# Issues in Web Content Replication

Michael Rabinovich
AT&T Labs
180 Park Avenue
Florham Park, NJ 07932
misha@research.att.com

**Abstract**

*Web content replication began as explicit manual mirroring of Web sites. It has since evolved into user-transparent request distribution among a static set of mirrors. The next step (especially important for large-scale Web hosting service providers) is dynamic content replication, where object replicas are created, deleted, or migrated among Web hosts in response to changing usage patterns. This paper examines issues that arise in Web content replication, paying special attention to challenges in dynamic replication.*

## 1 Introduction

As commercial interest to the Internet continues to grow, the issues of scalability and performance become increasingly important. According to UUNET, the largest Internet Service Provider (ISP), its network traffic has been doubling every 100 days [12]. In fact, we are on the verge of another qualitative jump in Internet load levels, due to the upcoming replacement of slow modem lines, which act as floodgates limiting user access to the Internet, with much faster alternatives like ISDN lines and cable modems. It is unlikely that the brute-force approach of adding ever increasing network and server capacities would solve the Web scalability problem in a foreseeable future. Consequently, caching and replication, being the primary tools that address these issues, are fast becoming a focus of attention in both industrial and academic research communities. This paper examines issues in replication on the Internet and outlines research directions that are both interesting and practically important.

One can distinguish two main environments where replication can be used. One is a stand-alone replicated Web site; the other is a platform for a large-scale hosting service that hosts Web objects from third-party information providers. In the limited-scale stand-alone environment, a brute-force worst case resource allocation can be feasible. Manual monitoring of demand and decisions on the number and placement of replicas is feasible as well. Thus, a stand-alone Web site can often use static replication (or *mirroring* in the Internet parlance).

In the hosting service environment, the waste of the worst-case design would be multiplied by potentially millions of hosted Web sites; the decision space for replica placement could also become unmanageable. Thus, dynamic automated replication is needed. According to Probe Research [14], the market for hosting services will reach $2.5 billion by 2001. This gives an extra imperative to develop appropriate technologies for automatic dynamic replication of Web objects.


**Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**

In general, there are two fundamental questions any replicated system must deal with. One is assigning requests to replicas. The other is deciding on the number and placement of replicas for all objects. These issues are often studied in isolation, as a server selection problem and a file allocation problem. However, they strongly influence one another, and in a system with dynamic replication must be studied together.

## 2 Caching vs. Replication

Being an immature field, the Web has created many terms to denote similar things. In particular, there is some confusion between caching and replication. To develop an intuition for the difference between the two, I propose the following distinction.

Caching is storing an object at a place that sees the object anyway in the course of request processing. Examples include browser cache, proxy cache, server main memory cache. Replication is storing an object at a place that would not otherwise see the object. Web site mirroring provides an example of replication.

Alternatively, one can say that the difference between a cache and a replica is that a cache routinely sees both hit and miss requests, while a replica normally sees only hits (except when a request arrives for an object that has been deleted). In other words, requests flow to the cache regardless of the cache contents, while requests arrive at a replicated server only if that server is believed to have a replica of the requested object.[1]

While useful and necessary, caching itself cannot solve the scalability problem of the Web. Indeed, many objects are not cacheable but replicable. These include dynamic objects with read-only access and personalized objects ("cookied" requests). In addition, many objects that are updated as a result of accesses are still replicable because updates commute. A typical example is provided by so called hit metering, where an object maintains a counter of times it has been accessed.

Another advantage of replication is that it is under full control of the service provider. Thus, issues like copyright enforcement, hit metering, data consistency are much easier. Reluctance to deal with these issues leads sometimes to "cache busting" by content providers, who use various available mechanisms to prevent caching of their pages. Finally, replicated servers are free to use proprietary protocols that would be more efficient for moving object replicas between servers or maintaining them consistent, while caches are confined to standards.

## 3 Mechanisms for Web Request Redirection

Transparent replication entails redirecting a client's request for a logical object to one of the physical object replicas. To be grounded in reality, any proposal for replication on the Web, must explain the assumed mechanism for such redirection. Understanding this issue lets one distinguish between approaches that are deployable in practice from more speculative research ideas (this is not to say that the latter are not valuable).

Accessing a Web resource, e.g., `www.foo.com/home.html`, involve the following main steps.

1. The client sends the domain name of the Web site (`www.foo.com`) to its local domain name *resolver*. The resolver obtains the corresponding IP address by contacting a well-known root of the domain name server (DNS) hierarchy and ultimately querying foo's DNS server. The resolver then returns this IP address to the client.

2. The client sends the request for `home.html` to the received IP address. This request follows a router path on the Internet until it arrives at the Web server, which sends back the requested page.

Let us consider points on the request processing path where the request can be transparently redirected to one of several replicas of the object, and the tradeoffs involved. (We will not consider non-transparent replication where users are asked to choose among multiple URLs of the object.)

[1] Note that replication in the above sense is often called "push caching" or active caching. I find these terms confusing and will not use them here.

## 3.1 Client Multiplexing

In this approach, the client (Web browser or a proxy server) obtains a set of physical replicas and chooses the one to send its request to. Three main mechanisms fall into this category.

In the first approach, the DNS server of the service provider returns the IP addresses of servers holding a replica of the object. The client's DNS resolver chooses a server among these. To decide, the resolver may issue probes to the servers and choose based on response times to these probes, or it may collect reports from the the client on performance of passed accesses to these servers. This approach is used by Bhattacharjee et al [5] and by Beck and Moore [4]. Its advantage is that no extra communication is added to the critical path of request processing. There are also several shortcomings. First, the approach relies on clients using a customized DNS resolver. If this resolver relies on client performance reports, then the client (i.e., browser or proxy) software must be modified as well. Second, the DNS infrastructure relies heavily on DNS response caching to cope with its load. Therefore, replica sets cannot be changed frequently without the danger of resolvers using stale replica sets. At the same time, reducing the caching time may just move the scalability bottleneck to the DNS infrastructure, provided enough clients adopt the approach. Thus, this approach is most appropriate for static replication.

The second approach relies on Java applets to perform replica selection on the client [21]. The URL of an object actually points to a Java applet, which embeds the knowledge about the current replica set and the procedure for replica selection. This approach requires no changes to clients. However, unlike the previous approach, it involves an extra TCP communication to download the applet.

The third approach, proposed by Baentsch et al [3], propagates information about replica sets in HTTP headers. It requires changes to both Web servers and clients (proxy servers in this case) to process extra headers. Clients must also be modified to implement replica selection.

## 3.2 IP Multiplexing

A plethora of commercial products offers multiplexing routers. A now-outdated survey can be found at [9]. In this approach, a special multiplexing router (or a multiplexor) is placed in front of a server farm. The domain name of the Web site is mapped to the IP address of this router, so all clients send their packets to it. When the first packet from a client arrives, the multiplexor selects a server in the farm and forwards the packet (and all subsequent packets in the same session) to this server. The multiplexor also intercepts the response packets and modifies their headers to contain its IP address rather than that of the server. The multiplexor maintains a *sessions database*, which records a server chosen for each active communication session. The sessions database is needed to ensure that, once a server is chosen for servicing a request, all packets from this client in this communication session go the same server.

This approach uses standard clients and DNS and Web servers. However, by having a focal communication point, it does not scale geographically. Thus, it is mostly used for load sharing in a server farm on a local-area network.

An interesting twist on this approach is provided by IBM Network Dispatcher [11]. Its multiplexor selects servers as before. But servers respond to clients directly, bypassing the multiplexor and using the multiplexor's IP address as the sender address. Since most of the data flows from servers to clients, this provides better geographical scalability, especially from the bandwidth consumption perspective. The latency gain is limited by the need for client acknowledgements to travel via the multiplexor.

## 3.3 DNS Indirection

Several domain name server implementations (e.g., the one used in CISCO Distributed Director [6]) allow the Web site's DNS server to map a host domain name to a set of IP addresses and choose one of them for every client query, based on such factors as the query origin and the load of server replicas. The difference with DNS-based client multiplexing is that choosing a replica occurs at the Web site, not at the client's DNS resolver. This approach can scale well geographically. Unfortunately, DNS response caching by clients complicates changing replica sets and controlling request distribution among replicas. At the same time, reducing the lifetime of cached DNS responses may shift the performance bottleneck to the DNS infrastructure.

In general, DNS system was designed for mostly an append-only database existing mappings between a host name and an IP address rarely ever changes. In dynamic replication schemes, however, an existing mapping changes with every change in the set of replicas for a given object.

### 3.4 HTTP Redirection

HTTP protocols allows a Web server to respond to a client request with a special message that tells the client to resubmit its request to another server. This mechanism can be used to build a special Web server which accepts client requests, chooses replicas for them and redirects clients to these replicas. Commercially, CISCO Distributed Director [6] and WindDance Web Challenger [19] implement this functionality.

An advantage of HTTP-level redirection is that replication can be managed at fine granularity, down to individual Web pages, whereas other mechanisms postulate the entire Web site as the granule. A disadvantage is that this mechanism is quite heavy-weight. Not only does it introduce an extra message round-trip into request processing, but also this round-trip is done over HTTP, which uses the expensive TCP protocol as the transport layer.

### 3.5 Anycast

In the next generation of the Internet routing protocol (IPv6), an *anycast* service [13] will be supported. Without getting into the details of the mechanism, this service assumes that the same IP address is assigned to a set of hosts, and each IP router has in its routing table a path to the host that is the closest to this router. Thus, different IP routers have paths to different hosts with the same IP address.

There is virtually no overhead for request indirection with anycast, since packets travel along router paths anyway. However, it assumes very static replica sets (since a change in the replica set would take long time to reflect in router tables throughout the Internet) and a rigid request distribution (because all requests from a given client normally go to the same host, chosen based on the network proximity). Advances in research on *active networks* [17], where applications can inject ad hoc programs into network routers, may alleviate the last limitation.

## 4 Research Issues - Architecture

We have seen that most existing architectures for request indirection lack the flexibility needed for dynamic replication. HTTP-based redirection might be suitable, but seems too heavy-weight for most cases. Thus, supporting dynamic replication requires a special highly updateable name service. Such a *Web name service* would maintain a *mapping database* that records logical object names and the names of their corresponding physical replicas, perhaps along with some prior replica usage or performance statistics. The Web name service would use this information to resolve requests for logical objects into requests to object replicas.

An architecture of such service is an open issue. There are a variety of questions to be resolved. The first question is whether the intended use is to be an Internet-wide service or specific to particular sites or hosting platforms. A general Internet-wide name service could be a logical outgrowth of the Uniform Resource Name (URN) proposal whose goals included adding location independence to the Web naming scheme [10]. Unfortunately, the URN effort was too ambitious for any consensus to emerge, and it has not caught up so far. Reviving a slimmed-down URN effort, that would concentrate only on location management, would be beneficial for the Web. A name service for an individual Web site would have very different architecture than a name service for a hosting platform, due to the differences in scale and perspective (e.g., backbone traffic reduction is a major concern for a hosting platform but is of peripheral interest to a Web site).

To be a little more specific, let us consider a hosting service provided by an Internet Service Provider (ISP). The main components of such a system are Web hosting servers (hosts for short), the backbone, and gateway routers through which the platform communicates with the rest of the Internet and with the subscribers of the ISP. Since gateway routers see all client requests anyway, they are a convenient place to redirect requests to various hosts that currently have a replica of requested objects. Redirection at these points could be achieved by co-locating Web name servers on the some LAN with the gateway routers. Requests could be diverted to these name servers by resolving the DNS name of any hosted Web site to the IP address of the name server that is the closest to the requester. Alternatively, it could be done by anycast when available. Note that we cannot rely on the gateway routers themselves to divert requests to their local Web name servers, since there is no guarantee that all packets comprising the request will come to the same gateway router.

Many important issues need be resolved within this framework. Since we have a replica of the name server co-located with every gateway router, the number of name server replicas may be high. However, the frequency of updates to the name server database limits the number of replicas, since updating a large number of replicas is impractical. In fact, some request distribution algorithms (e.g., round-robin) rely on prior replica usage information, which they record in the mapping database and update on every access. This would make name server replication even more problematic. A possible approach to resolving this dilemma is outlined in [16].

Another important decision is to choose between *iterative* and *recursive* request processing. A name server can either return the physical replica ID to the client and let the client access the replica directly, or it can obtain and forward the document to the client. The first alternative (which we call iterative, in analogy with the DNS terminology), reduces the load on the name server and allows clients to cache name resolutions for future use. The second approach, called recursive, hides the identity (and even existence) of hosts from the clients and eliminates an extra message exchange with the client.

One more area of interest is caching within this architecture. In the iterative request processing approach, clients might cache name resolutions for future use. What should be the policies for caching name resolutions? How to enforce these policies? What is the influence of such caching on request distribution and replica placement decisions? Also, one could further reduce backbone traffic by caching especially hot documents at the name servers, at the expense of increasing their load. What are the right policies in this respect?

# 5 Research Issues - Algorithms

As already mentioned, every replicated system must deal with two fundamental issues - distributing requests to object replicas and deciding on the number and placement of replicas.

## 5.1 Request Distribution

We already discussed architectural issues in supporting request distribution in Section 4. Let us now consider algorithms these architectures might run. One can distinguish two main classes of algorithms for request distribution - feedback and non-feedback algorithms [1]. Feedback algorithms rely on some aggregate metrics obtained from the hosts, e.g., in the form of load reports. Non-feedback algorithms use only information that name servers obtain from processing prior requests, such as the number of times they assigned previous requests to various replicas. Examples of non-feedback algorithms include the round-robin request distribution or always choosing the closest replica for each request.

Feedback algorithms in general can achieve better request distribution. For instance, a feedback algorithm can take a weighted average of server load and the client-server network distance as a criterion for choosing the replica for a request. On the other hand, feedback schemes make it difficult to predict the effects of an object migration or replication on request distribution. Indeed, request distribution for an object becomes dependent on the popularity of many other objects co-located at the same servers, so moving one object may affect request distribution for other objects and other servers.

Non-feedback algorithms distribute requests "blindly" and are necessarily suboptimal. At the same time, they de-couple distribution of requests to different objects and make estimating the effects of object relocations feasible. As a simple example, creating a second replica of an object in the round-robin scheme will reduce the request rate for the existing replica by half, leaving the request rate for all other objects the same.

Of course, round-robin and always-the-closest are too simplistic algorithms: the former is oblivious to network proximity and is not useful in reducing bandwidth consumption; the latter disregards the load and will continue to send all requests to an overloaded server no matter how many additional replicas are created, if the requests come from nearby clients. A more sophisticated non-feedback algorithm can combine the rank of replicas in the round-robin order with the proximity rank to account for both server proximity and load, while retaining the ability to predict the effects of replication. We described one such algorithm in [16]. However, much more work needs to be done. In fact, it appears that the best choice is to use a feedback algorithm when the overall load in the system is low and switch to non-feedback during high-load periods [1].

### 5.2 Replica Placement

Another important problem is deciding how many replicas of various objects to have and where in the network to place them. We call this a *replica placement* problem. There are number of issues to be resolved.

The most important issue to resolve is the choice between static and dynamic replica placement. Static replica placement assumes very coarse-granularity decisions that typically involve a systems administrator, based on the observed access characteristics. With dynamic replica placement, the system constantly monitors access to various resources and continuously adjusts replica sets for all objects. The choice between these two approaches is not that obvious. Static placement imposes less stringent requirements for the request distribution algorithm, because the latter does not have to allow predicting effects of individual object replication or migration. So, a more efficient request distribution algorithm can be used, which, for the same assignment of replicas to hosts, would provide better traffic and latency reduction. On the other hand, static placement is error-prone, and can become infeasible for a large-scale system. Moreover, it does not adjust to changes in access patterns or environment (e.g., Internet topology). Overall, static placement may be a feasible solution for a large stand-alone Web site, while dynamic placement seems essential for a *hosting service*, a platform that hosts Web resources from third party content providers. Indeed, a scale of a hosting platform can potentially make worst-case resource allocation of static approaches too wasteful, and the search space for placing replicas too large for a static solution to be feasible.

Within the static approach, the main issue becomes choosing a good request distribution algorithm (usually referred to as server selection in this context), which in this case would most likely be feedback-based. Server selection can use many heuristics, sometimes based on contradicting metrics. The problem here is choosing good metrics and combining them to arrive at a compromise strategy for server selection. We elaborate more on this issue in Section 5.3.

For the dynamic approach, request distribution must be considered in combination with the algorithm for replica placement. The combination must consider load and proximity factors and provide acceptable *responsiveness* in adjusting to changes in access patterns. The responsiveness can only be achieved if objects are relocated *en masse*, without waiting to observe effects after each move. This entails being able to predict (or at least bound) the effects of individual object migrations or replications on request distribution, so that placement decisions can be made for multiple objects at once, without waiting to observe the effects of each move.

Another issue related to dynamic replica placement is the *granularity* of replication. At the finest granularity, the placement of each Web page can be considered separately. The other extreme is to consider placement of entire Web sites. Various intermediate object groupings are also possible. Finer granularity imposes higher overhead for collecting and maintaining access statistics and for placement decisions. Coarser granularity may increase the overhead for transferring objects between sites. In some cases of dynamically generated pages, a group of files, executables, and other environment state *must* be migrated together, since they are used together for page generation. The cost of tranferring such bundled objects may be high and should be taken into account in a replica placement algorithm. A language for specifying these bundled objects must be designed. Marimba's DRP protocol is an example of work in this direction [8]. In addition to manual bundling of objects, some objects could conceivably be bundled together automatically based on similarity of access patterns. The policies for (and the feasibility of) doing this is another issue for future research.

There is also an issue of consistency between replica sets and name service database which maps logical object IDs to sets of physical replicas. Changes in replica sets must be reflected in the name service. If the mapping database of the name service does not agree with actual replica sets, requests may be assigned to non-existent object replicas. On the other hand, migrating an object and updating the mapping database within a distributed transaction would interfere with processing client requests by the name service, and is therefore undesirable. A better approach is to ensure that the replica set specified in the mapping database be always a subset of the actual replica set. In other words, when a replica is to be deleted, it is first deleted from the mapping database, and only then physically removed from the hosting server. When a replica is to be created, it is added to the mapping database only after being successfully created by the hosting server. Migration can be treated as replica creation followed by replica deletion.

An additional complication is that the name service might have previously directed some requests to the replica being deleted, and these requests have not yet reached the replica. This situation is especially dangerous iterative-style name service, with potentially longer delay between request assignment and request arrival at the replica. A protocol must be provided for tracking the number of outstanding requests to each physical replica of each object and for ensuring that a replica is physically deleted only after all such requests have been serviced.

## 5.3 Replica Ranking

Request distribution algorithms rely on some metrics to discriminate among replicas. Numerous different metrics have been proposed. Among them, there are Web server load, the latency of a ping message, the response time of prior Web document fetches, the latency of prior Web document fetches (the difference being that the latter measures time until the first portion of the data arrives), the network distance (in hops), the geographical distance (in miles). Some of these metrics postulate a place where request redirection takes place. For instance, latency and response time metrics depend on network congestion between a host and the requesting client. Thus, they make sense for a point that lies on the path from the host to the client. As an example, the DNS server of a Web site of hosting service cannot use ping latencies for choosing replicas because these latencies will measure network congestion between replicas and the DNS server, not between replicas and the requesting client. Other metrics, such as host load and distance can be used irrespective of the indirection point. Some metrics, like server load, require a feedback-based algorithm and complicate prediction of effects of changing a replica set on request distribution [1]. Finally, in most cases multiple metrics must be combined in some way, e.g., network distance and server load, in ranking replicas. Choosing a proper metric influences the architecture and algorithm choices and has a profound effect on performance.

## 5.4 Replica Consistency

This is the area where much synergy exists with existing database research. All Web objects can be divided into the following types:

1. Objects that do not change as the result of user access. These objects can be either static HTML pages, or dynamic pages that retrieve information, i.e., weather reports, or map drawing services.

2. Objects in which the only per-access modification involves collecting access statistics or performing other commuting updates.

3. Objects that be updated as a result of user accesses, and where these updates do not commute.

A majority of Web accesses are to static objects, which can only be modified by their authors but not user accesses. The rate of author modifications is negligible compared to the access rate, so it should not play role in replica placement decisions. Replica consistency can be maintained by using the primary copy approach. Depending on the needs of the application, updates can propagate from the primary asynchronously to the rest of currently existing replicas either immediately or in batches using epidemic mechanisms [7]. There are many ideas in epidemic replication research that are directly applicable to this application, such as determining efficiently which pages need to be copied [15], ensuring the consistency of propagation of multi-page updates [2], and that any given client sees monotonically increasing object versions [18]. One technical issue that must be resolved is how the primary copy is chosen, and what guarantees (if any) are provided to avoid choosing multiple primaries by multiple updaters.

Objects in the second category can still be freely replicated if a mechanism is provided for merging updates recorded by different replicas. The problem arises if content served to clients includes updateable fields, as is the case of hit counts appearing in some pages. If the application requires this information to be always current, then such objects become equivalent to objects in the third category for the purpose of replication.

Objects in the third category can be freely migrated, but their replication strategy must depend on the mix of read-only and update accesses. While some ideas from database research are clearly relevant to this problem (e.g., [20]), Web specifics require more work in this area.

# 6 Conclusions

This paper describes design space and research directions for content replication on the Web. Content replication is extremely important for Web scalability. Most work to date has concentrated on server selection algorithms assuming a fixed replica set. While much work still remains to be done in this area, the next big step is exploring dynamic replication and migration schemes.

Dynamic replication will be especially important for providers of *hosting services*, which maintain platforms for hosting Web content owned by third parties. As the scale of these systems grows, static manual mirroring will become increasingly wasteful and intractable. Consequently, dynamic replication will be one of enabling technologies for these systems. A plethora of issues must be resolved in the dynamic replication context, including request distribution and replica placement algorithms, granularity of replication, combining replica rankings according to various metrics, and consistency of control information in the system.

# Acknowledgements

I wish to thank Theodore Johnson and H. V. Jagadish for reading a draft of this article and their useful comments.

# References

[1] A. Aggarwal and M. Rabinovich. Performance of replication schemes on the Internet. Technical Report HA6177000-981030-01-TM, AT&T Labs, October 1998. Also available as http://www.research.att.com/∼misha/radar/tm-perf.ps.gz.

[2] D. Agrawal, A. El Abbadi, and R. C. Steinke. Epidemic algorithms in replicated databases. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '97)*, pages 161–172, May 1997.

[3] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. Enhancing the web infrastructure – from caching to replication. *IEEE Internet Computing*, pages 18–27, Mar-Apr 1997. Also available at http://neumann.computer.org/ic/books/ic1997/pdf/w2018.pdf.

[4] Micah Beck and Terry Moore. The Internet-2 distributed storage infrastructure project: An architecture for internet content channels. In *3rd Int. WWW Caching Workshop*, Manchester, UK, June 1998. Available at http://wwwcache.ja.net/events/workshop/18/mbeck2.html.

[5] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, and Z. Fei. Application-layer anycasting. In *INFOCOM*, 1997.

[6] Cisco Systems, Inc. DistributedDirector. White paper. http://www.cisco.com/warp/public/734/distdir/dd_wp.htm.

[7] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, August 1987.

[8] The http distribution and replication protocol. A WWW Consortium submission, August 1997. http://www.w3.org/TR/NOTE-drp.

[9] R. Farrell. Distributing the web load. *Network World*, pages 57–60, September 22 1997.

[10] Lewis Girod and Karen R. Sollins. Requirements for URN resolution systems. Internet Draft. http://ana-www.lcs.mit.edu/internet-drafts/draft-girod-urn-res-require-00.txt, June 1996.

[11] IBM Interactive Network Dispatcher. http://www.ics.raleigh.ibm.com/netdispatch/.

[12] Large scale network caches provide more bandwidth for your money. Inktomi Corp. White Paper, 1998. http://www.inktomi.com/products/traffic/tech/economics.html.

[13] C. Partridge, T. Mendez, and W. Milliken. RFC 1546: Host anycasting service, November 1993.

[14] Probe Research, Inc. http://www.proberesearch.com.

[15] M. Rabinovich, N. Gehani, and A. Kononov. Scalable update propagation in epidemic replicated databases. *Lecture Notes in Computer Science*, 1057:207–222, 1996. Proc. of EDBT'96.

[16] M. Rabinovich, I. Rabinovich, and R. Rajaraman. Dynamic replication on the Internet. Technical Report HA6177000-980305-01-TM, AT&T Labs, March 1998. Also available as http://www.research.att.com/~misha/radar/tm.ps.gz.

[17] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[18] D. Terry, A. Demers, K. Peterson, M. Spreitzer, M, Theimer, and B. Welch. Session guarantees for weakly consistent replicated data. In *Int. Conf. on Parallel and Distributed Information Systems*, 1994.

[19] WindDance Networks Corp. Webchallenger. http://www.winddancenet.com/webchallenger/products/frame_products.htm.

[20] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems (TODS)*, 22(4):255–314, June 1997.

[21] Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson, and David Culler. Using smart clients to build scalable services. In *1997 Annual Technical Conference, January 6–10, 1997. Anaheim, CA*, pages 105–117. USENIX, January 1997.