create such a view of the Web. These statistics could be used during index generation in combination with other common indexing factors. In contrast, crawling the Web gives us only a static picture of its contents. Statistics such as the number of times a particular page is linked to by other pages can be used to weight the importance of a page, but that is only a proxy for how many times that particular page is likely to be fetched. Active indexing can deliver both static information about Web pages, such as the number of times a page is linked to by other pages, and also dynamic statistics including the rate at which a page is fetched.

III. A CASE STUDY OF CRAWLER LOAD

To study the load imposed by crawlers, several experiments were conducted using the log files for the week from June 29, 2001 to July 5, 2001, for the Web server in the Department of Computing Science at the University of Alberta.

Figure 1 illustrates the ratio of crawler hits to total hits, and the ratio of bytes fetched by crawlers to the total bytes fetched for the week. The average hit and byte percentages are 27.3% and 20.9% respectively. A maximum of 40.6% of all hits was due to crawlers. The percentage of bytes fetched by crawlers reaches a maximum of 29.5%.

The crawler byte percentage is always lower than the crawler hit percentage because crawlers typically fetch short pages. There are two reasons for this. First, crawlers only fetch standard HTML pages and ignore all other media and document types, such as PDFs, images, and sound files. These latter types are usually larger than standard HTML pages. Secondly, crawlers from some services use the HTTP HEAD command to get meta information about a page before actually fetching it. This meta information is only a small portion of the page.

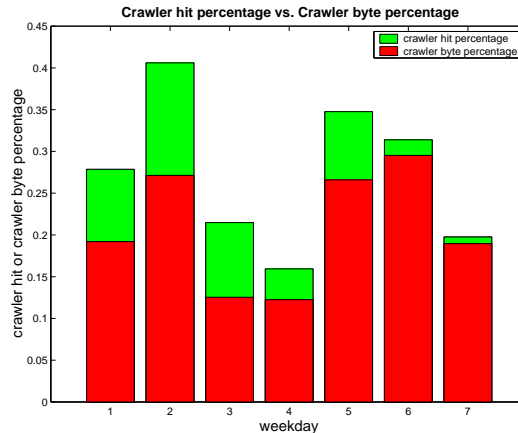


Fig. 1. Crawler hit and byte percentages

IV. ACTIVE INDEXING

The Internet can be viewed as a collection of subnetworks connected to a backbone. For each subnetwork, there is a gateway router connecting it to a router on the backbone. Usually there will be more than one intermediate router between a client and a Web server.

We propose to index Web pages on the gateway routers. Although the core routers have a more global view, additional processing load will be much more easily accommodated on the gateways. By using the gateways, there will still be fewer monitoring nodes than servers because there can be multiple Web servers on each stub network. This will help ensure that the application scales reasonably. Gateway routers, by definition, see all traffic between their network and the rest of the world, so all traffic will be indexed. Each gateway will calculate indexing data only for the Web servers in its adjacent stub network (i.e. for HTTP requests from clients outside its stub network).

Consistency of the final index is enforced by the back-end repository to which all the active routers return their results. The repository is responsible for ensuring that only one copy of the raw indexing data for a particular page is accepted and later processed by the algorithm responsible for producing the updated search index.

As for the nodes themselves, the structure and programming of active nodes and networks is currently being researched by several groups. The survey by Campbell [6] provides an overview of several projects. Decasper [7] documents the structure of a high-performance active node. Legacy routers that are not programmable could be accommodated by replicating their traffic, and handing the replicated streams off to a directly-connected active node for the purposes of indexing. The legacy router would remain responsible for routing the original traffic. Alternatively, because active indexing does not need to execute on every router, nor even on every gateway node, we could decide to execute active indexing only on routers that can accommodate it.

Indexing proceeds in two stages. In the first stage, individual

IP packets of each communicating pair are processed to reconstruct HTTP messages. The reconstruction of HTTP messages involves handling out-of-order, corrupt, and duplicate packets. First we demultiplex packets according to their source and destination IP address. Then we reorder packets according to TCP sequence numbers and eliminate duplicates[8]. We need to associate each response with its corresponding request since the URL is not present in the response headers. Since the HTTP response is the first data that the server sends back to the client and will acknowledge the last byte of the HTTP request, the first sequence number of the HTTP response should be equal to the acknowledgment number of the request. Consequently, the request and response can be matched using the sequence numbers and acknowledgment numbers. In the second stage, for all URLs produced in the first stage, URLs are hashed into page identifiers for the convenience of quick location of a URL. Then each page is parsed to remove HTML tagging, tokenize it into individual terms, and extract postings (term-location pairs) to build an inverted file. The inverted file is compressed using an efficient compression scheme, called mixed-list storage [9].

The resulting index is consistently about 7% the size of the input HTML text. This is so far the most efficient compression scheme reported. Thus, if the indexer builds indexing data from all the input it receives, and the generated inverted index is compressed by using the mixed-list scheme, the final compressed data which will be sent to the backend repository is roughly 7% the size of the original HTML text.

Further compression is also possible, if we exploit the meta-data in Web pages. A page just indexed is probably still the same the next time it passes through a router. Studies in [2] show that the average change interval of a Web page is about 4 months. In order to be efficient, we should only index changed and new pages.

To implement this, in each router a hash table is used to associate a URL with a page identifier. Identifiers of all the pages seen so far by the router are saved in the hash table. For each identifier, we store the value of the *Last-Modified* field. If a document was previously seen by the router and the *Last-Modified* field in the more recently fetched page is newer than the stored value, the page is reindexed and the *Last-Modified* in the hash table is updated. If the two *Last-Modified* values match, the page is not reindexed. If the page has never been seen by the router, a new entry is created in the hash table. Besides the *Last-Modified* field, other fields including *Expires* and *ETag* can also be used to compare an old page to a possibly newer one. By indexing only fresh pages, the final index sent to the repository will decrease in size significantly.

In our indexing system, we gather two important global statistical metrics: inverse document frequency for indexing terms and hit frequency for documents.

Given a specific term or word, the *inverse document frequency* or *idf* is the reciprocal of the number of documents in which the term appears. The motivation for using the *idf* is that terms which appear in many documents are not very use-

ful for distinguishing a relevant document from a non-relevant one. The *idf* is used in ranking the query results, and helps return documents which tend to be relevant to a user's query.

Hit Frequency (hf) is the number of times per unit time a page has been seen by the monitoring process on a router. It gives us a dynamic view of the Web. We gather this data on the active routers. In the hash table associating a URL with a page identifier, besides the *last-modified* field, an *hf* field is also added. Whenever a document is seen by the indexer, whether it is fresh or stale, the *hf* counter for the document is incremented. If two documents contain a query keyword, the one with the larger *hf* is ranked higher. It also can be combined with the *idf* in answering a search query.

V. SIMULATION

For this study, we chose a transit-stub model generated using the Georgia Tech Internet Topology Generator(GT ITM) [10] to create a network representative of the structure of the Internet. The transit-stub model is composed of interconnected transit and stub domains. A transit domain comprises a set of highly connected backbone nodes. A backbone node is either connected to several stub domains or other transit domains. A stub domain usually has one or more router nodes, which have links to transit domains. Our specific model had 4 core routers, 12 gateway routers (12 stubs), 20 sites for Web servers and 60 sites for Web clients. In the traditional case, we simulated four sites issuing crawler requests. In the active indexing case, three of those sites became routers and the fourth was used as the repository site.

Our simulations were carried out using SMURPH[11]. Eight runs were used for each data point, and the resulting variances were used to calculate 95% confidence limits using the t-statistic. These confidence limits were found to be very tight, and are not actually visible on the resulting plots.

Previous studies have shown that request interarrival times follow an exponential distribution[12]. Thus, the request arrival process corresponds to a *Poisson process*, where users arrive independently of one another. We vary the mean client request interarrival time to impose different workloads on the network. HTTP request packets are approximately lognormal distributed in size with a mean of 2880 bits and standard deviation of 848 bits[13]. The response sizes can be represented by a lognormal distribution combined with a heavy-tailed distribution[14]. In our simulation, we represented the response sizes with a lognormal distribution having a mean of 80,000 bits and a standard deviation(SD) of 200,000 bits[13].

The indexing messages have a fixed length of 10,000 bits. The message inter-departure time is unique for each router in a specific simulation case. We send the indexing traffic to the backend repository constantly, so theoretically all routers should spend the same time to transmit the indexing data as was used to accumulate it. Although each router generates a different amount of indexing traffic within a given time period, we make sure the index data are sent in the same fixed time.

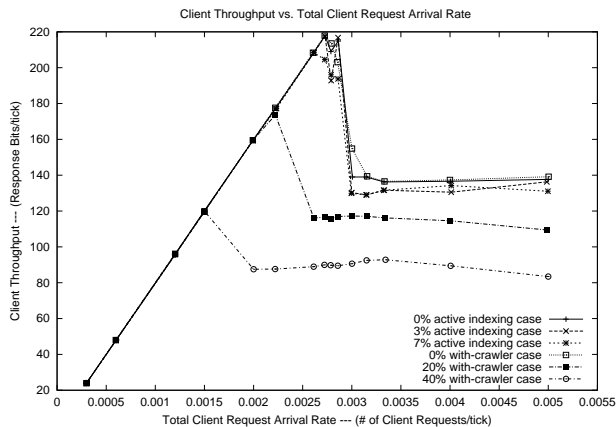


Fig. 2. Client Throughput in All Cases

We balance the index accumulation time and the index delivery time so that there is no extra index data left on routers. Each router sends the indexing data with a departure rate proportional to the volume of indexing data accumulated.

If the active routers index all passing HTTP traffic, then the compressed inverted file that is sent to the repository will be about 7% of the size of the original HTML documents. In fact, we only need to index new pages plus those pages which have been modified since they were last indexed. Therefore, the indexing data sent to the repository should be considerably less than 7% of the original HTTP messages. Thus, for the case of the active network, we simulated systems with 0% overhead, 3% overhead and 7% overhead.

By overhead, we mean the number of bytes generated by indexing as a fraction of the number of bytes in the passing Web pages. It should be stressed that the overhead values (0%/3%/7%) in the active indexing case represent the efficiency of indexing, and that the indexing traffic is concentrated on the routes to the repository rather than being more evenly distributed as is the case with the crawler traffic. Without studying active indexing on a representative, fairly complex network, it is not at all clear whether human users will see improved throughput and decreased delays.

VI. EXPERIMENTAL RESULTS

We simulated the active network with 0%, 3% and 7% indexing overhead respectively and the traditional network with 0%, 20% and 40% crawler traffic. Again, it must be stressed that the percentage of indexing overhead is not comparable to the percentage of overhead traffic. That is, one cannot conclude by simple subtraction that switching from a system with 40% crawler traffic to one with 7% indexing traffic will result in 33% less network load. The indexing load is concentrated onto just a few routes leading back to the repository, so the distribution of network load in the active indexing case will be quite different than in the with-crawler case.

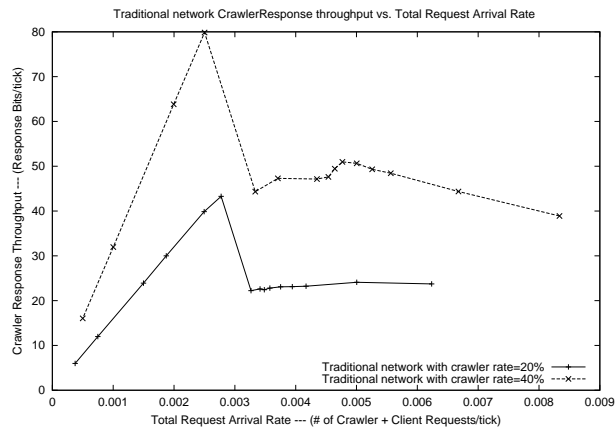


Fig. 3. Crawler Throughput in With-crawler Cases

Figure 2 illustrates the client throughput in both the traditional network and the active network. The vertical axis gives the *client throughput* - the number of bits received by normal clients per simulation time unit - while the horizontal axis gives the client request arrival rate. The throughput for 0% overhead active indexing matches that for the 0% with-crawler case. This helps establish the comparability of the remaining cases. Both simulations achieve the same peak throughput of about 222 bits/tick, after which the throughput drops rapidly as the systems become saturated. Thereafter, the throughput remains constant at about 140 bits/tick.

All three active indexing cases support approximately the same maximum throughput of 220 bits/tick. However, the 20% with-crawler case shows a 20% reduction in client throughput to 175 bits/tick and the 40% with-crawler case shows a 45% reduction to 120 bits/tick. The crawler load in these cases translates directly to a reduction in client throughput. The impact can be greater than 1:1 due to the effects of finite buffers, losses and retransmissions once the network becomes stressed.

Figure 3 shows the crawler throughput in the 20% and 40% with-crawler cases. The vertical axis gives the number of bits per simulation time unit received by crawlers, and the horizontal axis gives the total request arrival rate. This is composed of requests generated by both human clients and crawlers. The graph shows that crawler throughput in the 40% case is overall twice that in the 20% case. The greater-than-1:1 reduction in client throughput between these two cases shows that client traffic is being unfairly impacted by the crawler traffic (i.e. the crawler traffic is not suffering the reduction seen by the client traffic). This is likely due to the difference in message length between the two types of traffic, where the longer client messages are more likely to be delayed than the shorter crawler messages.

This is substantiated by the data presented in the next two graphs. Figure 4 gives the average client request delay for the with-crawler and active indexing cases. The vertical axis gives the average client response delay, and the horizontal axis gives

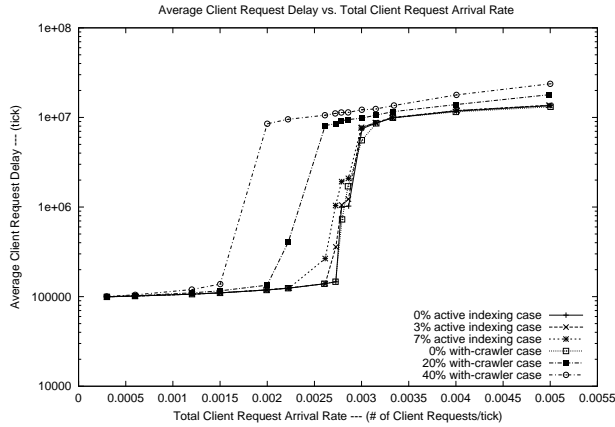


Fig. 4. Average Client Request Delay in All Cases

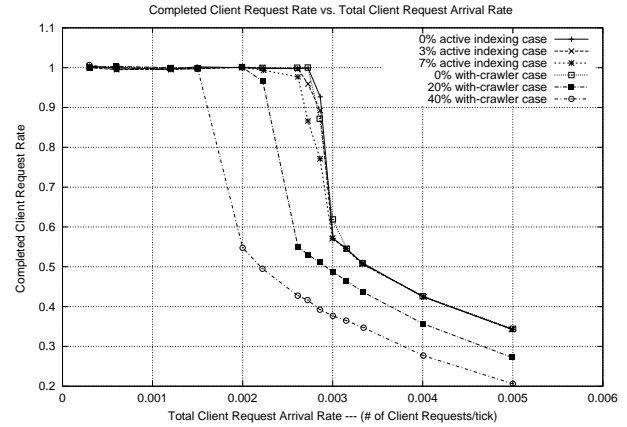


Fig. 6. Completed Client Request Rate in All Cases

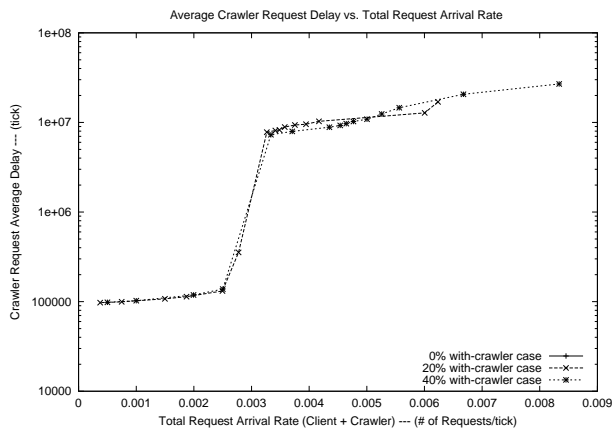


Fig. 5. Average Crawler Request Delay vs. Total Request Arrival Rate

the rate at which human clients generate requests. The traditional network with 20% or 40% crawler traffic has much longer average client delays than for the active network.

Figure 5 shows the average crawler request delay versus the total request arrival rate. The two curves are nearly identical, thus making it clear that increasing the relative amount of crawler load does not impact the delay seen by crawler sites. In contrast, the previous graph shows that increasing the relative amount of crawler load does impact the client delay.

Figure 6 gives the fraction of client requests that are completed. Almost all requests are satisfied when the request arrival rate is small. All three active indexing cases have nearly identical client completion rates. However, the 20% and 40% with-crawler cases show significant decreases in the rate at which client requests are completed. As the with-crawler networks become overloaded, the rate of request completion decreases dramatically. At a client request arrival rate of 0.0025, 40% crawler traffic is sufficient to decrease client rates by nearly 60%.

VII. CONCLUSIONS

In this paper, we propose an active indexing system for the World Wide Web. This system eliminates crawler traffic in favor of packet monitoring on strategic routers. This approach has significant advantages. Firstly, by removing crawler traffic, which we estimate is currently on the order of 40% of all network traffic, bandwidth is saved and made available to human requests. Our simulations show that client traffic is affected disproportionately by crawler traffic, both in terms of delay and probability of completion. Reducing or removing crawler traffic would have a significant effect on both fronts.

Furthermore, server resources such as CPU cycles and disk I/O are saved so that the servers can be dedicated to processing a variety of human requests more quickly. This leads to a further reduction of user perceived latency and improves the throughput of responses to human visitors. It also allows corporations to delay new investments in servers until the hit rate from human clients increases to replace the load previously seen from crawlers.

In addition, this system can index Web pages more exhaustively than crawlers. Crawlers follow links on the Web to find new pages. The chance of a page being fetched by a crawler is contingent on its link relation to other pages and the link structure of the Web. Therefore, some pages are never captured. In active indexing, the routers gather index data for each passing page. Therefore, each page on the Internet has an opportunity to be retrieved regardless of its link relation to other pages.

In addition to the statistical data currently gathered by search engines, active indexing can also derive dynamic data about a page, such as the rate at which it is visited. The dynamic data about a page can be used in conjunction with existing statistics to rank the query results.

Both active indexing and crawlers share the problem of pulling new pages into the index. For active indexing, either a client must somehow know the page exists, and fetch it, or the creator of the page could simply fetch it through the network after its creation. In the case of crawlers, the current practice

is to actively add a link from a portal site to the new page so it will be crawled.

Active indexing also opens up the possibility of indexing dynamic content, something that is not possible with crawlers. Although we do not suggest the appropriate mechanisms in this paper, we note that an increasing proportion of Web content is dynamic, and thus some means must be discovered to make it searchable.

The "activeness" of the network is crucial, not to the execution of the application, but to its deployment. It is unrealistic to assume that all gateway routers would necessarily be "willing" or capable of executing this application. Only those routers initially capable of executing it, and whose network administrators see benefit in doing so, would accept the passing code. Updates to the package and deployment further through the Internet would be enabled by the active mode of operation of the routers.

Active indexing is an interesting application, both because it illustrates the way in which new network protocols and applications can be deployed in an active network, and because it could have a very significant impact on the overall load on the Internet. The end-to-end paradigm is very powerful, and this is not a suggestion that it is not useful. Rather, this study shows that there is valuable information that can be derived in the interior of the network much more efficiently than on the endpoint systems.

REFERENCES

- [1] <http://www.google.com>.
- [2] J. Cho and H. Garcia-Molina. "The evolution of the web and implications for an incremental crawler". In Proc. of 26th Int. Conf. on Very Large Data Bases, pp. 117-128, September 2000.
- [3] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, G. J. Minden, *A Survey of Active Network Research*, IEEE Communications Magazine, Vol. 35, No. 1, pp. 80-86. January 1997.
- [4] S. Brin, L. Page, *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, WWW7/Computer Networks 30(1-7): 107-117(1998).
- [5] A. Broder, R. Kumar, F. Maghoul, et al, *Graph structure in the web*, WWW9, pp. 247-256, 2000.
- [6] A. T. Campbell, H. G. de Meer, M. E. Kounavis, et al, *A survey of programmable networks*, Computer Communication Review, Vol. 29, No. 2, Apr. 1999, pp. 7-23.
- [7] D. Decasper, B. Plattner, G. Parulkar, et al, *A scalable, high performance active network node*, IEEE Network, Vol. 13, No. 1, Jan., 1999, pp. 8-19.
- [8] A. Feldmann, *BLT: Bi-Layer Tracing of HTTP and TCP/IP*, WWW9 / Computer Networks 33(1-6): 321-335, 2000.
- [9] S. Melnik, S. Raghavan, B. Yang, et al, *Building a Distributed Full-Text Index for the Web*, WWW10, pp. 396 - 406, May 2-5, 2001, Hong Kong.
- [10] E. W. Zegura, K. L. Calvert, S. Bhattacharjee, *How to Model an Internet network*, INFOCOM'96. Fifteenth Annual Joint Conference of the IEEE Computing Societies. Networking Next Generation., Proceedings IEEE, Vol.2, 1996, pp. 594-602.
- [11] P. Gburzynski, *Protocol Design for Local and Metropolitan Area Networks*, Prentice Hall, 1996.
- [12] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva. *Web Traffic Modeling and Performance Comparison Between HTTP1.0 and HTTP1.1*. In System Performance Evaluation: Methodologies and Applications. August 1999, pp. 119-128.
- [13] H. Choi, J. Limb, *A Behavioral Model of Web Traffic*, Proceedings of the Seventh Annual International Conference on Network Protocols, 1999, pp. 327-334.
- [14] B. Krishnamurthy, J. Rexford, *Web Protocols and Practice, HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*, July 2001. ISBN 0-201-71088-9.