As appeared in: Proceedings of the ICCCN 1998, Lafayette, LA, October 1998.

# HYDRANET: Network Support for Scaling of Large-Scale Services

Hamesh Chawla Geoff Dillon Riccardo Bettati
Department of Computer Science
Texas A&M University

## Abstract

*With the explosive growth of demand for services on the Internet, the networking infrastructure (routers, protocols, servers) is under considerable stress. Mechanisms are needed for current and future IP services to scale in a client-transparent way. We present* HYDRANET*, an infrastructure that allows to dynamically distribute IP services by placing service agents (caching agents, mirrors, replicas) that are under the server's control at strategic points in the internetwork.* HYDRANET *is based on replicating transport-level service access points for transparent distribution or replication of IP services. Measurements on a local testbed show that the overhead of our scheme is small. This replication scheme is widely applicable. We use* HYDRANET *to implement* HYDRAWEB*, a system for active, push based, and client-transparent Web caching. Similarly,* HYDRANET *can be used to implement highly fault-tolerant servers or application-level gateways.*

## 1. Introduction

The rapid diffusion of the Internet and the explosive increase of the Web's popularity have contributed to an astounding growth in the volume of traffic on the Internet. The consequence has been a massive increase in the load on both the networking infrastructure (links, routers, protocols) and the server hosts. Hence the need to design and implement a network infrastructure that is able to diffuse extreme load conditions has become increasingly important.

In the domain of Web scaling, for example, caching (typically *client-based* or *proxy-based*) is widely seen as the solution to handle the exploding user demands. A large infrastructure for Web caching has been put in place [6, 20]. Although client-based and proxy-based caching greatly help in reducing congestion due to Web traffic *in general*, a variety of factors undercut their effectiveness: First, not all data is cacheable. Some data may be refreshed at a high rate, which greatly reduces the benefits of caching. Second, caching takes away control. A number of sites are reluctant to have their data cached, for a variety of reasons. The caching mechanisms can be easily bypassed by information providers. For example, the `Expire` field in the HTTP reply header [5] can be manipulated by the Web server so that the data is never cached by clients. Such sites then become effectively "cache breakers" [16]. The recent severe and widely publicized congestion of NorthWestNet by the release of Microsoft's Internet Explorer 3.0 happened in this way. The practice of circumventing the caching mechanisms puts enough strain on Internet service providers (ISPs) that some have resorted to ignoring the expiration information from a number of sites. Third, mirroring makes caching expensive. Mirrored data is not easily detectable as such by cache systems, and greatly increases the space requirement on caches for them to be effective.

To overcome some of these limitations, a number of *push-based* methods have been proposed, which give more control to servers about what is being cached [4, 19]. In these schemes, servers push data to the caches, effectively eliminating the problems of cache consistency and the need for mirroring. With the Internet connectivity becoming truly global, however, new large-scale services, beyond the Web, must be expected. Some of these will be transaction oriented, for which caching is largely ineffective.

In order to support the scaling of the Web and of next-generation IP services, an infrastructure is needed that allows the servers to diffuse load and break up hot spots by dynamically placing appropriately programmed components onto strategically placed nodes in the internetwork, typically "near" large client populations. Such components could be caching agents, mirrors, entire copies of servers, or others, and would run under the server's control. Such a scheme should be able to support current and future IP services, and not rely on a service-specific infrastructure (such as Squid [20] for the Web). In addition, this scheme should remain invisible for the clients.

In this paper, we describe an infrastructure that allows to dynamically replicate IP services to have them adapt to network and server load. This is accomplished by replicating transport-level service access points (in our case TCP and UDP ports) across multiple locations in an internetwork. A distributed server can then be realized by installing multiple "components" of the server (for example active caches, or

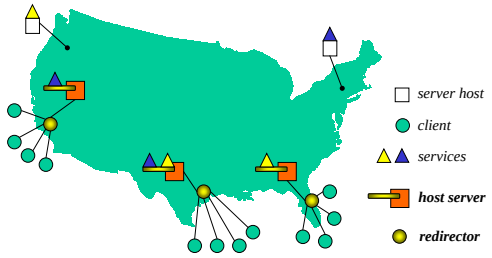



**Figure 1. Replicating Services in** HYDRANET

full-sized copies) at different locations in the internetwork. These components then bind to the network, each with its own copy of the same, replicated, transport-level service access point. In this way, the server appears monolithic to the clients, with a single service access point.

In order to test the feasibility of such an approach, we implemented HYDRANET, a protocol infrastructure for service distribution on IP. HYDRANET consists of two components: host servers and redirectors. *Host Servers* are hosts that act as servers-of-servers. They can host IP services, each of which may be known to the outside world under the IP address of another host. Typically, such services are replicas of the service running on the *Origin Host*, for example mirrors of sites of a Web server. They can also be scaled-down versions of the service (for example active caches) that run as agents of the server on the origin host. In the following discussion we do not distinguish between full-scale copies of the service on one side and scaled-down versions on the other, but call them all *replicas* of the service on the origin host, each running on one host server.

The location of the host servers is known to the *Redirectors*, specially equipped routers that maintain information about the host servers and the services installed on them. Redirectors detect requests for replicated services, and redirect the requests to the "nearest" available host server with a replica running. A replica-management protocol allows to dynamically install services or remove them, depending on the load in the network and on the servers.

The scenario in Figure 1 gives a general idea of how the components of HYDRANET work together: The two services `www.northwest.com` and `www.northeast.com` are accessed by large groups of users from the three ISPs `southwest.net`, `south.net`, and `southeast.net`. Without a replication scheme, the distance from the clients to the servers can cause increased access latencies and network load. In addition, the servers may be overly loaded. In this example, each of the three ISPs routes its traffic through a redirector, and has access to a host server. This allows `northeast.com` to install a replica for its Web server on the host servers of `south.net` and `southwest.net`. Similarly, `northwest.com` installs replicas on `south.net` and `southeast.net`. In this way the network traffic is reduced, and the load balanced, by increasing the service locality. This scenario illustrates how servers, redirectors, and host servers collaborate to bring the services "near" to the clients by conveniently installing replicas on appropriate host servers. In this way, the load on servers and network can be controlled, and hot spots proactively diffused.

One immediate concern is the possible overload on redirectors, which are routers after all. By strategically placing redirectors and host servers at centers with large client populations at the boundaries of the Internet (for example at large ISPs), significant benefits can be achieved without excessively burdening the routers that now act as redirectors. In addition, our measurements show that redirection adds little overhead in the routers.

In this paper, we present HYDRANET, an infrastructure for dynamic service distribution across an internetwork based on the replication of transport-level service-access points. Section 2 discusses related approaches for the replication of services and of service access points. Section 3 describes the architecture of HYDRANET, and Section 4 its realization on a local testbed. We also present the results of a number of performance measurements on HYDRANET. In Section 5 we describe HYDRAWEB, a distributed Web server based on active caches, which we realized on HYDRANET. Section 6 presents the conclusions and an outlook for future work.

## 2. Related Work

A number of schemes have been proposed that address IP service replication and load balancing of distributed servers.
**DNS Based Approaches:** The Internet naming server provides support for load balancing by having a one-to-many mapping from hostnames to IP addresses [7]. Early on, a round-robin DNS was used with success for the NCSA Web server [14]. Replicating services using naming mechanisms has a number of disadvantages. First, substantial caching happens in the name resolution hierarchy of DNS. This adds inertia before the load balancing effect of round-robin resolution kicks in. To make DNS based schemes react more quickly, caching must be limited. This in turn puts substantial load on the name servers. Second, the replication happens at hostname level, which makes the selective replication of services awkward. At the least, services that are replicated differently from each other would require different host names. Third, transparent reconfiguration after a host failure is not possible. Clients typically do not resort to repeated name resolution when a server is not reachable. DNS based approaches to recover from host failures work rather well for the Web because URLs mostly contain host

names and HTTP requests are largely idempotent.

**Intelligent Clients:** A simple solution for service scaling is to have the clients know about the protocol used for service replication. When a new service is deployed, the load balancing protocol can be hard-wired into the client programs. A well-known example is the hidden mirroring in the Netscape browser, which detects requests to the `netscape.com` server and maps them to requests to a range of servers (`www1.netscape.com` to `www32.netscape.com`). Similar approaches can be taken for clients of other services. A more general approach is to down-load client programs from the server site in form of applets and to equip them with the appropriate protocol for load balancing and the contingency plans for server failures. Such an approach is taken in [23].

**Protocol-Specific Replication:** A HTTP specific protocol is used in Cisco's DistributedDirector to redirect requests to other servers [9]. It forces the request redirection by issuing a HTTP "302 Temporarily Moved" status code to the client, along with the URL for the "best" server. A similar approach is used in the SWEB project [1] for clusters of servers. This approach is highly application specific.

**Locally Distributed IP Addresses:** A number of schemes, mostly based on variations of network address translation (NAT) [12], have been proposed that have a cluster of servers be visible under the same IP address. Typically, the server-side router makes one IP address visible and routes incoming requests to one of the several servers in the cluster. The different approaches differ in the details of the translation: Cisco's *LocalDirector* [8] and the *Magicrouter* [2] adhere rather strictly to NAT and have the router perform all the translations. IBM's *TCP Router* [3] does translation on in-bound packets, and allows reply packets from the cluster to be returned directly to the clients. NAT and similar schemes modify the source and destination IP addresses in the packet header. Some higher-level protocols and applications that use IP addresses in the application (for example ICMP or FTP) may not work correctly without appropriate – application-specific – patches of NAT [12].

A different approach is used in *One-IP* [10], where incoming requests are processed by a dispatcher, which forwards them to the cluster nodes. The cluster nodes use the `ifconfig alias` option to attach two IP addresses to the network interface. The primary address uniquely identifies the node in the cluster and is used by the dispatcher to forward requests from clients. Only the secondary address , the *cluster address*, is visible to the clients.

All these schemes are limited to localized clusters of servers, typically on the same subnet. To allow for multiple routers [11], or for clusters that are distributed over multiple domains [9], these approaches resort to a combination with round-robin DNS, which we described above.

## 3. HYDRANET: Architecture for IP Service Replication

An architecture for service replication must be able to selectively and dynamically replicate IP *services* (not hosts) fully transparently to the clients. It must be completely application independent, making it a solution for current and next-generation services. In addition, it should be easy to incrementally shoe-horn into the existing IP infrastructure (similar to the MBONE).

With these requirements in mind, we designed and implemented HYDRANET, an infrastructure for service scaling on IP. HYDRANET replicates services by *globally replicating IP addresses*. A service is replicated by installing replicas on one or more *host servers* and have them bind to the same set of TCP or UDP ports as the service on the origin host. *Redirectors* ensure that the replicas on the host servers are accessible under the same IP address as the origin host. When a redirector receives an IP packet destined to a replicated service, it determines the location of the "nearest" replica of the service, which is identified by the pair of IP address and port number. If the destination of the packet does not appear to the redirector as a service with replica, the packet is simply forwarded to its destination. This allows to dynamically, and transparently, install replicas at strategic locations (for example "near" large client populations).

We borrow the general idea for the support of replicated services from *Mobile IP* [18]. We can think of the processes running replicated services being software equivalents of mobile hosts, and the host servers being the equivalent of foreign agents. In addition to the differences between host servers and foreign agents, the two main points of difference are (1) that *multiple* copies of the replicated servers can be active *simultaneously*, while only one copy of the mobile node exists at any point in time; (2) the redirectors can be thought of as a *distributed* version of the home agent.

**IP-Redirectors.** The location of the host servers is known to the redirectors, which are specially equipped routers. They maintain information about host servers and the services currently installed on them. Each redirector maintains a *redirector table*, which lists the transport-level service access points (in our case pairs of IP addresses and port numbers) for which packets must be redirected, and the host server to which the packets must go. When a redirector receives an IP packet, it checks the destination IP address and port in the header against the entries in the redirector table. If it finds a match, it forwards the packet to the appropriate server host. If there is no match, the packet is simply forwarded to the origin host. A packet is redirected to the appropriate host server by *tunneling* it using IP-in-IP encapsulation. The host server is equipped to detect tunneled packets and to forward them internally to the service.

**Host Servers.** Replicas of server processes are dynamically

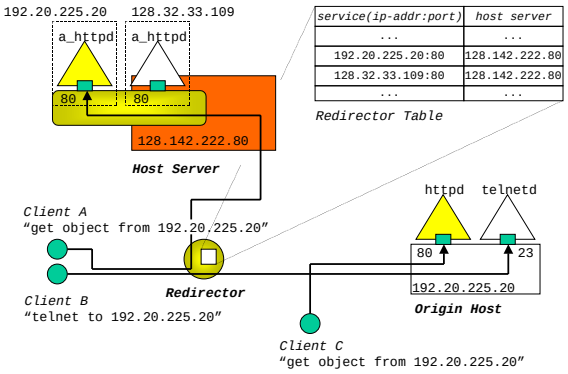



**Figure 2. Components of** HYDRANET

installed on or removed from host servers, which are specially equipped hosts that act as *servers-of-servers*. Before a service replica is installed on a host server, a copy of the origin host's environment is established for the server process to run. We say that a *virtual host* is installed. The protocol software on the host server is informed about the new virtual host. When a packet destined to a virtual host is received by the host server, its destination IP address and port number are compared against currently installed virtual hosts and the ports applications are bound to. If a match is found, the data is deposited at the appropriate socket buffer.

Figure 2 shows how the components of HYDRANET interact. We observe from the figure that Host `128.142.222.80` is a host server. The Web service (realized by the `httpd` daemon) on Host `192.20.225.20` is replicated on the host server, where it is realized by the `a_httpd` replica daemon. Whenever the process on the host server binds to a TCP or UDP port, the host server and the redirectors are informed, and the redirector tables updated. The HTTP requests from Client $C$ are routed to the origin host. The same requests from Client $A$ are intercepted by the redirector, which happens to be on their route, and which was informed earlier that the nearest Web port for host `192.20.225.20` is located on host server `128.142.222.80`. The requests are routed accordingly. Client $B$'s requests for the telnet service continue being forwarded to the origin host; the redirector does not have an entry for the telnet port of host `192.20.225.20`. We note that neither the clients nor the non-participating servers are affected by this scheme. We will see in Section 5 that even participating servers need not be directly affected.

## 4. Implementation

We realized HYDRANET as a set of simple modifications to the process management and the IP protocol stack in FreeBSD. In this section we describe the most important aspects of the design, both on the host server and on the IP redirector.

### 4.1. Host Server

On the host server, we provide mechanisms for installation of virtual hosts, which run the replicated IP services, and for processing and demultiplexing of packets to the virtual hosts.

**Installation of Virtual Hosts:** A replicated service runs as a server program in a virtual host on the host server. Virtual hosts are identified by the IP address of their origin host, and are associated each with a process running on the host server. The kernel of the host server maintains a *virtual-host table*, which contains information about the virtual hosts currently located on the host server. A new virtual host is created by the system call

```
int v_host(u_long ip_address);
```

which associates the currently running process with the given IP address. Whenever the process (or any of its descendents) thus associated to an IP address binds a socket to a port, the port belongs to the virtual host associated with the process. Whenever a socket is created, the kernel checks against the virtual-host table to see whether the socket belongs to a virtual host and marks the socket's protocol control block appropriately. A routing protocol is in place to inform the redirectors about the newly created port on the host server.

**Decapsulation and Transport-Level Demultiplexing:** As mentioned earlier, datagrams are redirected to virtual hosts by tunneling them to the host server. This is achieved by encapsulating the datagram using IP-in-IP encapsulation. Datagrams with encapsulated payload are marked as such in the protocol identifier of their IP headers. Typically, the IP-level demultiplexer feeds the incoming datagrams into the appropriate transport protocol stack according to this protocol identifier. Decapsulation can be easily realized by adding a pseudo protocol stack for datagrams that are marked as IP-in-IP encapsulations. This stack strips the datagram of the encapsulation header and puts the remaining part back into the IP input queue. The IP header of the decapsulated packet now has the address and port of the virtual host installed. When the datagram is processed again by the transport demultiplexer, it is passed to the appropriate transport level protocol, which makes the data available at the correct socket buffer.

### 4.2. Redirector

The redirectors are routers that keep track of the location of host servers and of the virtual hosts installed on them. For this purpose, they maintain a *redirector table*, which is used to detect incoming datagrams that need to be redirected to a host server. The redirector table contains the IP address of the origin host and the port number for the replicated service, and the IP address of the host server to which

the datagrams must be forwarded. Whenever a datagram matches a pair of destination IP address and port number in the redirector table, it is encapsulated and tunneled to the host server. The redirector tables are maintained on the redirectors by HYDRANET routing daemons, which are patterned along the traditional IP routing daemons: The communication among redirectors and between redirectors and host servers happens via well-known UDP ports; each routing daemon accesses and modifies the redirector table in the kernel of the local redirector via routing sockets. If a datagram needs to be redirected, this is detected when the datagram goes through the IP-level processing before it goes out to the interface. If an entry is located in the redirector table, an encapsulation header with the host server's IP address is wrapped around the datagram before it is passed to appropriate interface. This interface can be the local loopback, which means that the same mechanism can be used to redirect datagrams within the host server.

### 4.3. Measurements

We measured the performance impact of our BSD implementation of HYDRANET on a small testbed, which, for measurement purposes, consists of two Pentium/120 PCs and one 486/33 SX PC.

We did measurements with `ttcp` to determine the overhead in redirectors and host servers. In our testbed, the 486/33 SX PC is sufficiently slow that it acts as a bottleneck. By swapping the hosts in the configuration, we could have the redirector be the bottleneck, or the receiving host server. We compared the sustained bandwidth of TCP for the following three series of measurements. (For the measurements, we turned off buffering of small datagrams at the TCP sender, preventing it from batching multiple small datagrams into a datagram of MTU size.)

• *Clean*: All machines run unmodified system software. No redirection happens and no services are replicated. These measurements act as baseline for performance comparison.

• *To real host*: The routers and the receivers run modified system software. There is no redirection, however; the packets are sent to a port on the host server's IP address. This series of experiments measures the amount of overhead in case of no redirections.

• *To virtual host*: Configuration is same as above. The packets, however, are destined to a port on a non-existent host with a replica running on the host server. This line of experiments illustrates the penalties caused by tunneling in the redirector, and of lookup, decapsulation, and additional demultiplexing at the host server.

The above comparisons were made for two configurations of the testbed. In a first set of experiments, the 486/33 SX PC was configured as the redirector, which made the redirector be the bottleneck. Figure 3a illustrates the performance. The results indicate that the overhead on the router/redirector is negligible. We can see a small penalty for redirection when the packet size is close to MTU size (1500 Byte) or slightly larger. This is due to the additional datagram fragmentation that is needed in these cases to add the encapsulation header. For example, a TCP message of 1600 Byte is fragmented at the sender into two datagrams, one of MTU size, which is typically 1500 Byte including TCP and IP header, and one with the rest. When the first datagram reaches a redirector, the latter cannot accommodate the encapsulation header within the datagram size limits. It therefore must further fragment the datagram, which causes overhead in form of additional computation at the router and bandwidth on the links. This problem is pervasive with all approaches based on encapsulation.

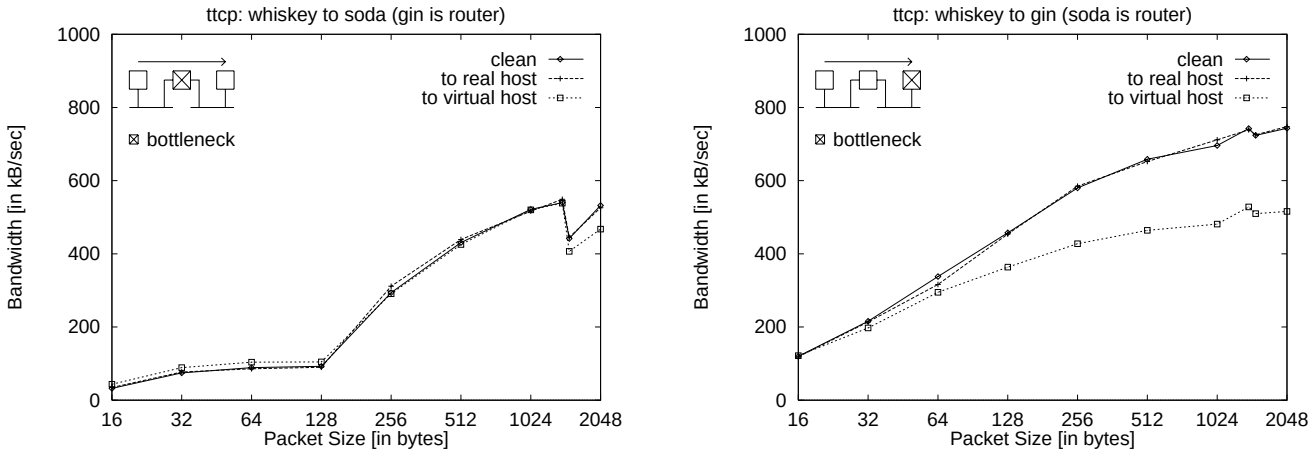
In the second set of experiments, the 486/33 SX PC is configured as the host server, which makes the host server be the bottleneck. Figure 3b illustrates the performance. We notice a significant drop in sustained bandwidth for the case of redirection. Interestingly enough, the fixed per-packet processing overhead seems to be negligible, as the results for small packets show. This indicates that the overhead must be caused by excessive copying of the contents of datagrams as they are processed by the host server. We are optimistic that these data copies can be eliminated, with a beneficial effect on the performance. In addition, the host server in these experiments was severely overloaded. Running replicated services on severely anemic servers would be a bad choice, independent of the replication scheme used.

Separate measurements of TCP connection-setup latency indicate that connection setups to replicated TCP ports on host servers take only marginally longer (less than 0.1msec) than to traditional ports.

## 5. Building Applications on HYDRANET

From an application perspective, HYDRANET allows processes to execute remotely while keeping their "home IP address". This is conceptually very simple, but extremely versatile. We have developed a replica management protocol, which allows servers to dynamically install replicas on host servers, and remove them when they are no longer needed. We are currently developing a suite of applications, which illustrates the wide variety of benefits that can be gained from network support for service replication.

The first such application is HYDRAWEB, a distributed Web server with a traditional server on the origin host and active caches on host servers. Active caches provide the means to eliminate most of the problems encountered in Web caching. First, Servers can dynamically install active caches and push data to make them hot in order to proactively diffuse expected flash crowds. While the caches remain installed, the servers can keep them hot by pushing modified or soon-to-expire data. Second, active caches run




(a) Redirector is Bottleneck

(b) Host Server is Bottleneck

**Figure 3. Comparison of Protocol Overhead**

*under the control of the origin's site*. This means that access control, hit metering, and dynamic page reconfiguration (for example for dynamic placement of advertising banners) is done in behalf of and under the control of the original site. Third, copyright issues for cached data are eliminated, because the site controls the placement of its data. Fourth, management of cache sites is simplified, because they are under centralized control of the original site. The configuration of distributed Web caches is a serious problem. As an example, [24] describes how fourteen separate Australian SQUID sites link themselves directly onto the cache tree in the U.S. Each page is fetched across the Pacific separately by each of the fourteen sites. Next, the generation of dynamically computed pages can be done at the cache sites. And, finally, Explicit mirroring is eliminated.

Given the above points, this solution can be used for cache breakers, notoriously problematic Web sites, and so dramatically improve the performance of already deployed passive Web cache infrastructures. In addition, active caches on the host servers can take over control in the case of failure or excessive congestion of the origin host, effectively providing a highly fault-tolerant Web service.

HYDRAWEB is a simple Java-based realization of an active-cache based replicated Web server. It consists of replicas and replica managers. *Replicas* are small programs that can be down-loaded onto hosts servers. They provide a HTTP interface to clients (i.e. behave like Web servers). Whenever possible, the client requests are handled locally by the replica. For this purpose, the replicas maintain a cache of HTTP objects. The *replica manager* on the origin host installs and removes replicas and generally manages them. Whenever client requests cannot be handled by a replica locally, the replica manager is contacted, and the request is handled on the origin server.

To measure the performance of HYDRAWEB, we installed a Web server at the International Computer Science Institute at UC Berkeley. A number of Web clients at the Computer Science Department of Texas A&M University requested objects from that location in a controlled fashion. We have a redirector and a host server locally, and deployed HYDRAWEB by installing a replica manager on the origin host at Berkeley; that in turn installs a replica locally at Texas A&M. Figure 4 illustrates the setup. In these experiments, the replica is preheated at installation by receiving a predefined set of HTTP objects to cache locally. After it is installed, it does no caching on its own. Whenever the cache is missed, the replica forwards the request to the replica manager, which replies to the replica after having contacted the local Web server.

Figure 5 compares the Web service latencies with and without HYDRAWEB deployed. All requests are for pages of 1200 bytes each. Figure 5(a) shows the service latency distribution for a client at Texas A&M for accesses to the Web server at ICSI with no local replica installed. The average time to get a page from the Web server is 1.5 sec. The ping trace in Figure 5(a) shows that the round-trip time for 64 byte ping packets during the experiment averages 390 msec.

Figure 5(b) shows the result of the same experiment, but with a local replica installed. The replica cache registers 1235 hits against 765 misses. The average service latency for a page in the cache is 120 msec, while for a missed page it is 2.2 sec. Compared to the experiment without HYDRAWEB, page misses in this experiment take 700msec longer to be served. This is in part due to protocol overhead. When a page miss occurs, the replica contacts that replica manager, which gets the page from the Web server and returns it to the replica. The latter then sends it to the client. However, the results of the two experiments should not be directly compared. As the ping traces indicate, the Internet

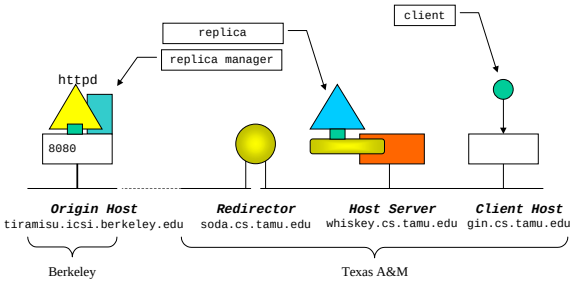



**Figure 4. Deployment of HYDRAWEB between Texas A&M and Berkeley**

was more congested during the second experiment. Indeed, the round-trip for ping packets in the second experiment averages 520 msec.

## 6. Conclusions

In this paper we have identified a number of problems with scaling of very-large services. We described how many of these problems can be eliminated with appropriate network support. In particular, the ability to dynamically install replicas of the transport service access points in strategically placed locations, for example "near" large client populations, greatly increases the capability of the network to balance the load on the servers and to pro-actively diffuse flash crowds to a service. We have implemented the concept of TSAP replication in HYDRANET, an extension to the BSD process management and IP stack software, which allows to dynamically install agent programs on host servers and have redirector routers load balance the servers by appropriately directing requests to either the origin host or to locations of replicas.

The versatility of replicating TSAPs extends far beyond what we described in Section 5. For example, transparent replication of services is an alternative to the use of multicast approaches for realizing fault-tolerant servers. In the context of CORBA, for example, current efforts to provide reliable ORB technology (e.g. [15]) rely mostly on multicast capabilities at protocol level, and typically require clients to use these capabilities as well. HYDRANET would allow to replicate ORBs over several host servers. When a particular ORB becomes inaccessible (because of network partitioning, congestion, or host failure,) we rely on the reconfiguration capability of routers and redirectors to redirect requests to the remaining replicas of the ORB.

A number of issues need to be addressed further. First of all, we need to find ways to guarantee safe execution to replicas. We envision host servers to be managed by a variety of operators: ISPs, network services, or third-party operators ("service hosting for hire", similarly to the WebOS's "Rent-A-Server" concept [22]). Host servers will therefore often be outside the control of the entity controlling the origin host, but host services from potentially large numbers of different sources. Mechanisms must be in place that enforce safe coexistence of multiple third-party processes on host servers. We plan to set up a transparent run-time environment that provides a replicated server a "sandbox", the "size" of which is negotiated when the replica is installed.

The fact that programs run "under your name" on multiple hosts across the Internet poses serious security risks. Appropriate mechanisms must be in place in the host servers (e.g. digital signatures of down-loaded server replica code) and redirectors (e.g. authentication of host servers) to prevent unauthorized execution of replica code under a wrong identity.

## References

[1] D. Andresen, T. Yang and O.H. Ibarra. SWEB: Towards a Scalable World Wide Web Server on Multicomputers. *Proceedings of the IPPS'96*, April 1996.

[2] E. Anderson, D. Patterson and E. Brewer. The Magicrouter, an Application of Fast Packet Interposing. URL *http://http.cs.berkeley.edu/ ˜eanders/projects/magicrouter*.

[3] C.R. Attanasio, S.E. Smith. A Virtual Multiprocessor Implemented By an Encapsulated Cluster of Loosely Coupled Computers. *IBM Research Report, Advanced RISC Systems, RC18442*, April 1992.

[4] M. Baentsch, L. Baum, G. Molter, S. Rothkugel and P. Sturm, Enhancing the Web's Infrastructure: From Caching to Replication, *IEEE Internet Computing*, March 1997.

[5] T. Berners-Lee, R. Fielding and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. *RFC 1945*

[6] C.M. Bowman, P.B. Danzig, D.R.Hardy, U.Manber and Michael Schwartz, The Harvest Information Discovery and Access System, *Computer Networks and ISDN Systems*, 28 (1995), pp. 119-125

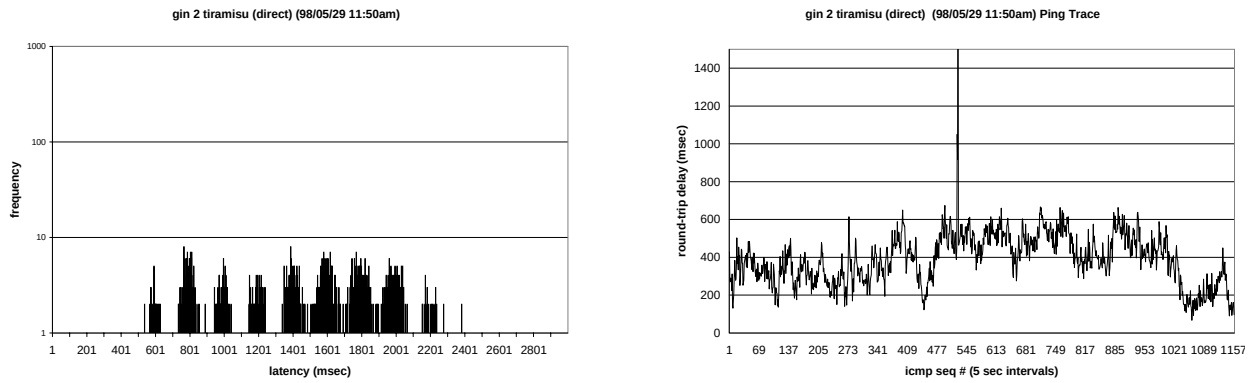
[7] T. Brisco. DNS Support for Load Balancing. *RFC 1794*

[8] Cisco Systems, LocalDirector, URL *http://www.cisco.com/warp/public/751/lodir/index.html*.

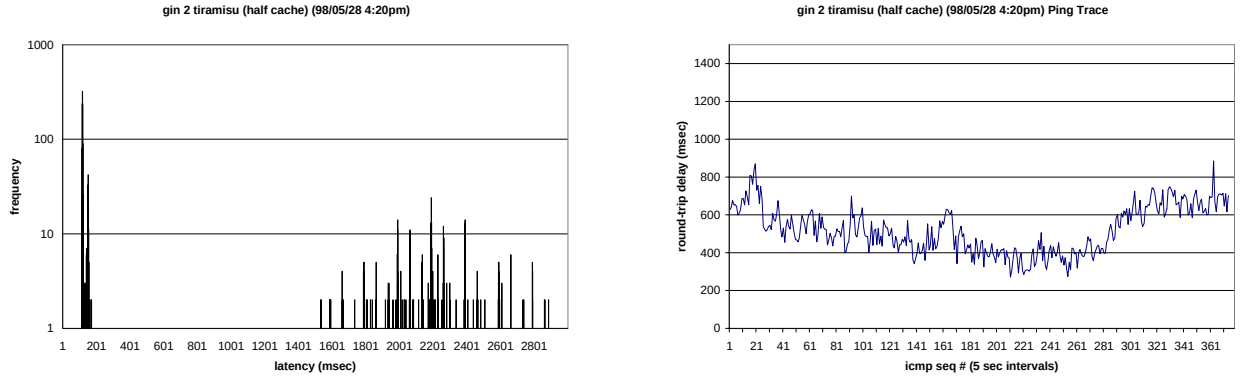
[9] Cisco Systems, K. Delgadillo, Cisco DistributedDirector, URL *http://www.cisco.com/warp/public/751/distdir/dd_wp.html*.

[10] O.P. Damani, P.E. Chung, Y. Huang, C. Kintala, Y.-M. Wang. ONE-IP: Techniques for Hosting a Server on a Cluster of Machines. *Hyperproceedings of the Sixth International World Wide Web Conference*, URL *http://www.nttlabs.com/HyperNews/get/PAPER196.html*

[11] D.M. Dias, W. Kish, R. Mukherjee and R. Tewari. A Scalable and Highly Available Web Server. *Proceedings of COMPCON'96*, Santa Clara, CA, 1996.




(a) Delay Distribution without Replication




(b) Delay Distribution using HYDRANET

**Figure 5. Web Access Delays from Texas A&M to Berkeley**


[12] K. Egevang and P. Francis. The IP Network Address Translator (NAT). *RFC 1631*, May 1994.

[13] J. Gwertzman, M.Seltzer. The Case for Geograhical Push Caching, *Fifth Workshop on Hot Topics in Operating Systems*, 1995.

[14] T. Kwan, R. McGrath and D. Reed, NCSA's World Wide Web Server: Design and Performance, *Computer*, Vol. 28, No. 11, Nov. 1995.

[15] S. Landis and S. Maffeis, Building Reliable Distributed Systems with CORBA, *Theory and Practice of Object Systems*, John Wiley, New York. (to appear)

[16] Cache Breakers. URL *http: //www.iphil.net/˜map/-cache/breakers.html*.

[17] National Laboratory for Applied Network Research. Insight into Current Web Caching Issues: Research Questions. URL *http: //ircache.nlanr.net/Cache/-Learn/learn-2.html*

[18] C. Perkins. IP Mobility Support. Internet Draft of the IETF, January 1995.

[19] M. Seltzer, The World Wide Web: Issues and Challenges, URL *http:// www.eecs.harvard.edu/margo/-slides/www.html*.

[20] Squid Internet Object Cache, URL *http: //squid.nlanr.net/Squid*.

[21] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, G.J. Minden. A Survey of Active Network Research. *IEEE Communications*, January 1997.

[22] A. Vahdat, E. Belani, P. Eastham, C. Yoshikawa, T.E. Anderson, D.E. Culler, and M. Dahlin. WebOS: Operating System Services For Wide Area Applications. *Seventh Symposium on High Performance Distributed Computing.* July 1998.

[23] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, D. Culler. Using Smart Clients to Build Scalable Services. *USENIX 97*, January 1997.

[24] L. Zhang, S. Floyd, and V. Jacobson, Adaptive Web Caching, *NLANR Web Cache Workshop'97*, Boulder, CO, June, 1997.