Towards Content Distribution Networks with Latency Guarantees [*]

Chengdu Huang and Tarek Abdelzaher
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
e-mail:{*ch4pp, zaher*}*@cs.virginia.edu*

Abstract

This paper investigates the performance of a content distribution network designed to provide bounded content access latency. Content can be divided into multiple classes with different configurable per-class delay bounds. The network uses a simple distributed algorithm to dynamically select a subset of its proxy servers for different classes such that a global per-class delay bound is achieved on content access. The content distribution algorithm is implemented and tested on PlanetLab [24], a world-wide distributed Internet testbed. Evaluation results demonstrate that despite Internet delay variability, subsecond delay bounds (of 200-500ms) can be guaranteed with a very high probability at only a moderate content replication cost. The distribution algorithm achieves a 4 to 5 fold reduction in the number of response-time violations compared to prior content distribution approaches that attempt to minimize average *latency. To the authors' knowledge, this paper presents the first wide-area performance evaluation of an algorithm designed to bound* maximum *content access latency, as opposed to optimizing an* average *performance metric.*

1 Introduction

This paper presents the first wide-area evaluation of a content distribution network (CDN) designed to provide delay bounds on content access. The predominant use of the Internet for Web content delivery has recently spurred much research on CDN performance. Existing research on CDNs includes techniques for efficient client request redirection [1, 2, 37, 3], server placement strategies to improve average response time or bandwidth consumption [20, 25, 11, 15, 27], logical overlay topologies for large-scale content distribution [34, 28, 29], consistency maintenance mechanisms [38, 23], and empirical CDN performance measurement studies [14, 18]. Little attention has been given to QoS guarantee issues in CDNs. In this paper, we address the specific problem of providing subsecond guarantees on access delay. The paper evaluates the feasibility and cost of providing such guarantees on the current Internet. By "guarantee", we do not mean deterministic delay bound guarantees, which are virtually impossible without controlling the Internet backbone. Instead, in this paper, we use the word "guarantee" in a loose sense, to mean trying to achieve latency bounds with a high probability. Our evaluation results are drawn from an actual implementation and deployment of a service prototype in a world-wide WAN testbed. The results explore the trade-off space between algorithm cost and its efficacy in meeting the desired latency bounds. The trade-offs demonstrate the practical feasibility of providing latency guarantees in content distribution networks.

Operationally, our network is composed of a large set of content distribution proxies. Content is distributed across a minimal subset of these proxies such that the contractual delay bound is met for all requests. These

[*]The work reported in this paper was supported in part by the National Science Foundation under grants CCR-0093144, ANI-0105873, EHS-0208769, and MURI N00014-01-1-0576.

proxies are updated sporadically throughout the day, as is the case with sports and news sites, but not necessarily in a continuous fashion (as opposed to streaming media). Regular web caching, outside the purview of our service, may create further copies of content on demand depending on client access patterns. Content providers negotiate with our distribution network the intended service access latency for their content, defined as the time elapsed from the arrival of a client's request to some CDN server until the requested content has been sent to the client. This time includes the delay resulting from forwarding the request within the CDN if the initially accessed server does not have a local copy of the content. The content distribution network dynamically determines the number and locations of content distribution proxies that will need to host the content in order to globally guarantee the negotiated bound on access delay.

The remainder of this paper is organized as follows. Section 2 provides a brief feasibility study for content distribution with latency guarantees. Section 3 elaborates on the service model. The distributed algorithm for proxy selection is described in Section 4. Implementation is described in Section 5. An extensive performance evaluation is presented in Section 6. Section 7 presents a survey of previous work. The paper concludes with Section 8.

2 Feasibility of Delay Guarantees

The first question that needs to be answered before embarking on building our content distribution framework is whether the cost of provisioning subsecond latency guarantees is practical. This cost is related to the Internet delay variability. Highly volatile delays will make it difficult to choose stable locations for the replicas, and hence increase the cost of the bounded-delay service. In this section, we first confirm that Internet delays generally have a large distribution that is not always in the subsecond range. Hence, a delay-sensitive content distribution service is needed if subsecond guarantees are to be provided. Next, we investigate the variability of delays between pairs of nodes to assess the cost of an algorithm that will choose replica locations based on current delay measurements.

To illustrate the need for latency guarantees in content distribution, we first conducted a simple experiment on the Internet to study the network performance in the absence of a delay-sensitive content distribution service. In this experiment, a web server was setup on a PlanetLab node. Thirty other PlanetLab nodes were chosen as HTTP clients. During a 24-hour period, the clients kept sending HTTP requests directly to the web server, asking for files of sizes range from 1KB to 64KB. For the sake of separation of concerns, we intentionally kept the clients' request rates low to avoid web server overload, such that the latencies we measured were predominantly network latencies.

The cumulative distribution function (CDF) of measured latencies of all client requests is plotted in Figure 1 for files of different sizes. From the graph, we can see that the average latencies for these files are a significant fraction of a second, and that larger files may observe delays in excess of a second with a non-trivial probability. For example, for files of 40KB, 40% of the requests have latencies longer than 500ms and 10% of requests have latencies longer than 1s. Humans are able to perceive delays as small as 100ms. Hence, we believe that delays in excess of 200-300ms are generally not acceptable. This experiment evidently demonstrates that although network latencies for small files are generally small, there is a strong need for a delay-sensitive content distribution service if delays are to be maintained within a small margin from the human perception threshold.

Having introduced the need for delay-sensitive content distribution, the next question is whether it is practical to assume that such a service, if constructed, will be able to maintain delays within small subsecond bounds at reasonable cost, in the presence of delay variability on the Internet. The cost is clearly related to the amount of variability in network delays. The higher the variability the more effort is spent on replica placement and rearrangement to meet delay guarantees. This variability is illustrated in Figure 2. In this experiment, we chose four representative clients and plotted the latencies in requesting files of 16KB. Intervals between each measurement were roughly 5 minutes and the entire duration of the experiment was about 24 hours. The web server was placed in Virginia and client #1, #2, #3 and #4 were in North Carolina, Missouri, Utah and France respectively. The order of their average latencies matches the order of their geographic distances to the web server site.

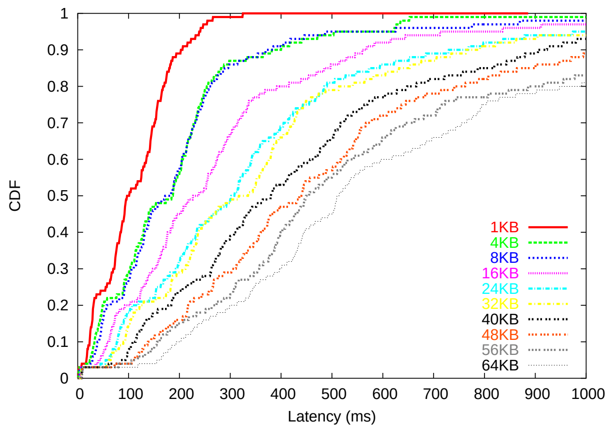


Figure 1. Clients Perceived Latencies Without Replicating Service

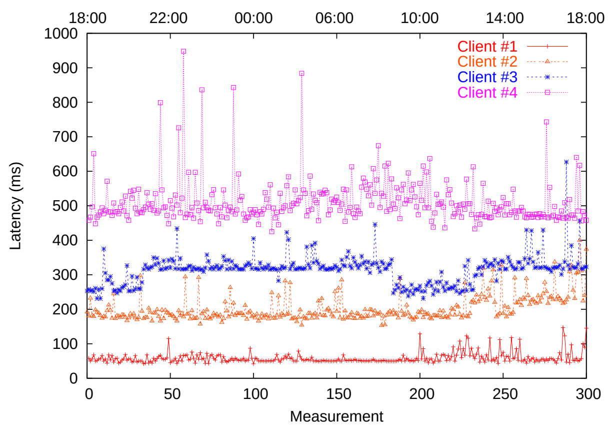


Figure 2. Latencies of 4 representative clients. File size = 16KB

The results give us some evidence that although clients' perceived latencies are not time-invariant, latencies for the same client-server pair oscillate within a relatively small range most of the time. There exist some spikes, which can probably be attributed to network instability, but the percentage is very low. The latency changes seen on client #3 were probably due to a network route change. An important observation from these data is that clients having lower average latencies also have fewer fluctuations. This can be explained by the fact that shorter latencies usually map to fewer network hops. Roughly speaking, the more network hops a flow travels, the higher the chance that its packets get long delays. We changed the site of our web server and repeated the experiments several times. Similar trends were observed in each run. Figure 2 serves as some evidence that network latencies are in general not highly variable. This could be attributed to the relative underutilization of the Internet backbone [31] and is also observed by industry in reported backbone delay statistics [32].

Hence, if a distribution service can replicate content so that a replica is always located within a short bounded latency from any content proxy then client requests can mostly be served within the desired delay bound. In the rest of this paper, we describe such a content distribution service and evaluate its performance on a distributed Internet-based testbed.

3 System Model

Consider a CDN system with four parties: content providers, the CDN provider, ISPs and clients. Content providers outsource content distribution and authorize the CDN provider to host and replicate their content objects. Along with the establishment of this content distribution business relationship between a content provider and a CDN provider, a QoS contract is specified. The QoS contract specifies the latency bound on access to the content provider's objects. Different content providers can have different latency bound requirements and therefore have different QoS contracts with the CDN provider. Content objects of a same content provider but of different sizes can have different latencies. There are many ways to specify latency bound values for objects of different sizes. One way is to provide a uniform latency bound value for objects of sizes up to a certain upper bound. Another way is to make the latency bound a function of object size, making larger objects have larger latency bounds. In our current implementation, the latency bound is a step function of object size:

$$L(s) = L_{base} + L_{incr} \times \lfloor s/S_{step} \rfloor \tag{1}$$

where s and $L(s)$ are the size and latency bound of the object and L_{base}, L_{incr} and S_{step} are negotiable QoS parameters. Each class of objects has a QoS contract in the form of Equation (1). The discretization of achievable bounds allows the system to classify content into large aggregate classes by latency bounds, as opposed to having to maintain a different bound for each object individually depending on its size.

Having established a QoS contract, the CDN provider replicates the content objects on its servers. In our CDN model, servers of the CDN provider are deployed on the edge of the Internet backbone and may be colocated with ISP points of presence or with entry points into ISP backbones. This is known as the *colocation* model. One advantage of this model is that CDN servers can relatively easily get access to some important information such as client request rates of different content objects that can be collected by ISP entry points.

We assume client requests can be redirected to the nearest CDN server using various request routing mechanisms [1, 2]. The ISP may already employ web caches which will attempt to serve such requests. Cache misses are redirected to the nearest (or colocated) CDN server. When a CDN server receives a request, if the requested content object is available at the server's local storage, reply is sent back to the client directly. Otherwise, the CDN server acts as a proxy and forwards the request to some replica (or to the origin server). The forwarding module of a CDN server is responsible for picking the best replica among those replicas of the requesting object that the CDN server knows about. When a reply is received from the selected replica, the reply is sent back to the ISP proxy from which the request originated. Observe that the ISP and CDN provider can be the same. AT&T, for example, also offers content distribution services.

The latency perceived by a client when requesting a content object hosted by our CDN service actually consists of two parts, latency between the client and ISP entry point/CDN server, which is known as the last-mile problem, and latency caused by requesting the object from some peer CDN server or the origin server across the backbone when the requested object is not available at the CDN server that received the client request. The second part is the focus of our service because the last-mile problem is usually out of a CDN provider's control. Also, as broadband services of ISPs become more and more popular, we envision that the last-mile problem will become less severe in the future.

Our CDN service causes clients to experience improved response time because of additional deployed replicas. Content providers benefit by improving the experience of their clients and by offloading web requests from their servers. In a commercial deployment, these providers will cover the service cost. ISPs also benefit from our service because with a colocated CDN server, the cache miss traffic that needs to be sent across the backbone is reduced.

4 Algorithm

In this section, we formulate our QoS guarantee problem as a graph domination problem. A distributed algorithm is then presented to solve this problem. Our replica placement has to ensure that requests received by the CDN servers are served within the latency bound specified in the QoS contract. In the extreme case, if the content is replicated to *all* CDN servers, network latencies of requests for the content will be minimized. However, the cost of replicating the content everywhere is large. First, content update traffic is not free. We envision future content that is volatile and requires frequent updates (e.g., live coverage of sports events with photographic snapshots, commentary, and video clips). The fewer the number of replicas the lower the update cost. Second, having too many replicas for each content object can significantly reduce overall throughput of a CDN system. The reason lies in the fact that partitioning client requests to too many different replicas lessens temporal locality of content access requests. As a result, many client requests have to be served from servers' disks instead of memory. This phenomenon has been confirmed in previous research [37]. Finally, depending on content size, storage cost may or may not be an issue. For example, consider a future Starbucks chain that entertains its customers by offering wireless access to real-time (still) snapshots taken by webcams located in its 3000 T-mobile enabled branches. To make it entertaining, the service offers VCR-like functionality that allows users to fastforward and rewind through animations of the stills of any location worldwide, recorded within the last week. Assuming 10KB pictures taken once every 5 seconds, and 3 webcams per branch, the total storage requirements for all the weekly pictures combined is roughly 11 terabytes. While this particular application will have other bottlenecks, the point is that storage requirements of individual CDN customers might be non-trivial.

Hence, we state the content distribution problem as that of finding a replica placement that uses the minimum number of replicas and ensures that requests received by any CDN server of the content provider can be served within the latency bound.

Our CDN servers form an overlay network. Although every CDN server can reach every other server via the underlying IP routing, only links whose delay is less than the bound are valid in the overlay topology. Observe that, under these conditions, different content with different QoS requirements could have a different view of the overlay network. The higher is the delay bound the more connected is the network. An arbitrarily high delay bound results in a fully connected network as all links in the underlying IP network satisfy the bound. Due to the asymmetric nature of Internet routing, all the edges are directional, meaning an edge from server S_i to S_j does not imply the existence of an edge from server S_j to S_i. With this in mind, we can formally state our QoS guarantee problem as follows. Given a set of CDN servers $\mathcal{S} = \{S_i | i \in [1, N]\}$, a content object C, and its latency bound L, we construct an overlay network whose nodes are the CDN servers $\mathcal{S}$. Edges are added following the rule that $\overrightarrow{S_i S_j}$ exists if and only if S_i can download C from S_j within a time less than L. Trivially, $\overrightarrow{S_i S_i}$ exists for all $i \in [1, N]$. Our objective is to find a minimum size subset $\mathcal{D} \subset \mathcal{S}$ to hold replicas of C, such that for every node S in $\mathcal{S} - \mathcal{D}$ there exists at least one edge $\overrightarrow{SD}$ in the overlay network such that $D \in \mathcal{D}$. Observe that because of the way edges are defined, the aforementioned condition implies that the delay between any node S and at least one replica D is less than the bound.

The algorithm to solve the above replica placement problem classifies files into discretized categories by latency bound. The algorithm needs to run for each content class (with a common latency bound) instead of for each content object. To further reduce the scale of the problem, observe that network edges which satisfy a shorter bound also satisfy a longer one. Hence, approximate solutions for a longer latency bound can be derived incrementally from solutions for shorter latency bounds. In this paper, we separate probing network delay from running the placement algorithm. Network probing is done in a common module. The placement algorithm runs concurrently (separately) for each content class.

4.1 Centralized Algorithm

Consider the graph domination problem formulated above for a single content class. A set of vertices of a graph is called dominating if every other vertex is adjacent to at least one vertex of the set. The graph domination problem asks for a dominating set of the minimum possible size in a given graph. Graph domination is known to be NP-Hard [8]. A common approximation algorithm for the graph domination problem is the greedy algorithm shown in Figure 3. Vertex S_i is said to be able to "cover" vertex S_j if and only if $\overrightarrow{S_j S_i}$ is in the graph. In accordance with previous papers on dominating set algorithms (e.g. [12]), we call the number of uncovered vertices that a vertex can cover (including itself) the *span* of the vertex. In our overlay network, the span of some CDN server, S, is the number of CDN servers that can download the content object from S within specified latency bound. The intuition behind the greedy algorithm is that the vertices with high span values are more likely to be selected in a dominating set. Hence, the heuristic orders vertices in decreasing order by span and selects from the top recursively until a dominating set is reached.

It is known that the above greedy algorithm achieves an H_Δ-approximation where Δ is the maximum degree of the nodes and H_i is the ith harmonic number (i.e. $H_i = \sum_{k=1}^{i} \frac{1}{k}$) [6, 13]. This is actually the best approximation known so far. The greedy algorithm is very simple but has its drawbacks in practice. Most importantly, it requires a centralized coordinator to perform the replica assignment. It assumes that network topology is known to the centralized coordinator, which in this case means that the coordinator knows the latency from *any* node to *any* other node in the network. Such centralized algorithms generally have scalability and robustness limitations and lack the flexibility of doing online adjustments to accommodate changes in network conditions. We use the aforementioned centralized greedy algorithm only as a baseline to compare against. Our goal is to find a distributed algorithm that would perform close to the centralized one.

4.2 Distributed Algorithm

A few distributed algorithms for constructing dominating sets have been proposed. Liang and Haas used a distributed algorithm called DDCH [21] to generate virtual backbones for Ad Hoc networks. When synchronously executed, DDCH achieves the same approximation ratio as the greedy algorithm (Figure 3) but there exist networks for which DDCH takes a very long time ($\Omega(n)$ rounds, n is the number of nodes) to complete. LRG introduced by Jia et al.[12] is a refinement of DDCH, in which randomization is used to break symmetries when multiple nodes attempt to add to the dominating set. It's proven that LRG terminates in $O(\log n \log \Delta)$ rounds with high probability, where Δ is the maximum span of the graph and yields an approximation ratio with expectation of $O(\log \Delta)$. Kuhn and Wattenhofer designed a more sophisticated distributed algorithm [19] based on LP relaxation techniques. Given an arbitrary constant k, the algorithm computes a dominating set of expected size $O(k\Delta^{2/k} \log \Delta)$ times optimal in $O(k^2)$ rounds. Though theoretically appealing, the performance of these algorithms when used in asynchronous systems was not studied. Moreover, these algorithms share a common disadvantage: they need multiple rounds to finish. Kuhn's algorithm [19] can be made to finish in one round by setting $k = 1$ but then the performance will be far from optimal. In practice, more rounds translates into more overhead and a longer termination time, which may be problematic if network conditions change before the algorithm terminates.

In this paper, we seek an algorithm that is both very fast (ideally, constant-time) and yields a small dominating set in a highly asynchronous execution environment. We designed a simple distributed algorithm to meet these requirements. The algorithm is fully distributed and asynchronous. Figure 4 shows the pseudocode that is executed at each node independently. We require that every node know only its own span value and the number of nodes *it* can reach within the delay bound including itself. Each node sends its span (SPAN), i.e. the number of nodes that can download the object from it within the latency bound, to all the nodes it can cover and collects other nodes' spans. The message specifies the span for each content class. If a node can only be covered by itself, the node has to be in the dominating set and sends a DOMINATOR message to nodes it can cover claiming that it is in the dominating set. Otherwise, after it gets SPAN messages from all the nodes that can cover it, it chooses the node with highest span among them (which could be itself) and makes that node a member of the dominating set by sending it a NOMINATION message. When a node receives a NOMINATION message, it joins the dominating set. It then sends a DOMINATOR message to nodes it can cover and will not nominate another node if it has not sent out any NOMINATION message yet. Assuming no message loss, since every node selects a node that covers itself, when all nodes exit the algorithm the graph is dominated.

The mechanism that each node independently chooses a nominee makes the algorithm be able to terminate in one round. This mechanism only uses information about direct neighbors but is very effective in choosing the right nodes to be added to the dominating set. When a node S_i is nominated by S_j, it will be added to the dominating set despite the possibility that S_i may not be the highest span node among the nodes that can cover it, or it has already been covered by some other node, and hence would not nominate itself. However, since S_i is the highest

```
/* all vertices start as "uncovered" */
while (there is "uncovered" node in the graph) {
    select the vertex with highest span;
    add the vertex to dominating set;
    mark the vertex and vertices it can covered as "covered";
}
```

Figure 3. Centralized Graph Domination Algorithm

span node among nodes that cover S_j chances are in order to cover S_j, S_i end up still being the right node to join the dominating set. The rule that receiving a DOMINATOR message from some adjacent node makes a node refrain from nominating another is very helpful for reducing the size of the final dominating set because senders of the DOMINATOR messages are already in the dominating set and nomination can only possibly cause a new node to be added to the dominating set. Note that the NOMINATION and DOMINATOR messages are class-specific. However, if the same server is nominated for multiple classes, the corresponding messages are coalesced.

```
/* For node $S_i$ */
send SPAN (span of $S_i$) to all nodes;
if (I am the only node can cover me) {
    join the dominating set;
    send DOMINATOR to nodes I can cover;
    return;
}
while (true) {
    if (receive DOMINATOR from $S_j$ && $S_j$ covers me) {
        send COVERED to nodes I can cover;
        return; /* $S_i$ is covered by $S_j$ */
    }
    if (receive SPAN from $S_j$)
        updates span of $S_j$;
    if (receive COVERED from $S_j$)
        decrease span $S_j$ by 1;
    if (receive NOMINATION from $S_j$) {
        join the dominating set;
        send DOMINATOR to nodes I can cover;
        return;
    }
    if (received SPAN of all nodes that can cover me) {
        select the node $S_k$ with highest span (with random tie-breaking)
           from nodes that can cover me;
        send NOMINATION to $S_k$; /* $S_i$ is covered by $S_k$ */
    }
}
```

Figure 4. Distributed Graph Domination Algorithm

This distributed algorithm does not guarantee the same performance as its centralized counterpart. It is not difficult to find scenarios in which the centralized greedy algorithm gives a smaller dominating set than the distributed algorithm. However, there also are scenarios where the distributed algorithm performs better than the centralized one. In Section 6.1 we give a detailed performance evaluation and comparison of the centralized greedy algorithm, previous distributed algorithms mentioned above, and our distributed algorithm.

5 Implementation

We implemented our content distribution service by instrumenting a Squid Proxy Cache [33] and deployed it on PlanetLab. Although the distributed replica selection algorithm is simple in appearance, there are many technical issues that need great care when implemented on a real system and deployed in a realistic WAN environment. These issues are described below.

5.1 Probing

In Section 4.2, we assumed that nodes know their span values. This knowledge does not come for free in the real world. When our system is to host a content class, we may need to construct the overlay network for the class before invoking the distributed replica selection algorithm. This is done based on probing results. Probing is an independent operation that is performed for different files sizes and the results are interpolated to provide delay information for an arbitrary file size.

A CDN server probes each peer a few times by sending requests for content of different sizes. The measured delays are fitted to a linear equation describing delay as a function of content size. Curve fitting is done using a least squares estimator. The parameters (slope and intersect) of the delay equation for a given peer are then sent to that peer in a REACHABILITY message.

Each server aggregates REACHABILITY messages and computes its span for each class. The per-class span values are sent out in a SPAN message. In addition, we use a *safety margin* parameter to help cope with fluctuations in network latencies. A safety margin of $s, (0 < s \leq 1)$ means that only if the probing delay (obtained from the delay equation) is less than sL, where L is the latency bound, will the server consider the peer reachable. The typical safety margin we used in our experiments was 0.75.

Since our probing actually downloads files from the remote servers, it has a higher overhead compared to other approaches such as sending ping packets. However, the overall probing cost remains acceptable because (1) probing happens between servers of the CDN provider at a very low frequency, (2) probing does not have to be active all the time. When the system is running, client requests forwarded by one CDN server to another can serve as measurements, and (3) traffic volume of probing messages is typically orders of magnitude smaller than traffic generated by clients' requests. Probing is a relatively independent module of our system and is not the focus of this work. Exploring more sophisticated probing techniques could help construct more accurate overlay topologies but is beyond the scope of this paper.

5.2 Asynchronous Execution

Imposing a synchronous execution mode on a large scale CDN system is not acceptable. The performance of a distributed algorithm in an asynchronous mode is the meaningful metric for our system. The distributed algorithm implemented in our system is able to run asynchronously. In the asynchronous mode nodes constantly collect control packets (SPAN, DOMINATOR, COVERED, and NOMINATION) from other nodes. After a certain pre-defined period, each node makes nomination decisions, if it has not sent any NOMINATION messages yet, based on the information it has at the instant. This guarantees that a dominating set will be generated but the cardinality of the dominating set could be larger than that generated in a synchronous execution. In [21] the authors discussed the asynchronous operation of DDCH, which uses a periodic mode too, except that their algorithm usually takes multiple rounds to finish. The mode can also be applied to LRG [12] since it is a refinement of DDCH. We followed that mode in our implementations of DDCH and LRG.

5.3 Replica Selection

A server can receive multiple DOMINATOR messages from different reachable servers for the same content class. Since all these servers are replicas and can be reached within the latency bound, any of them can be the server's replica. Recall that we mentioned in Section 2 that when latencies between two servers are low on average, the variances of the latencies tend to be low too. Based on this rationale, we make the server choose the replica that has the lowest latency as its primary replica. Other reachable replicas are marked as backup replicas. Note that the origin server always has valid copies of content. It is automatically a backup replica for all the CDN servers that can reach it within the latency bound.

5.4 Periodical Invocation

The assignment of replicas is not fixed for the lifetime of the system. The distributed replica selection algorithm is actually invoked periodically using the most recent probing data. This enables the system to self-adjust to accommodate network traffic and server workload condition changes. Probing enables the servers to have a fairly up-to-date view of latencies to other servers all the time. When a server is about to run the replica selection algorithm, it first sends out REACHABILITY messages to other servers according to its current latency statistics. After these messages are exchanged among the servers, they can broadcast SPAN messages to report their span values and run the distributed algorithm.

Adjustments to replica selection can be done in between periodic algorithm invocations as well. When a server detects that a high percentage of client requests forwarded to its primary replica have recently exceeded the latency bound, it infers that there may be some problem with its primary replica. In this situation, if the server has backup replicas, it will pick one of them and start forwarding client requests to this backup. However, this switching is just temporary. The server will try to fall back to its primary replica after some time by sending some probings in the hope that latency to its primary replica is back to normal. This temporary diversion process may be repeated multiple times. If probing sent to the primary replica still yields latencies higher than the latency bound, the server will give up its primary replica and make one of the backup replicas its new primary replica. If the server unfortunately does not have a backup replica, then it has to make itself be the replica. Empirically, this situation rarely happened in our experiments on PlanetLab. The mechanism above can also be used to handle server failures.

6 Evaluation

In this section, we evaluate the performance of our system and the effects of various design considerations on the efficiency of our distributed replica selection algorithm. We begin our evaluation by investigating the two key performance metrics of our algorithm; namely, the number of replicas it generates, and the accuracy with which it satisfies a desired latency bound. To evaluate the number of replicas, we compare the number generated by our distributed algorithm to those generated by the centralized greedy algorithm and two previous distributed algorithms mentioned in Section 4. To evaluate delay bound satisfaction, we compare our algorithm to several baseline algorithms that include a trivial baseline, a random placement algorithm, and an algorithm drawn from prior literature. We also explore techniques to handle different file sizes. A discussion follows to analyze our system's ability to achieve load-balancing and scalability. Finally, we explore the performance of the algorithm in the absence of accurate system knowledge. In particular, it is shown that the algorithm performs well even if individual CDN nodes are aware of only a small subset of other CDN nodes in the system. The evaluation demonstrates that our algorithm is able to achieve the desired delay guarantees in the absence of global knowledge and at a very moderate cost.

6.1 Evaluation of the Number of Replicas

As discussed in Section 4, the number of replicas (NOR) needed for a certain content object directly relates to the cost of hosting the object and is one of the primary performance metrics of our system. Our goal is to minimize the NOR. We have formulated the replica selection for maintaining a latency bound as a graph domination problem and presented a decentralized approximation algorithm in Section 4. Our replica selection algorithm runs in a purely distributed fashion. Every CDN node selects and nominates replicas autonomously. No replica knows the exact NOR in the whole system.

To understand the performance in terms of minimizing NOR, we compare our distributed algorithm with the centralized greedy heuristic listed in Figure 3 and two distributed algorithms mentioned in Section 4, namely DDCH [21] and LRG [12]. We did not include Kuhn's algorithm [19] in our comparison study because firstly

	Centralized			DDCH			LRG			DG		
30 Nodes	200ms	300ms	400ms	200ms	300ms	400ms	200ms	300ms	400ms	200ms	300ms	400ms
set 1	8	7	6	8.8	7.7	6.3	9	7.8	6.5	8.6	7.5	6.4
set 2	6	5	5	6.5	5.5	5.5	6.4	5.5	5.8	6.9	5.6	5.2
set 3	4	4	4	4.4	4.3	4	5	4.8	4.1	5	4.7	4.6
80 Nodes	200ms	300ms	400ms	200ms	300ms	400ms	200ms	300ms	400ms	200ms	300ms	400ms
set 4	19	18	17	21.6	18.8	16.3	21.2	18.9	16.7	20.6	18.3	16.5
set 5	17	16	14	17.5	16.7	14.2	17.3	16.6	14.5	17.6	16.4	14.8
set 6	13	12	12	13.6	12.2	11.1	13.3	12	11.3	14	12.9	11.4

Table 1. Number of Replicas

the algorithm uses a purely synchronous model for communication and secondly when the number of rounds k is small, performance of the algorithm is much worse than the other three distributed algorithms we studied.

We ran all the three distributed algorithms and the centralized greedy algorithm on two CDNs with different scales. One consists of 30 servers of PlanetLab, the other consists of 80 servers. For each scale, algorithms were run at three different times and for three different latency bound values. We averaged the NOR of 10 invocations of our distributed algorithm. The results are summarized in Table 1 where our distributed algorithm is labeled "DG". In this experiment, the centralized algorithm was implemented by a coordinator that we setup to collect probing results from the individual servers. Based on these probing results and the specified latency bound value, the coordinator would construct the network topology and run the centralized greedy algorithm.

Observe that although we used the same set of servers in all experiments of same scale, network conditions were slightly different, which resulted in a different NOR at different experiments for the same latency bound. We can see that compared with the centralized algorithm, the average NOR of our distributed replica selection algorithm is only slightly higher than that of the greedy algorithm when network scale is relatively small (30 nodes). When network scale is large (80 nodes), in some settings our distributed algorithm even yielded smaller NORs than the centralized algorithm. Also, the number of replicas as a fraction of the total network size did not increase with the increase in network scale. This makes us believe our distributed algorithm is simple yet has very good scalability. Moreover, it justifies using only 30 nodes in the remaining experiments instead of 80, as the results remain representative. Comparing the three distributed algorithms, DDCH, LRG and DG, we can see that no one clearly outperforms the other in terms of minimizing NOR. However, our algorithm is the simplest of the three and hence has an advantage in terms of implementation convenience.

Another metric we looked at is the termination time of the distributed algorithms which is the time elapsed until all nodes are covered. As mentioned in Section 5.2, when running in asynchronous mode, the algorithms will be executed periodically. For the fairness of comparison, we set the same period for all the three distributed algorithms. Note that our distributed algorithm can finish in constant time regardless of the network scale and topology. We define *normalized termination times* as the ratio of an algorithm's termination time over our distributed algorithm's termination time. Figure 5 plots the normalized termination times of DDCH and LRG in the experiment sets of Table 1.

From the figure, we can see that both DDCH and LRG take much longer to terminate. Another phenomenon observed about the two algorithms that makes them less attractive is that the normalized termination times for the 80-node network (set 4, 5, 6) are generally higher than those for the 30-node network. We conclude that our algorithm fits the goal of our system, which is finding a small dominating set very fast, and that it scales better than the rest. This algorithm will therefore be used in the rest of the evaluation section.

Figure 6 shows the actual replica locations in a representative instance of the above experiment. In this instance, 3 replicas are assigned in the United States and 1 in Europe. The figure also shows other CDN servers and the replicas they are connected to by our algorithm. Observe the natural clustering of CDN servers around their nearest

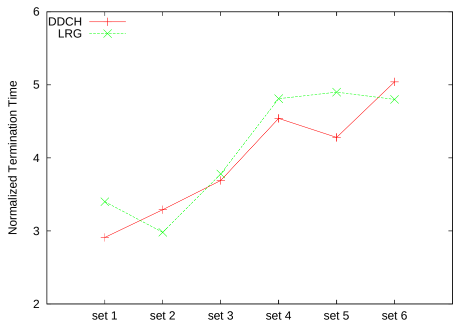


Figure 5. Normalized Termination Time

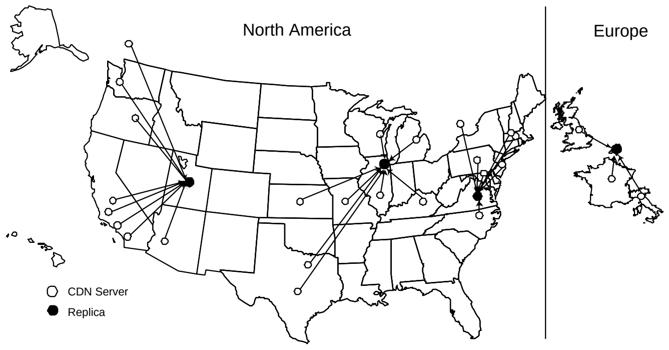


Figure 6. One Instance of Experiments on PlanetLab

replica, which provides an intuitive sanity check on the quality of our overlay structure formation.

6.2 Evaluation of the Latency Bound Guarantee

In this section, we evaluate our system's ability to meet latency bounds with high confidence. To solve this problem, we need a distributed graph domination algorithm. In the previous section, we have chosen the best algorithm to use. The next question to ask is how well the algorithm performs in terms of achieving delay bounds. Note that meeting delay bounds is a different problem from the one typically addressed in previous efforts; namely, one of minimizing average latency. We confirm below that algorithms for solving the latter do not work well for the former.

To put ourselves at the maximum disadvantage, we compare our algorithm only against latency minimization algorithms that are centralized. A centralized algorithm has the advantage of global knowledge and hence the potential to produce better quality solutions. Nevertheless, we show that when it comes to meeting delay bounds, our algorithm outperforms its centralized rivals. Notice that, unlike our distributed replica selection algorithm which determines the NOR in a decentralized fashion on its own, all the centralized algorithms compared here take the NOR as an input parameter. For the fairness of our comparison, in each experiment, we first ran our distributed algorithm. After it settled down, we collected information from all the CDN servers and determined the actual NOR yielded by the distributed algorithm. We then fed this value to all the centralized algorithms.

For the centralized algorithms we setup a centralized coordinator to synthesize information collected from each individual server. When enough information is obtained for the coordinator to construct the network topology, the centralized algorithms were invoked. The servers picked by the coordinator were then assigned as replicas. The other servers probed all the assigned replicas and, same as in our distributed algorithm, chose the one with the lowest latency. We use the following three centralized replica selection algorithms as baselines:

Single Server: In this baseline, every client request goes to the origin server. No replicas are created. This algorithm is expected to have poor performance. We include it for completeness to offer a “worst-case” point of reference.

Random: This algorithm randomly chooses a specified number of replicas among the entire set of CDN nodes. This number is the same as that used in our distributed algorithm. This algorithm is included here to separate the performance enhancement attributed to the use of multiple replicas from the enhancement that results from their non-random proper placement by our algorithm.

Average Latency Greedy: A greedy network performance optimization algorithm has been studied in [25, 11]. Although its authors mentioned that it can essentially be applied to optimize any metric, it is applicable to optimizing only overall or average performance metrics, which is not the same as guaranteeing delay bounds. Hence, in our study, we use this algorithm to minimize the overall (average) latency. We call the resulting latency minimization algorithm Average Latency Greedy (ALG). To minimize latency, ALG first computes the total cost associated with each server assuming that all the client requests received by all the N servers are forwarded to that server. It then picks the server that yields the lowest cost as a replica. This computation is repeated by looking for the next replica that, in conjunction with the previously selected ones, yields the lowest cost. The algorithm keeps selecting additional replicas until the NOR reaches a specified number. When computing cost, ALG assumes that the client requests received by a CDN non-replica server can always be directed to the nearest replica and takes the corresponding latency value as the cost. According to simulation results of [25], ALG performs very close to optimal in minimizing the average client-perceived latency in a CDN.

In addition to the centralized algorithms mentioned above, we also introduce a variant of our original distributed algorithm. In this variant, each server nominates the nearest (as opposed to the highest-span) neighbor among all the neighbors that are within the latency bound. If no neighbor is reachable within latency bound, the server itself becomes a replica. We call the original algorithm and this variant "span-first" and "latency-first" distributed algorithms respectively. The latency-first algorithm explicitly attempts to minimize latency and guarantee the latency bound at the same time. We show that latency first can reduce the average latency but may increase NOR.

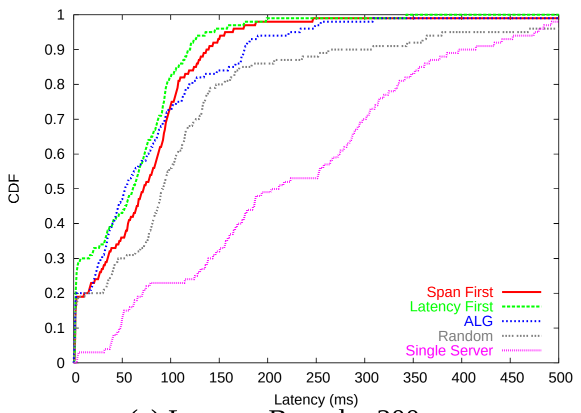
We installed the above baseline algorithms as well as our distributed algorithm on all servers. All algorithms were then run at the same time in different processes. The purpose is to make our system and baselines witness the same network conditions. For each CDN system process, clients were created that request files at a specified rate. Client processes run on servers in the same LAN as the servers running CDN system processes. We first evaluate the performance of the algorithms above for each of three different latency bound values. We collect client-perceived latencies of a total of 150,000 requests over the whole network and plot the cumulative distribution functions (CDF) of latency values. Figure 7 and Figure 8 show the cumulative latency distribution of downloading 16KB and 32KB files respectively. Similar trends were observed for other file sizes.

From the graph we can tell that our distributed algorithms, span-first and latency-first, consistently perform better than the baselines. In Figure 7 the latency bound conformity ratios (i.e., percentage of requests that met their delay bounds) are 98%, 99% and 99% for latency bounds of 200ms, 300ms and 400ms respectively while those of ALG are 94%, 94% and 95%. Similarly, in Figure 8 the conformity ratios are 98%, 99%, 99% for latency bounds of 250ms, 350ms and 450ms and those of ALG are 94%, 95% and 93%. The random algorithm has much poorer performance in terms of both latency bound conformity ratio and average latency. Having complete global information makes the ALG algorithm able to achieve comparable or even lower average latency but it is inferior to our distributed algorithms in providing the latency bound guarantee because it does not take the latency bound into consideration when selecting replicas. Figure 7 also shows that both random and ALG algorithm's performance degrade as the latency bound value increases, because NOR decreases with the latency bound.

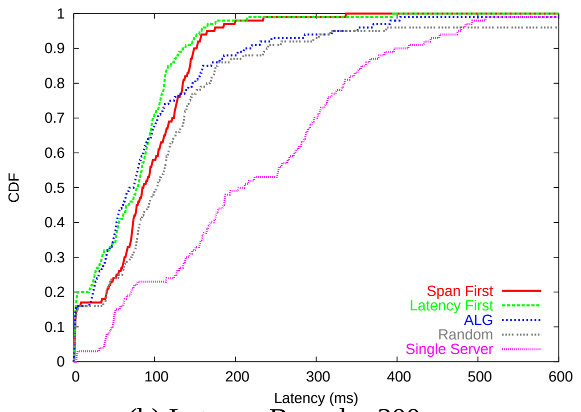
Compared to the span-first algorithm, latency-first can achieve the same or even higher conformity ratio and a lower average latency. However, it creates more replicas. The conformity ratio is better because latency-first nominates the lowest latency node among those that satisfy the delay bound. Hence, the system has a larger cushion against delay perturbations. The NOR is higher with latency-first because the metric used to nominate neighbors to become replicas (latency between two nodes) is a function of both the source and destination nodes. Hence, it may vary depending on the nominating node. In contrast, the span of a nominated node is a function of only that node. Hence, different nominating nodes are more likely to nominate the same neighbor. Consequently, the total number of nominated nodes decreases.

6.3 Effect of File Size

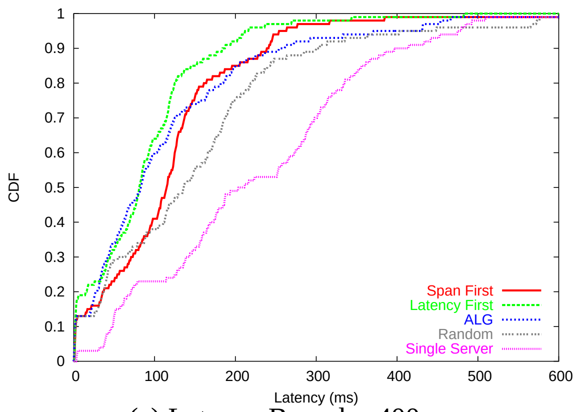
In our CDN system, each CDN node determines its set of neighbors using latency information. However, different file sizes have different latencies and web objects can essentially be of any size. Hence, we need techniques to estimate the latency of downloading an object as a function of file size using only a limited number of probes. Fortunately, our measurements show that the average network latency of downloading a file is roughly proportional to its size when the file size is between 1KB and 100KB. This range encompasses the majority of web objects [7, 30] and is the range our system is targeted for. Hence: $L(s) = As + B$. We leverage this property to quickly



(a) Latency Bound = 200ms

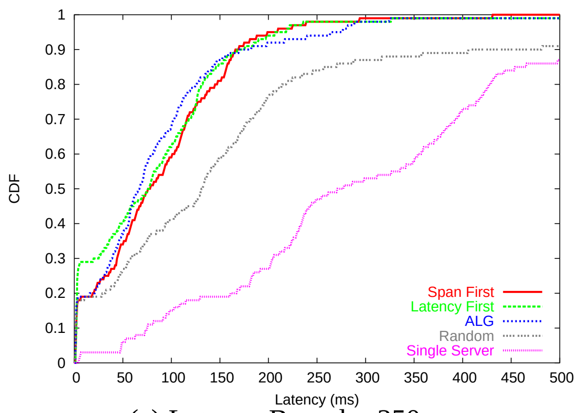


(b) Latency Bound = 300ms

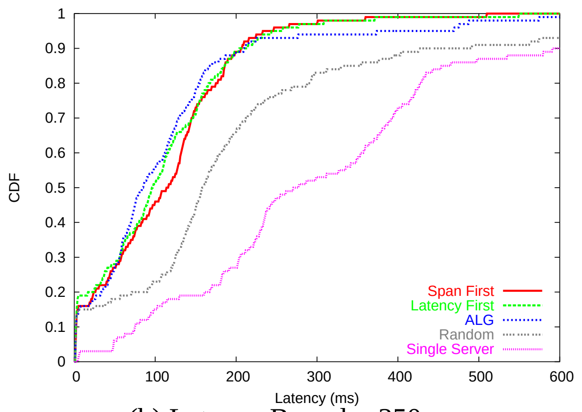


(c) Latency Bound = 400ms

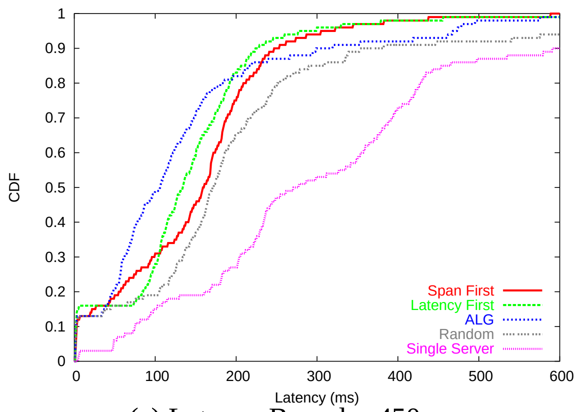
Figure 7. Client Perceived Latencies (16KB)



(a) Latency Bound = 250ms



(b) Latency Bound = 350ms



(c) Latency Bound = 450ms

Figure 8. Client Perceived Latencies (32KB)

estimate the latency function. The estimation module first probes objects of several different sizes, then uses a recursive least square (RLS) estimator to estimate the parameters A and B of the linear equation above for the given client-server pair. We remove some outliers when doing the online estimation by setting a *noise factor* f. Noise factor simply means that if the result of a measurement is out of range $[E/f, f \times E]$ where E is the estimator's prediction, the result of this measurement will not be fed into the estimator to avoid confusing it. Dynamic adjustment of estimator's parameters helps our system adapt to network and system condition changes.

We validate our estimation technique by testing it on realistic Internet servers. We setup one node of PlanetLab as the server and chose four representative nodes as clients. The clients kept requesting files of sizes that range from 4KB to 100KB and ran the estimation algorithm. The experiment lasted for over 24 hours. Figure 9 plots the average latencies of files with different sizes and the estimation values of our algorithm. We can see that when the client is relatively close to the server (server #1, #2), our estimation is very accurate. When the client is far away (server #3, #4) but the file size is under 64KB, our estimation is still accurate. When client is far away and the file size is relatively big, the estimation error of our algorithm is no more than 10%.

While we believe that the above simple delay estimation is sufficient for our purposes, other more sophisticated network measurement/estimation techniques in existing research such as bprobe [4], Pathload [10], PTR/IGI [9], and Spruce [35] can be plugged into our system, if needed, without any impact on other modules.

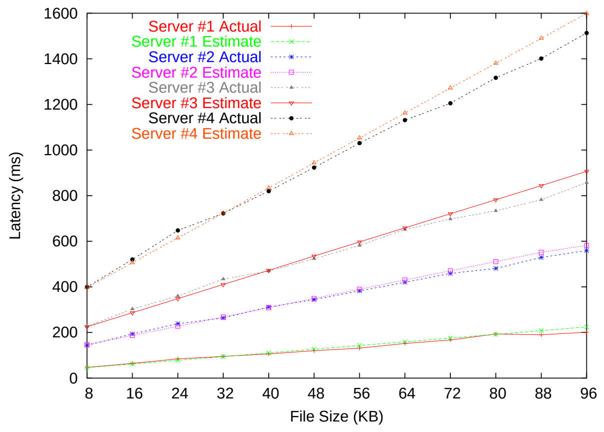


Figure 9. Latency Estimation

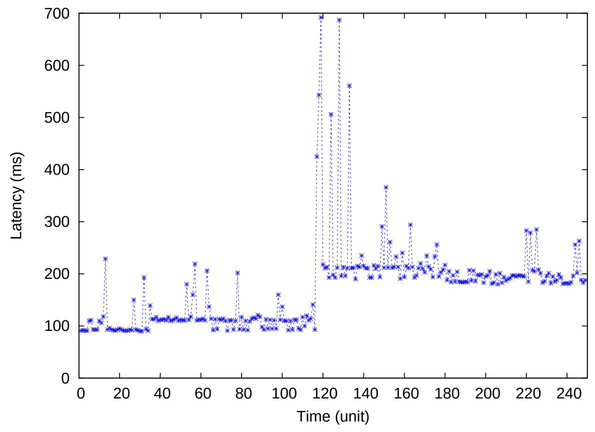


Figure 10. Overloading Test

6.4 Overload Avoidance

CDN servers can get overloaded if client requests approach or exceed server capacity, thus causing an abrupt rise in service time. In our content distribution network, overload situations can be categorized into two separate cases. One occurs when some content gets very popular suddenly, which is sometimes referred to as a *flash event*. In this case, client requests received by a single CDN server can overload that server. The other situation is when one server gets continuously selected as a replica for different content objects, for example, because of its favorable location in the network. It can therefore suffer too much traffic from both clients and peer CDN servers. To handle these situations, we need to make sure that when a server is overloaded, the content objects it hosts will be reduced and therefore its workload can be reduced.

Redirecting client requests [3, 37] is one important technique to protect servers from flash events. While this technique alleviates overload, it does not solve the delay bound problem, as the delay elapsed to reach the overloaded server is not removed by redirection. Instead, our system includes explicit provisions for avoiding server overload. Firstly, when a server broadcasts its span to its neighbors, it also attaches its workload information. CDN servers will avoid nominating servers with very high workload. Secondly, when a CDN server detects that it is overloaded, it can avoid being selected as replica by refraining from sending out its span to its neighbors. Lastly, when a CDN server is overloaded, its peers will notice an increase in the latencies seen when probing the server. Hence, the peers will exclude this server from being nominated as replica. When an overloaded CDN server is deselected as replica, some other replica(s) will be created. So when the overloaded server's workload gets reduced and it is selected as a replica again later, it will share the total traffic with the newly created replica and therefore avoid becoming overloaded again.

In the following, we present an experiment that illustrates overload avoidance in our system. Since PlanetLab is a shared platform, we were explicitly discouraged from running extensive overload experiments on its servers. Instead, we used a server S on our LAN as the overload victim. A few PlanetLab servers then chose this server as their replica. Figure 10 plots the latencies of client requests and probings of one of the servers C that chose S as its replica. At time 115, we started multiple httperf [22] clients in our LAN, sending requests at very high rates to S to overload it. C detected that latency to S became very high. Client requests directed to S continuously exceeded the 300ms latency bound. As described in Section 5.3, C then chose another replica as a temporary replica. Although the temporary replica was not as close to C as S, it was still within the latency bound. While using the temporary replica, C still periodically tried to fall back to its original replica S by probing the original replica, which are seen as the spikes from time 120 to 135 in the figure, because S used to have lower latency than its current temporary replica. After a few tries, C gave up and stayed with its temporary replica until the next invocation of distributed replica selection which happened from time 160 to 180. After that C acquired a new replica. From this experiment we can see that our system has the ability to avoid overloaded servers.

6.5 Visibility

An important consideration in the design of our algorithm was that it should perform well in the absence of any global knowledge shared among the CDN servers. This includes the knowledge of the identity of servers that are members of the CDN network. In practice, we believe that for moderately sized CDN networks that belong to the same administrative entity, it is not unreasonable to assume global knowledge of membership. Such knowledge can be preconfigured into the system and will not change frequently. However, larger systems with a decentralized administration can benefit from protocols that work well in the absence of any global information.

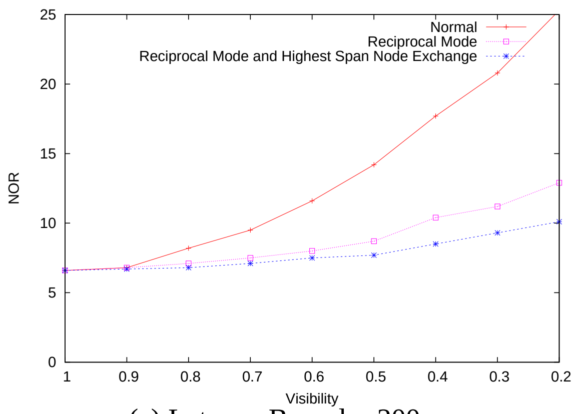
While many peer-to-peer frameworks adopt the extreme requirement that servers can only have a very local view of the network such as only a few immediate neighbors, our model lies in the middle in that we allow each server to know of some appreciable fraction of other servers. However, we do not require that this fraction be close to 1. We believe that our assumption is more appropriate for the current size of CDNs. Moreover, it avoids the negative performance implications of having to traverse application-layer multi-hop paths in a logical overlay topology, which may have a severe impact on the network's ability to efficiently meet subsecond delay bounds.

To evaluate the performance of our algorithm in the absence of global knowledge, we let each node know of only a certain percentage of other nodes of the whole system when it starts up. We call this parameter *visibility*. Imperfect visibility causes more replicas to be selected because some nodes with high spans are not visible to all other nodes. We evaluate two simple heuristics to reduce the NOR when visibility is low:

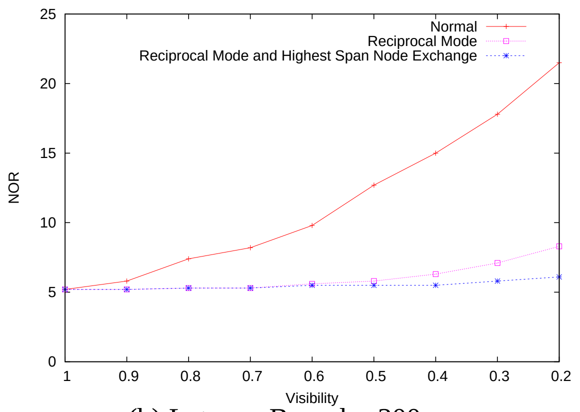
Reciprocal Mode: If servers know only an arbitrary subset of the network, cases may arise when server A knows server B but B does not know A. In this case, average visibility can be increased if each server simply introduced itself to all other servers it is aware of. This is called the reciprocal mode. Assuming there are N servers in the CDN system and every server's initial visibility is x $(0 < x \leq 1)$, then normally, the number of nodes each node knows is Nx. In the reciprocal mode, after each node introduces itself to the servers it initially knows, the average number of nodes each node knows increases by $Nx(1-x)$, reaching a total of $N(2x - x^2) > Nx$. Thus, this mode increases the average visibility.

Highest Span Node Exchange: The reason why reducing visibility incurs higher NOR lies in that some high span nodes are not visible to all servers. To overcome this problem, we use another heuristic, namely, nodes exchange information of the node with the highest span among their known neighbors. When a node sends out its own span values, it attaches the highest span node it knows. When a node receives a span report and finds that the attached high span node is not known to itself, the node is added to its known server list. This technique does not increase the number of nodes each node knows by much, but gives nodes very important information for their replica selection process.

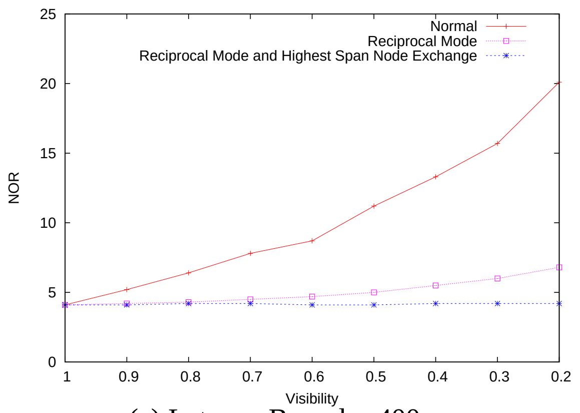
Figure 11 investigates the impact of reducing visibility on the NOR and the effectiveness of our heuristics in keeping the NOR small despite low visibility. For each latency bound and visibility value, we run our algorithms 10



(a) Latency Bound = 200ms

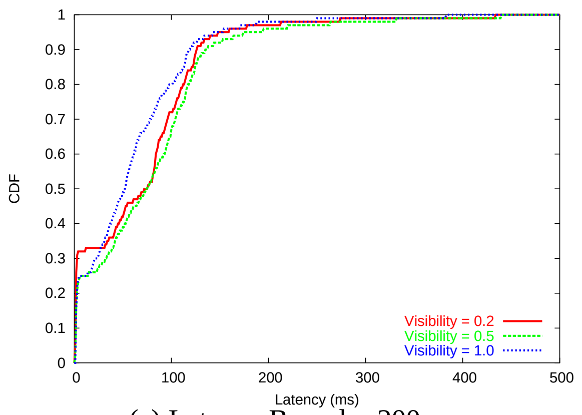


(b) Latency Bound = 300ms

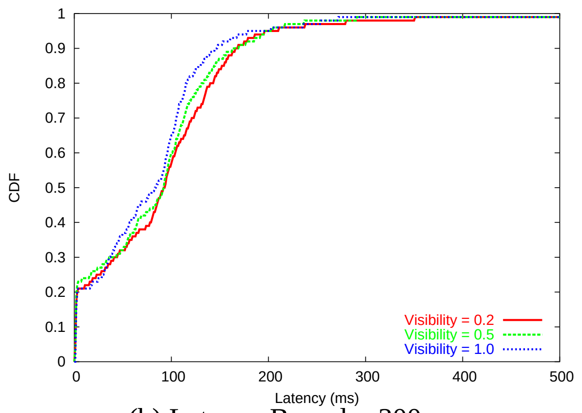


(c) Latency Bound = 400ms

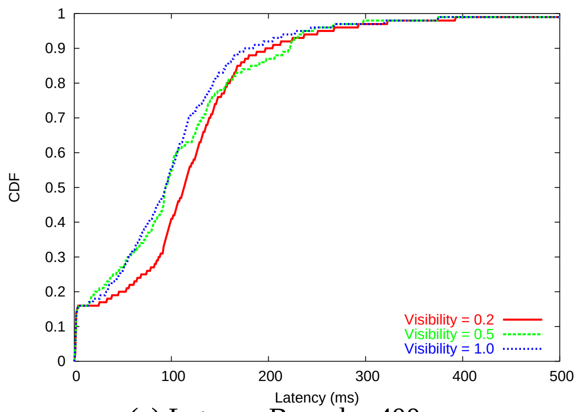
Figure 11. NOR, Changing Visibility



(a) Latency Bound = 200ms



(b) Latency Bound = 300ms



(c) Latency Bound = 400ms

Figure 12. Client Perceived Latencies with Imperfect Visibility

times and take the average NOR. As shown in Figure 11, for all visibility values, the NOR of the system decreases with the increase in the latency bound. For all latency bound values, NOR increases as visibility decreases. Using the reciprocal mode greatly reduces NOR when visibility is above 0.5. An interesting observation from Figure 11 is that the system in the reciprocal mode with x as the initial visibility performs much better, in terms of minimizing

NOR, than a system that simply drops packets from unknown servers, but has $2x - x^2$ as its initial visibility. This is true even although the average numbers of nodes each node ends up knowing about are the same in both cases. For example, the NOR at an initial visibility of 0.5 in the reciprocal mode is less than that at a visibility of 0.8 in a system that drops unknown server packets. The reason lies in the fact that in the reciprocal mode the peer knowledge relationships are bi-directional after the initial introduction, while that is not true in the second case.

When visibility is below 0.4, the reciprocal mode alone is not sufficient. However, using both this mode and the highest span node exchange heuristics can further reduce NOR. From Figure 11, we can tell that when both heuristics are used, the increase in NOR is negligible even when visibility is as low as 0.2, when latency bounds of 300ms and 400ms are requested. When the latency bound is very tight (200ms), the NOR increases somewhat more appreciably with decreased visibility. This is because when the latency bound is loose, more nodes have a high span and there is a better chance of finding a small number of replicas that dominate the network. When the latency bound is rigid, there are fewer choices for the dominating set.

In principle, imperfect visibility may also affect the latency bound conformity ratio because having low visibility makes it harder to find replicas that have low latencies. To show the effect of imperfect visibility on meeting latency bound guarantees, we ran three concurrent experiments configured with visibility values of 0.2, 0.5 and 1.0 respectively. Three processes were created on each server, one on behalf of each experiment. These processes ran our distributed replica selection algorithm with both the reciprocal mode and the highest span node exchange heuristics enabled. The CDF of client perceived latencies of the whole network is plotted in Figure 12 in each case. Observe that in all cases the success ratio in meeting the bound remains very high, differing in less than 1% across visibility values. This result is very encouraging.

When latency bounds are more relaxed (Figures 12b and 12c), the system yields the same NOR's for all three visibility values. Average latencies increase slightly with reduced visibility. In the case where the bound is tight (200 ms), shown in Figure 12a, the NOR increases slightly with decreased visibility. An interesting consequence is that the average latency actually improves over that at a higher visibility as evident by comparing the latency curves at visibility values 0.2 and 0.5.

Based on the analysis and experiments above, we conclude that perfect visibility is not needed in our system. Our system's performance in terms of minimizing NOR and achieving a high confidence in meeting the latency bound are insensitive to the visibility parameter.

Finally observe that update traffic occurs when hosted content objects are not completely static but may change on the origin servers. The updated objects need to be propagated to all their replicas so that consistency is maintained. In our system, when an object is updated on the server, an update is propagated to all its replicas via application layer multicast. Without going into the details of this mechanism, observe that the total number of update messages sent in the application layer multicast tree remains the same independently of tree topology. This is because an update message must be sent exactly once to each replica. The only difference between the topologies lies in the source of each message. If the backbone provider charges the CDN provider per bit regardless of path length, the cost of updates is proportional to the number of replicas regardless of the topology. Since imposing more stringent delay bounds leads to more replicas, a provider can easily quantify the incremental cost of tightening the delay bound from the resulting increase in the number of replicas together with assumptions on the average update rate and size. This analysis can be used to determine the cost of QoS contracts.

7 Related Work

There are many theoretic results on graph domination problem and its variants. The three distributed algorithms [21, 12, 19] we studied are the most related ones to our problem. In this paper, we gave our own distributed algorithm which is very simple yet performs well in a highly asynchronous environment and terminates faster than previous algorithms.

The efficacy of content distribution depends primarily on the placement of replicas. While the replica placement problem in CDNs has been extensively studied in previous literature [20, 25, 11, 15, 27], the placement objectives have mostly been to optimize some *average* performance metric such as client-perceived latency, number of ASes traversed, or some notion of cost of link traversal.

Li et al. [20] simplified Internet topology as a tree and developed an optimal algorithm based on that topology. However, the authors did not evaluate the performance of their dynamic programming algorithm for a realistic Internet topology. Besides, the algorithm has a relatively high computational complexity $O(N^3M^2)$.

Qiu et al. [25] formulated the replica placement problem as that of choosing a fixed number of replicas among a given set of locations to minimize the overall client request latencies. This problem is mapped to a K-median problem in graph theory which is NP-Hard. The authors assumed that each individual CDN server has unlimited disk capacity and therefore each client uses a *single* replica. Hence, their replica granularity is very coarse. Some heuristics were developed and compared using simulations. A greedy algorithm was shown to perform very well in practice and to be relatively insensitive to imperfect input data.

Kangasharju et al. explicitly took storage capacity of each individual CDN server into account and considered each AS as a node in a graph representing one CDN server [15]. The problem of optimizing average number of ASes traversed for client requests was formulated as a combinatorial optimization problem.

The study by Jamin et al. [11] investigated the impact of the number of replicas on the performance of various replica placement methods. Their major finding was that increasing the number of replicas is effective in reducing client latencies and reducing server load only for a small number of replicas, regardless of the placement algorithm. One algorithm studied in [11], called *Transit Node*, has a similar flavor to our replica selection algorithm. The transit node algorithm assumes that the node with the highest degree can reach more nodes within a smaller latency. The algorithm therefore places replicas on candidate hosts in descending order of degree. Note that the concept of a host's degree referred to in the transit node algorithm is roughly the outdegree of the AS the host belongs to. Hence, the degree of a server is rather static and does not change with network traffic conditions. This is different from the degree concept in our distributed replica selection algorithm which is tightly related to network and system load conditions. A later work [27] further evaluated a router fanout-based replica placement algorithm and found that with careful design, the router-level fanout-based placement algorithm is almost as good as a greedy algorithm in [25] in most cases.

Venkataramani et al. [36] studied the replica placement problem of minimizing overall client access time under servers' bandwidth constraints. The algorithm approaches the placement problem by hierarchically refining an initial per-server greedy algorithm. This work used a simple hierarchical model: all cache/CDN servers are leaf nodes of the hierarchy tree. Intermediate nodes are dedicated for maintaining book-keeping information.

The above algorithms all share a same limitation, namely that they are all offline centralized algorithms. In a recent study [16] on general replica placement algorithms, a unified framework was designed to classify them. According to their findings, most existing algorithms for CDNs such as algorithms in [25, 11, 15] do not scale for systems with more than 10^4 nodes because of the large number of content objects a CDN could host. They identified decentralized algorithms as an important direction for replica placement in CDNs. Our algorithm, being a purely distributed scheme, falls into this category, and can thus address scalability issues of existing replica placement algorithms.

SCAN [5] approaches the replica placement problem by minimizing the number of replicas while meeting client latency and server capacity constraints. While it has the same objective as our algorithm, SCAN has a totally different context and solution. SCAN utilizes Tapestry [39] as the underlying distributed object routing and location system and proposes a dynamic replica placement algorithm. With the support of Tapestry, SCAN can trace back to the root of a content via a self-organizing application-level multicast tree to find a server that meets the client perceived latency requirement and server resource constraints to be the replica.

Rabinovich et al. [26] proposed a protocol suite for dynamic replication and migration of Internet objects. It has algorithms for deciding on the number and placement of replicas and an algorithm for distributing requests among available replicas. However their work does not address QoS guarantee issues.

Ko and Rubenstein [17] investigated an abstracted problem of placing replicas of different types of resources in large-scale network systems such as P2P networks and wireless sensor networks. They considered a different network model from ours in which each node must hold some resource. The placement is to achieve that any resource is reachable over a short path from any point in the network. The authors presented a decentralized, self-stabilizing algorithm in which each node continually change the resource type it holds to maximize its own distance to a node that has the same type of resource.

8 Conclusions

In this paper, we presented a content distribution network (CDN) designed to provide latency bounds on content access and gave an extensive performance evaluation. We approached the delay guarantee problem by dynamically selecting replicas using a distributed algorithm that attempts to meet global latency bounds. The algorithm formulates the replica selection problem as a graph domination problem. Performance of our distributed replica selection algorithm in minimizing the overhead, the number of replicas, and meeting the delay bounds is evaluated and compared with other distributed graph domination algorithms as well as algorithms for latency minimization on an actual implementation on PlanetLab. Our results show that our system can achieve the needed latency with a very high confidence and at a very limited content replication cost. Being completely decentralized, our system enjoys the ability to adapt to various network and server condition changes and has very good scalability. For future work, we are interested in extending the performance evaluation with real traces of client traffic as opposed to synthetic workload. We are also interested in providing more mechanisms to tune the trade-offs between meeting latency bounds and minimizing the number of replicas, as well as to allow the confidence in the latency guarantee to be a tunable parameter.

References

[1] Akamai. http://www.akamai.com.

[2] A. Barbir, B. Cain, R. Nair, and O. Spatscheck. Known content network (CN) request-routing mechanisms, July 2003.

[3] V. Cardellini, M. Colajanni, and P. S. Yu. Request redirection algorithms for distributed web systems. *IEEE Transactions on Parallel and Distributed Systems*, April 2003.

[4] R. L. Carter and M. E. Crovella. Server selection using bandwidth probing in wide-area networks. In *Proceedings of IEEE INFOCOM'97*, 1997.

[5] Y. Chen, R. H. Katz, and J. D. Kubiatowicz. Scan: A dynamic, scalable, and efficient content distribution network. In *Proceedings of The International Conference on Pervasive Computing (Pervasive'02)*, August 2002.

[6] V. Chvátal. A greedy heuristic for the set-covering problem. In *Mathematics of Operations Research*, 1979.

[7] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of SIGMETRICS'96*, May 1996.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.

[9] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE JSAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, August 2003.

[10] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *Proceedings of ACM SIGCOMM'02*, August 2002.

[11] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. In *Proceedings of IEEE INFOCOM'01*, April 2001.

[12] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. In *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC'01)*, August 2001.

[13] D. S. Johnson. Approximation algorithms for combinational problems. In *Journal of Computer and System Sciences*, 1974.

[14] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek. The measured performance of content distribution networks. In *Proceedings of the 5th International Web Caching Workshop and Content Delivery Workshop (WCW'00)*, May 2000.

[15] J. Kangasharju, J. Roberts, and K. W. Ross. Object replication strategies in content distribution networks. In *Proceedings of The 6th International Web Caching Workshop and Content Delivery Workshop (WCW'01)*, Boston, MA, June 2001.

[16] M. Karlsson, C. Karamanolis, and M. Mahalingam. A framework for evaluating replica placement algorithms. Technical Report HPL-2002-219, HP Labs, 2002.

[17] B.-J. Ko and D. Rubenstein. Distributed, self-stabilizing placement of replicated resources in emerging networks. In *Proceedings of the 11th International Conference on Network Protocols (ICNP'03)*, November 2003.

[18] B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *Proceedings ACM SIGCOMM Internet Measurement Workshop 2001*, November 2001.

[19] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC'03)*, July 2003.

[20] B. Li, M. Golin, G. Italiano, , and K. Sohraby. On the optimal placement of web proxies in the Internet. In *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.

[21] B. Liang and Z. J. Hass. Virtual backbone generation and maintenance in ad hoc network mobility management. In *Proceedings of IEEE INFOCOM'00*, March 2000.

[22] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance. In *Proceedings of The First Workshop on Internet Server Performance*, June 1998.

[23] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, and R. Tewari. Cooperative leases: scalable consistency maintenance in content distribution networks. In *Proceedings of International World Wide Web Conference (WWW'02)*, 2002.

[24] PlanetLab. http://www.planet-lab.org.

[25] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *Proceedings of IEEE INFOCOM'01*, April 2001.

[26] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A dynamic object replication and migration protocol for an Internet hosting service. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS'99)*, 1999.

[27] P. Radoslavov, R. Govindan, and D. Estrin. Topology-informed Internet replica placement. In *Proceedings of The 6th International Web Caching Workshop and Content Delivery Workshop (WCW'01)*, Boston, MA, June 2001.

[28] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM'01*, 2001.

[29] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001.

[30] S. Saroiu, K. P. Gummadi, R. Dunn, S. D. Gribble, and H. M. Levy. An analysis of Internet content delivery systems. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, December 2002.

[31] Spring IP Monitoring Project. http://ipmon.sprint.com.

[32] Spring IPMon Delay Analysis. http://ipmon.sprint.com/delaystat.

[33] Squid Web Cache. http://www.squid-cache.org.

[34] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM'01*, Aug 2001.

[35] J. Strauss, N. Katabi, and M. F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of Internet Measurement Conference (IMC) 2003*, October 2003.

[36] A. Venkataramani, P. Weidmann, and M. Dahlin. Bandwidth constrained placement in a WAN. In *Proceedings of ACM Principles of Distributed Computing (PODC'01)*, August 2001.

[37] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on CDN robustness. In *Proceedings of The Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, December 2002.

[38] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering server-driven consistency for large scale dynamic web services. In *Proceedings of International World Wide Web Conference (WWW'01)*, 2001.

[39] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.