RC 22115 (W0107-014) 17 July 2001 Computer Science

IBM Research Report

Policy Based Management of Content Distribution Networks

Khalil S. Amiri, Seraphin B. Calo, Dinesh Verma
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

Policy based Management of Content Distribution Networks

Khalil Amiri, Seraphin B. Calo, **Dinesh Verma**
IBM T. J. Watson Research Center,
30 Saw Mill River Road
Hawthorne, NY 10532
{amirik,scalo,dverma}@us.ibm.com,

Abstract

We present a policy based architecture for the control and management of content distribution networks that form an overlay over underlying physical networks. The architecture extends the policy framework used for controlling network QoS and security to the case of content distribution networks. The fundamental advantage of a policy based framework is that it allows a machine-independent scheme for managing multiple devices from a single point of control. In this paper, we describe this architecture and demonstrate how it can enable dynamic updates to content distribution policies. Furthermore, we analyze the impact of such dynamic distribution on the cost of content serving.

Keywords:
Distributed Systems, Web Services, Application Acceleration, Content Distribution

1. Introduction

Ensuring a satisfactory performance level for web-based applications on the Internet is a long-standing problem within the networking research community. In order to address the performance issues, several approaches have been developed. The three most prominent approaches to managing the performance of applications include the IntServ/RSVP [1] signaling approach, the DiffServ approach [2] and the content distribution approach [3] [4]. The IntServ approach is based on the notion of reserving resources within the network, the DiffServ approach is based on the notion of mapping traffic aggregates into several different service classes, and the content distribution approach is based upon deploying a number of proxy servers scattered within the network that use a variety of caching techniques to improve the response time of web-based applications.

While each approach has its own advantages and disadvantages, they all share one common problem. The proper operation of each approach requires a consistent configuration of a large number of widely distributed devices. The policy based management paradigm which has recently gained prominence provides an approach that can be used to address this problem. Policy based techniques are being developed for the Integrated Services and the Differentiated Services technologies by various working groups within the IETF [5]. In this paper, we explore the methods by which the same technical approach can be applied to a content distribution network.

The rest of this paper is organized as follows: in Section 2, we provide a brief overview of policy based networking and its operation; and, in the next section, we present an overview of content distribution networks. In Section 4, we discuss the application of policy based techniques to content distribution networks, and determine the form that typical policy rules for such an environment would take. Then, in Section 5 we present a system architecture for a content distribution network that employs the techniques of policy based management. Finally, we identify our conclusions and provide a summary discussion.

2. Policy based Networking

A network policy is a statement defining which classes of traffic should be treated differently in the network, and in what ways. It is defined by an administrator and specifies the rules related to the handling of the different types of traffic. Policies provide a way to simplify the management of multiple devices deploying complex technologies.

The policy framework defined by the IETF consists of four basic elements as shown in Figure 1 The basic elements are: the policy manager, the policy repository, the policy enforcement point (PEP), and the policy decision point (PDP). The *policy management tool* is used by an administrator to input the different policies that are to be active in the network. The devices that can apply and execute the different policies are known as the *policy* enforcement points (PEPs). The preferred way for the management tool and policy targets to communicate is through a policy repository. The *policy repository* is used to store the policies generated by the management tool. Instead of communicating directly with the repository, a policy enforcement point can also use an intermediary known as the policy decision point. The *policy* decision point is responsible for interpreting the policies stored in the repository and communicating them to the policy enforcement point.

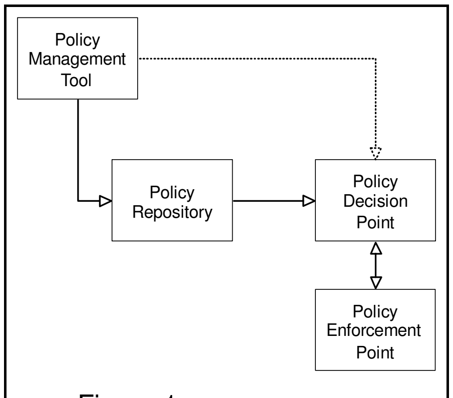


Figure 1.

The simplification in management is obtained primarily by centralizing the definition of policies in a single repository, and by defining abstractions that provide a machine-independent specification of policies. The centralization goal is obtained by defining the policies that apply to the entire networked system at a single location. This allows checking the consistency among the policies being defined at different devices in the network. The policy specification contained within the repository is defined in terms of the technology to which a policy would apply, rather than in terms of the configuration parameters of the device. Thus, a policy may specify which DiffServ Per-Hop Behavior (PHB) should be used for a specific application flow, rather than how this mapping would be configured for a specific device.

The specification of policy rules within the repository is key to obtaining inter-operability. The information model for the rules to be specified are being defined by the standards bodies [5]. While the intricacies of the information model are beyond the scope of this paper, we will follow the basic principle of defining a policy as an expression of the form "if condition then action". Here, the conditional part describes a specific situation that can be encountered by a network device, and the action component specifies the action that is to be performed by the device in that situation.

There are many ways to interpret and implement the rule-based format of the policies. In the most general sense, the action may be construed as a procedure that is defined and stored in the repository (e.g. a piece of Java code, or a program in a language like CacheL [6] or PONDER[7]). A less general interpretation is to consider the policies as instances of first order predicate expressions. An even more limited interpretation is to consider the policy conditions as key fields that are used to index a table of actions, where the corresponding actions are the set of simple operations that need to be performed. When a more general implementation of policies is followed, more complex policies can be specified, but they are also much harder to analyze for consistency. The standards bodies have tended to follow the limited tabular approach, leading to policies that are easier to analyze, and can be stored conveniently in an LDAP repository. The information model defined in the standards drafts is more complex than the simple tabular model we are presenting, but the two can be mapped onto each other. For the networking disciplines that have been the focus of policy based management, namely Integrated Services, Differentiated Services and IP-security, the tabular approach has been found to be adequate. The goal of this paper is to examine whether the same approach can be used in defining distribution policies for content replication.

3. Content Distribution Networks

Content Distribution Networks attempt to reduce user response time by caching content closer to the end user. The primary focus of this approach has been on web-based applications. A content distribution network consists of a network of caching proxy servers to which clients are directed transparently using various wide-area load balancing schemes. The caching approach works well for data that is static and unchanging, e.g. images, video clips, etc. There are many companies providing such a service for static content. The research community is also exploring approaches that would enable more of the content to become cacheable. This is achieved by deploying a variety of techniques such as query-result and database caching [8], high-volume publish-subscribe systems [9], change graphs [10], active middleware[11] etc.

A typical content distribution environment is shown in Figure 2. It contains machines belonging to three different administrative entities. A web-browser is used by the client to access a web-server on the Internet. The web-server owner has a contract with another organization (which we refer to as a CDSP or Content Distribution Service Provider) that has multiple sites hosted throughout the network. These sites are shown as the crosshatched boxes in Figure 2.

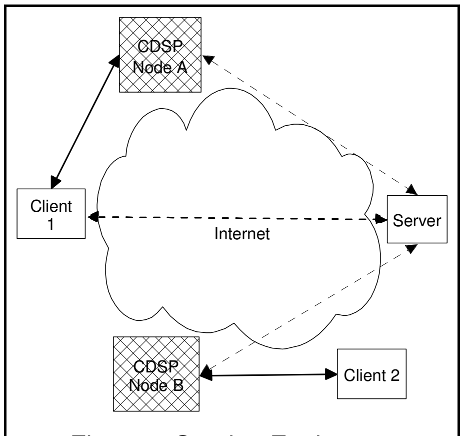


Figure 2 Service Environment

Normally, the client browser interacts with the main web-server (the dashed flows). The goal of the content distribution network is to replace the client-server communication by a set of communication flows, one occurring between the proxy and one of the CDSP nodes (the solid flow), and the other occurring between the CDSP node and the main server (dotted flow).

The assumption is that there are a large number of applications where the dashed flows can be decomposed into a set of solid and dotted flows such that the solid flows predominate, and the bandwidth/frequency of dotted flows are relatively small. In these circumstances, execution of web-based applications at a CDSP node would result in reduced response time to the end user, as well as a reduced load on the network and the main web-server.

In this model, the browser would end up communicating with a CDSP node that is close to it, and retrieve files or invoke cgi-bin programs/JSPs/servlets at that node. These program invocations may result in invocations to programs at the web-server, or access to databases at the web-server. However, the latter invocations would be less frequent and would require less bandwidth.

Note that different clients may access different CDSP servers. Figure 1 shows client 1 accessing the CDSP node A, while client 2 accesses the CDSP node B. We assume that there is a business agreement between the CDSP nodes and the web-server owner that allows for the execution of programs owned by the web-server at the CDSP nodes. This will allow offload of dynamic content as well as static content [12].

Within such an environment, one needs to determine the policies that would be appropriate for controlling the distribution of programs and files across the various nodes. Not all types of data and programs are suitable for execution within the proxy environment. The decision on what can be moved out for execution and caching at the different edge servers depends upon several factors, such as: the type of page being accessed, the URL being invoked, or the frequency of access to a site. The intent of this paper is to explore whether the tabular paradigm used in the policy framework is adequate for controlling the distribution of code and data in this environment.

4. Policy Rules for Content Distribution Networks

In order to examine whether the simple policy model can be applicable to content distribution networks , we look at the different components that will make up its condition and action parts. Specifically, we look at the individual fields that would constitute the condition and action parts of the policy statement. A policy in the context of a content distribution network would be a statement defining what type of programs/data can be moved out to a CDSP node and what type of programs/data must be executed at the origin server.

With this definition of policy, there are only two types of actions that are possible, namely a piece of program/data is either edgeable or not edgeable, where edgeable means that the program is capable of executing at a CDSP proxy or that a piece of data is cacheable at the CDSP proxy. However, any type of caching comes with assumptions regarding how the consistency of the cached data ought to be maintained. Thus, when an action directs that a piece of program/data be edgeable, the action must also specify the consistency model that must be followed by the cached data/program.

Various applications have different cache consistency requirements. However, the different cache consistency models needed to describe the types of requirements demanded by the various web applications are finite and can be easily enumerated. We will focus here on the consistency models for static cached data as an illustrative example. In this case, at least three different consistency models could be used. The first, *strong consistency*, requires that an access to the cached page return the most up-to-date copy of the page (the page last written to the origin server). This type of consistency can be guaranteed, for example, by validating a cached page against the origin server upon every access to the cache using the conditional-get request of the HTTP protocol. Another protocol that can be used is to have the origin server invalidate (or update) all copies at the caches before completing an update to the master copy of the page. That is, an update at the main site is pushed to all the edges before the update is considered complete. A second consistency model, *time-limited inconsistency*, requires that a cached page be updated within a specified maximum time window following an update to the main page at the origin site. This model can be implemented, for example, by refreshing cached data at frequent-enough regular time intervals. A third consistency model, *loose consistency*, requires that pages at the cache be updated as soon as the network can do so. This can be achieved by having the origin server batch updates or invalidations and broadcast them to the edge caches less frequently according to a best-efforts policy. Such a protocol may help conserve load and network bandwidth and may be sufficient for some applications.

In summary, we can distinguish three types of application consistency requirements for statically cached pages. For each consistency model, one or more possible consistency protocols can be employed to actualize that model. However, the number of interesting protocols and models is finite and therefore can be summarized in a concise table. The action field in the table can specify the particular protocol that ought to be used for the particular deployment.

Not all data may be considered cacheable at all locations. Our assumption is that the CDSP proxies may be located throughout the world, and some of them may be subject to regulations that prevent them from caching specific types of content. Therefore, a policy action must be able to restrict the caching function to be performed only at specific sites. The site restriction may be specified by either a subnet mask of the IP addresses that a proxy may have, or by a regular expression that ought to be matched by the domain name of the proxy server.

For the condition field, the standard 5-tuple that is ubiquitously present in the current schemes for QoS and IP-security would also form part of the condition field for content distribution. Each CDSP edge-server may only be caching content for a certain set of origin sites, so the source IP address will be part of the condition. The source port number is not required for web-based clients, but may be needed for some applications that will only use a specific range of port addresses. The destination IP address is needed to restrict the scope of distribution to a specific server, and the destination port number is needed for identifying the specific application at the server. The protocol field is also needed as part of the process for identifying the application.

This 5-tuple is not adequate for identifying the type of application that can be cached. Assuming a web-based application, the cacheability of the application/data can also be determined by the type of Universal Resource Identifier (URI) that the client is accessing. The URI may define whether or not a specific resource is edgeable. The URI is often used to map to a specific program that is executed, or a file that is retrieved. The class of programs, or the exact name of the program can determine whether or not a resource is edgeable. As an example, a website may decide to put all servlets that are edgeable under a subtree `/edgeable_code` under the root of the website, and the prefix of the program name invoked by a specific access would then determine whether or not the program was edgeable.

Other application level data can also be used to determine if parts of a user-interaction can be satisfied by a CDSP proxy. The user identity could determine whether the contents being accessed by the user were to be made edgeable. Since CDSP offload may be associated with a charge, a site may decide only to offload contents for its Gold-quality customers rather than for all its customers. In some application instances, the type of offload that is feasible would depend on the nature of the user. As an example, for LDAP directory query caching, it is straightforward to support offload for the user-id of anonymous or for users that only have read-only access, but relatively difficult to satisfy the offload for users that have write-access, or varying security and access control needs.

In many instances, the decision to allow offloading of applications or data to a CDSP site is determined by the current state of the network. Many origin sites may want to use the CDSP network only as a backup mechanism when the load on the site increases significantly. In these cases, a minimum threshold can be defined to determine the arrival rate beyond which offloading would be enabled. We assume that there is a separate process for monitoring the work load in the system outside of the policy framework. A single CDSP proxy server is usually unaware of the current load on the entire distributed system. Therefore, it would be appropriate for a CDSP to use a monitoring system to track the load on a specific site, map it onto two or more levels, and use these levels to determine whether the contents of the site are to be made edgeable or not. As an example, a performance monitor may use the information about a backend site's load to classify it as underloaded, moderately busy, or very busy. Application offload and caching would be enabled only when the site’s load was considered to be very busy. The status of the performance monitor would need to be part of the condition rule in the policy clause.

The condition and action components as described above can be used for a tabular representation of policies related to content distribution networks, or they may be stored in the information model being considered within the IETF.

5. Policy Architecture for Content Distribution Networks

This section describes the architecture of a CDSP node as well as an origin site when the policy based framework is used to drive the operation of a content distribution network. The basic architecture is as shown in Figure 3.

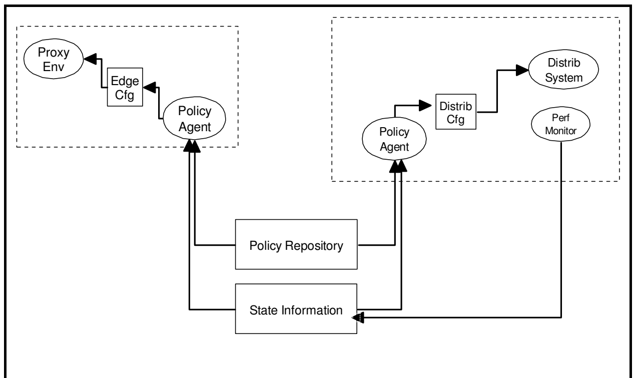


Figure 3

The policies are stored in a policy repository. We assume that there is a performance monitor component that collects the current state of the network, and stores it in a state repository accessible by the various sites. Each of the edge-servers and the origin site need to have specific contents active as shown in the figure. We assume that the performance monitor that determines state is at the origin web-server, since it would typically want to control the times when application offload begins or ends.

In addition to the performance monitor, both the edge-site and the origin site need a policy agent. The policy agent is responsible for retrieving the content distribution policies from the single repository, and for determining the appropriate configuration at the local site for content distribution. The origin site configures its content distribution system to act in accordance with the distribution policies. The edge-server configures its environment to ensure that the policies are complied with for offloading specific applications.

The other components of the policy system are not shown in the environment. The policy agent to policy repository communication would happen in the same manner as in the general policy framework. As an example, it can be done by means of policy agents that implement LDAP clients to retrieve policy information from the repository, which is an implementation in accordance with the IETF information model [5].

A policy based framework as defined above can simplify the complexities involved in the operation and management of a large content distribution network. This also enables the creation of dynamic policies, wherein the characteristics of an application can be used to determine the fraction of the application components that can be deployed at an edge-server. Enabling dynamic policies can result in significant cost-savings and performance gains for the users of content distribution. In the next section, we look at the gains that dynamic policy mechanisms can provide to the users of a content distribution service provider.

8. Dynamic Policy Management

In order to understand the potential benefits of a policy based system, we look at the cost savings that can be obtained in a content distribution environment when we include the use of dynamic policies. We consider a symmetric environment as shown in Figure 4, where the content distribution network consists of the server in the center of the circle and a number of proxies that are distributed around the server at equal distances. Each proxy services a set of clients which we assume are also at equal distance from their closest proxies.

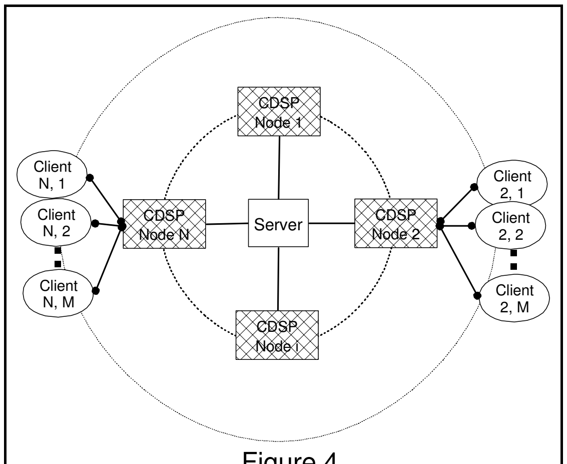


Figure 4

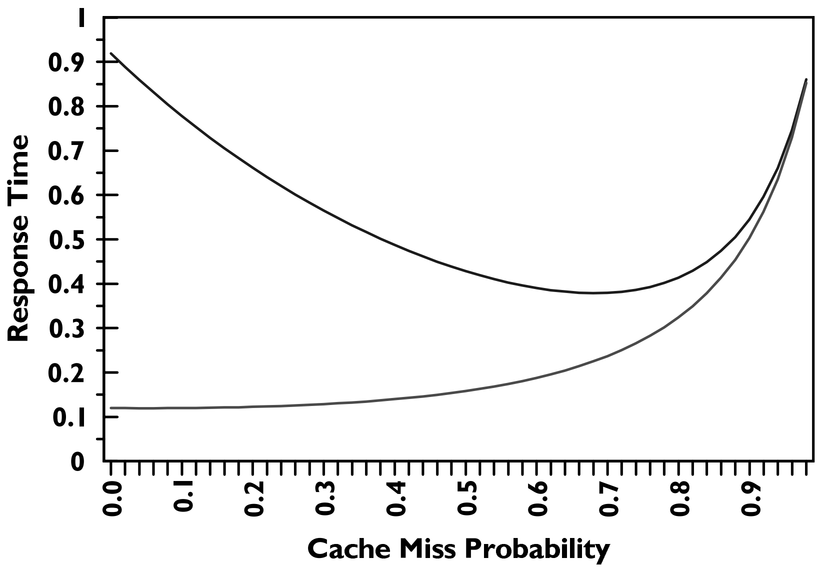
We assume that a fraction p of the requests originating from a client are serviced directly by the central server, while the remaining fraction of the requests are serviced by the proxy. We further assume that: the average latency between the client and proxy is τ_p ; the average latency between the client and the central server is $\tau_p + \tau_d$; the requests arrive at the rate of λ per client, and there are n clients; the service rate of a proxy is μ_p requests per second; and, the service rate of the central server is μ_s requests per second. Using M/M/1 models for the processing time at the servers leads to an average response time of

$$R = \tau_p + p\tau_d + (1-p)/(\mu_p - \lambda(1-p)) + p/(\mu_s - np\lambda)$$

The solution is bounded provided that $\mu_p > \lambda(1-p)$ and $\mu_s > np\lambda$. Using simple calculus, it can be shown that the impact of distribution upon the response time of users would strongly depend upon the factor $\Gamma = \mu_p \tau_d$. The use of distribution would only help in reducing response time from a totally centralized solution if $\Gamma > 1 - \mu_p\mu_s/(\mu_s - n\lambda)^2$. In these cases, the distribution could either result in a decreasing response time (i.e., the best response time is obtained with full distribution), or there is a optimal distribution point where the response time is minimal. The other parameters in the solution determine whether or not the optimum point is less than full distribution.

Figure 5 shows the result of plotting response time as a function of the cache miss probability for 10 servers when τ_p =0.01, τ_d =0.01, μ_s = 10 and λ = 0.9 for two values of μ_p. For the dashed line, the value of μ_p is 2 while for the solid line, the value of μ_p equals μ_s of 10. For the dashed line, an optimum response time is obtained when the server processes 70% of the requests, and offloads 30% of the requests to the proxy servers. For the solid line, where the proxies are comparable in performance to the central server, full distribution is the optimum solution.

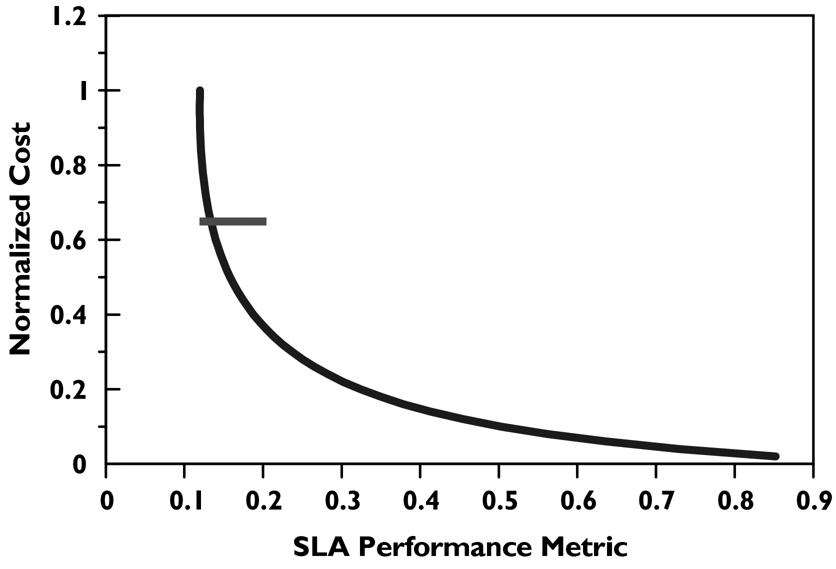
Figure 5



A direct benefit of using the policy based scheme is to reduce the costs of operating a content distribution system while satisfying a given response time SLA. In many content distribution solutions, the cost of operating the system is directly proportional to the amount of content offloaded. This is true in the model offered by content distribution providers such as Akamai, who charge by the amount of content they serve from their infrastructure. In this case, a dynamic policy based system can minimize the cost for a specific SLA, and a given customer load. A solution with a static caching ratio will have a fixed cost for a given traffic load and a response time target. A solution with dynamic policies is able to determine the offload ratio in accordance with the desired performance targets, and can thus save in the cost of offloading the contents to remote sites.

Figure 6 shows the cost of supporting a specific response time SLA with and without dynamic policy based management. It is assumed that the CDN consists of 10 servers, τ_p=0.01, τ_d =0.01, μ_s = 10, and μ_p = 2. In the case of a static offload, the cost depends on the a-priori fixed offload percentage. Thus, there would be a fixed upper response time beyond which a fixed offload scheme will not be able to attain the response time goal. A fixed offload scheme with an SLA target of 0.2 time units (in terms of the scale of Figure 6) can be supported by serving about 35% of the content only from the central server. This scheme is shown by the vertical line.

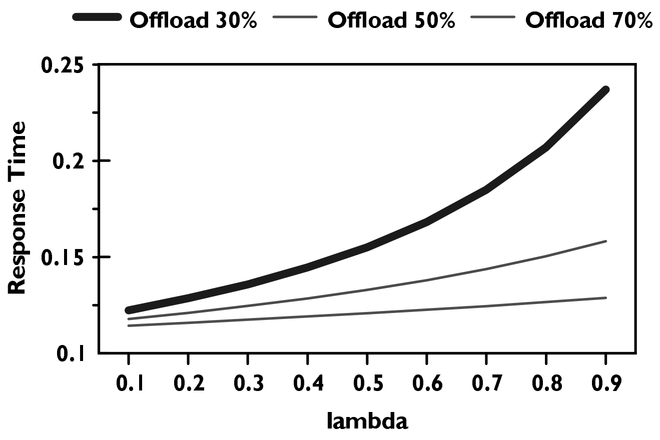
Figure 6



The disadvantage of the fixed scheme is that it can not accommodate the dynamic characteristics of a typical communications network. Since the workload on the server is not likely to stay constant, the workload generated by each client can play a dominant role in the response time perceived by the client. Figure 7 shows the results of the maximum response time that can be achieved at different client load levels using dynamic policies at a fixed cost. If we were using a fixed cost and a non-dynamic approach to content distribution, the range of load that could be supported would be limited, and a maximum load would need to be determined a-priori beyond which a response time SLA could not be supported.

Our analysis of the dynamic policy based distribution assumes that the applications can be offloaded to any arbitrary level of granularity. However, this would not be the case for a real application where the offload would be limited by application characteristics as well as latencies in measuring current client load and server response times. However, the curves shown in this section provide an upper bound on the effectiveness of dynamic policy distribution.

Figure 7



9. Summary & Conclusions.

In this paper, we have examined how policies can be applied to the topic of content distribution networks. We have extended the policy framework to this new domain, and proposed an architecture for policy based content distribution. We have also provided a taxonomy for specifying policies related to content distribution. With the use of the proposed policy based architecture, we show that dynamic content offload decisions are possible, and we analyze the impact of such decisions on the response time SLAs that can be provided to users of a content distribution network.

In ongoing work, we are exploring prototype implementations of the policy based architecture, and investigating schemes that would enable the offloading of dynamic content to edge caches in a real-world environment.

References

[1] R. Braden et al., *Resource ReSerVation Protocol (RSVP) Version 1 - Functional Specification*, RFC 2205, September 1997.

[2] S. Blake et. Al. *An Architecture for Differentiated Services*, Internet RFC 2475, Decemember 1998.

[3] Akamai Technologies Inc., "*FreeFlow content distribution service*," www.akamai.com.

[4] Digital Island Inc., "*Footprint* ***content distribution*** *service*," http://www.digitalisland.net/services/cd/footprint.shtml.

[5] B. Moore et. al., *Policy Core Information Model -- Version 1 Specification*, RFC 3060, February 2001.

[6] J. Fritz Barnes and R. Pandey, “CacheL: Language Support for Customizable Caching Policies”, Proceedings of the 4th International Web Caching Seminar, San Diego, CA. March 1999, http://workshop99.ircache.net/Papers/barnes-abstract.html.

[7] N. Damianou, N. Dulay, E. Lupu and M. Sloman, *The Ponder Policy Specification Language*, Workshop on Policies for Distributed Systems and Networks (Policy2001), HP Labs Bristol, 29-31 Jan 2001. URL http://www.doc.ic.ac.uk/~mss/Papers/Ponder-Policy01V5.pdf.

[8] P. Deshpande, K. Ramasamy, A. Shukla and J. Naughton, *Caching Multidimensional Queries Using Chunks, Proceedings of ACM SIGMOD*, 1998. Volume: 27 , Number: 2, Pages: 259 - 270

[9] G. Banavar, T. Chandra, B. Mukherhee, J Nagarajarao, R Strom, and Dl Sturman, *Multicast Protocols for Content-Based Publish-Subscribe Systems*. In Proceedings International Conference on Distributed Computing Systems (ICDCS'99), May 1999.

[10] J. Challenger, A. Iyengar, and P. Dantzig. *A Scalable System for Consistently Caching Dynamic Data*. Proceedings of IEEE INFOCOM '99, March 1999.

[11] M. Fry and A. Ghosh, Application Level Active Networking, Fourth International Workshop on High performance Protocol Architectures (HIPPARCH 98), June 98.

[12] IBM Corp., IBM WebSphere Edge Services Architecture, - Application Offload capability (page 6). http://www-4.ibm.com/software/webservers/edgeserver/doc/esarchitecture.pdf