In Proceedings of ICNP'98: The 6th International Conference on Network Protocols, Austin, Texas, USA, Oct 1998.

Distributed Packet Rewriting
*and its Application to Scalable Server Architectures**

Azer Bestavros Mark Crovella Jun Liu
Computer Science Department
Boston University
Boston, MA 02215
{best,crovella,junliu}@cs.bu.edu

David Martin†
Dept of Math and CS
University of Denver
Denver, CO 80208
dm@cs.du.edu

Abstract

To construct high performance Web servers, system builders are increasingly turning to distributed designs. An important challenge that arises in such designs is the need to direct incoming connections to individual hosts. Previous methods for connection routing (Layer 4 Switching) have employed a centralized node to handle all incoming requests. In contrast, we propose a distributed approach, called Distributed Packet Rewriting (DPR), in which all hosts of the distributed system participate in connection routing. DPR promises better scalability and fault-tolerance than the currrent practice of using centralized, special-purpose connection routers. In this paper, we describe our implementation of four variants of DPR and compare their performance. We show that DPR provides performance comparable to centralized alternatives, measured in terms of throughput and delay. Also, we show that DPR enhances the scalability of Web server clusters by eliminating the performance bottleneck exhibited when centralized connection routing techniques are utilized.

1. Introduction

The phenomenal, continual growth of the World Wide Web (Web) is imposing considerable strain on Internet resources, prompting numerous concerns about the Web's continued viability. In that respect, one of the most common bottlenecks is the performance of Web servers—popular ones in particular. To build high performance Web servers, designers are increasingly turning to distributed systems. In such systems, a collection of hosts work together to serve Web requests. Distributed designs have the potential for scalability and cost-effectiveness; however, a number of challenges must be addressed to make a set of hosts function efficiently as a single server.

Connection Routing: Consider the sequence of events that occur as a result of a client requesting a document from a Web server. First, the client resolves the host's domain name to an initial IP address. Second, the IP address itself may represent a distributed system, and one of the hosts in the system must be chosen to serve the request. There are many ways to perform the first mapping (from domain name to initial IP address). For example, this mapping could be coded in the application as is done within Netscape Navigator to access Netscape's Home Page [8]. Alternately, this mapping could be done through DNS by advertising a number of IP addresses for a single domain name. Similarily, there are many ways to perform the second mapping (from initial IP address to actual host). For example, this mapping could be done at the application level, using the HTTP redirection approach [1] or using a dispatcher at the server [2, 17].

While initial attempts to implement connection routing for scalable Web servers focused on using the mapping from domain names to IP addresses [10], recent attempts have focussed on the second kind of mapping (IP addresses to hosts) because of the potential for finer control of load distribution. One common feature of all of these attempts (whether proposed or implemented) is that a centralized mechanism is employed to perform the mapping from IP addresses to hosts. Examples include the Berkeley MagicRouter [2], the Cisco Local Director [17], and IBM's TCP Router [7] and Network Dispatcher [9].

Distributed Connection Routing using DPR: In contrast, DPR is a technique that allows the mapping between IP address and host to be implemented in a *distributed*, efficient, and scalable fashion. In particular, DPR can be viewed as a distributed method of mapping m IP addresses to n servers.[1] Using DPR, every host in a Web server cluster acts *both* as a server and as a connection router. Thus, unlike existing solutions that rely on a single, centralized connection router, DPR enables both the service and the routing responsibilities to be

*This work was partially supported by NSF research grants CCR-9706685 and CCR-9501822.

†Research completed while co-author was at Boston University.

[1] If $m = 1$, then DPR becomes similar to the centralized solutions mentioned above—the difference being that DPR allows *both* packet routing and service to be combined on the same node.

shared by all hosts in the cluster. Distributing the connection routing functionality allows for true scalability, since adding a new host to the cluster automatically adds enough capacity to boost *both* Web service and connection routing capacities.

To illustrate the benefits of using DPR, consider the problem of scaling up a Web site that initially consists of a single server host. Adding a second server host using typical existing solutions (for example, Cisco's Local Director [17], or IBM's NetDispatcher [9]) requires using special-purpose hardware to distribute incoming HTTP requests between the two server hosts. This kind of centralized solution provides connection routing capacity that far surpasses what a two-host server is likely to require. In other words, the upgrade path (and hence the price tag) for a centralized solution is not truly incremental: the two-host server will be roughly three times the cost (if an ordinary PC is used as a centralized router) and may reach ten times the cost of a single-host server.

An additional, important issue for many content providers is that the centralized solution creates a single-point-of-failure in the system, which leads to even more costly solutions such as using a second, standby connection router. Thus for mission-critical Web sites, centralized connection routing escalates the imbalance in capacity between connection routing and connection service. These problems disappear when using a DPR-based architecture. Adding a second server to the site requires no special hardware, introduces no single-point-of-failure, and utilizes the added capacity (and hence dollars spent) to scale both the connection routing and connection service capacities equally.

Paper Contribution and Scope: The novelty of DPR lies in its *distribution* of the connection routing protocol (Layer 4 Switching), which allows all hosts in the system to participate in request redirection, thereby eliminating the practice of using a special purpose connection router to achieve that functionality.

DPR is one of the salient features of COMMONWEALTH—an architecture and prototype for scalable Web servers being developed at Boston University. The design of DPR is driven by a large set of goals that the COMMONWEALTH architecture strives to achieve. These goals are:

1. *Transparency:* Clients should not be exposed to design internals. For example, a solution that allows a client to distinguish between the various servers in the cluster—and hence target servers individually—is hard to control.
2. *Scalability:* Increasing the size of the cluster should result in a proportional improvement in performance. In particular, no performance bottlenecks should prevent the design from scaling up.
3. *Efficiency:* The capacity of the cluster as a whole should be as close as possible to the total capacity of its constituent servers. Thus, solutions that impose a large overhead are not desired.
4. *Graceful Degradation:* The failure of a system component should result in a proportional degradation in the offered quality of service. For example, a solution that allows for a single point of failure may result in major disruptions due to the failure of a miniscule fraction of the system.
5. *Connection Assignment Flexibility:* Connection assignment techniques should be flexible enough to support resource management functionalities—such as admission control and load balancing.

In the remainder of this paper we show how DPR supports these goals in the construction of the COMMONWEALTH server. In the next section we review related work and show why DPR is different from previous proposals for connection routing in Web servers. Then in Section 3 we describe the design tradeoffs for DPR and the variants of DPR that we have implemented and tested in our laboratory. In Section 4 we show performance results using DPR, indicating that DPR induces minimal overhead and that it achieves performance scalability superior to that achievable using existing centralized connection routing. Finally, in Section 5 we conclude with a summary.

2. Related Work

Preliminary work on scalability of Web servers has been performed at NCSA [10] and DEC WRL [14]. In both cases, load is balanced across server hosts by providing a mapping from a single host name to multiple IP addresses. In accordance with DNS standard, the different host IP addresses are advertised in turn [16]. In addition to its violation of the transparency property discussed in the previous section, both the NCSA and DEC WRL studies observe that this “Round Robin DNS” (RR-DNS) approach leads to significant imbalance in load distribution among servers. The main reason is that mappings from host names to IP addresses are cached by DNS servers, and therefore can be accessed by many clients while in the cache. The simulations in [7] suggest that, even if this DNS caching anomaly is resolved, the caching of Host-to-IP translations *at the clients* is enough to introduce significant imbalance.

Rather than delegating to DNS the responsibility of distributing requests to individual servers in a cluster, several research groups have suggested the use of a local “router” to perform this function. For example, the NOW project at Berkeley has developed the MagicRouter [2], which is a packet-filter-based approach [13] to distributing network packets in a cluster. The MagicRouter acts as a switchboard that distributes requests for Web service to the individual nodes in the cluster. To do so requires that packets from a client be forwarded (or “rewritten”) by the MagicRouter to the individual server chosen to service the client's TCP connection. Also, it requires that packets from the server be “rewritten” by the MagicRouter on their way back to the client. This *packet rewriting* mechanism gives the illusion of a “high-performance” Web Server, which in reality consists of a router and a cluster of servers. The emphasis of the MagicRouter work is on reducing packet processing time through "Fast Packet Interposing", not on the issue of balancing load. Other solutions based on similar architectures include the Local Director by Cisco [17] and the Interactive Network Dispatcher by IBM [9].

An architecture slightly different from that of the MagicRouter is described in [7], in which a "TCP Router" acts as a front-end that forwards requests for Web service to the individual back-end servers of the cluster. Two features of the TCP Router differentiate it from the MagicRouter solution mentioned above. First, rewriting packets from servers to clients is eliminated. To do so requires modifying the server host kernels, which is not needed under the MagicRouter solution. Second, the TCP Router assigns connections to servers based on the state of these servers. This means that the TCP Router must keep track of connection assignments.

The architecture presented in [11] uses a TCP-based switching mechanism to implement a distributed proxy server. The motivation for this work is to address the performance limitations of *client-side* caching proxies by allowing a number of servers to act as a single proxy for clients of an institutional network. The architecture in [11] uses a *centralized* dispatcher (a Depot) to distribute client requests to one of the servers in the cluster representing the proxy. The function of the Depot is similar to that of the MagicRouter. However, due to the caching functionality of the distributed proxy, additional issues are addressed—mostly related to the maintenance of cache consistency among all servers in the cluster.

3. Implementation of DPR

As described in Section 1, our goals in developing DPR were transparency, scalability, efficiency, fault tolerance, and flexibility in connection assignment. Previous centralized approaches (described in Section 2) have focused on transparency and load balance: these are natural features deriving from a design using centralized routing. The two dominant styles of centralized routing are shown in Figure 1 (a) and (b). Figure 1 (a) shows the MagicRouter style, in which packets traveling in both directions are rewritten by a centralized host. Figure 1 (b) shows the TCP router style, in which only packets traveling from the clients are rewritten, still by a centralized host. An important advantage of the TCP router style is that the majority of bytes in a Web server flow from the server to the client, and these packets do not require rewriting.

In contrast to centralized approaches, we seek to address our wider set of goals, which also include scalability and fault tolerance. As a result we adopt a distributed approach to TCP routing, namely distributed packet rewriting. Under DPR, each host in the system provides both Web service *and* packet routing functions, as shown in Figure 1(c). Under DPR the structure of any connection is conceptually a loop passing through three hosts (client and two server hosts). The entire set may have no hosts in common with another connection on the same distributed server. We refer to the first server host to which a packet arrives as the *rewriter,* and the second host as the *destination.*

Centralized schemes place the rewriting task within the routers connecting a distributed web server to the internet (or as close to such routers as possible). DPR instead transfers this responsibility to the Web servers it concerns. This can be seen as an instantiation of the end-to-end argument: the choice of the final server is essentially a service-specific decision, and so should be made as close as possible to the service points rather than being distributed throughout general-purpose network components.

Another important advantage of DPR is that the amount of routing bandwidth scales with the size of the system, in contrast to the centralized approaches. Furthermore, since the routing function is distributed, this system can not be wholly disabled by the failure of a single node—as is possible under centralized approaches.

The DPR scheme assumes that requests arrive at the individual hosts of the server. This can occur in a number of ways. The simplest approach (which we currently use) is to distribute requests using Round-Robin DNS. Although requests may well arrive in a unbalanced manner because of the limitations of RR-DNS, hosts experience balanced demands for service because of the redistribution of requests performed by DPR.

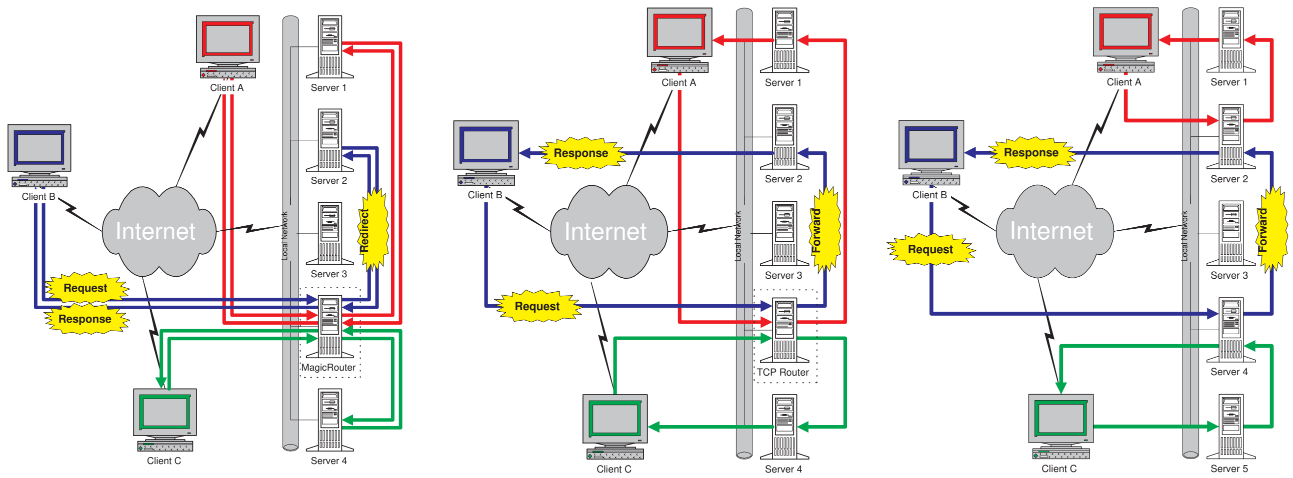
Design Tradeoffs

Two design issues arise in determining the specific capabilities of a DPR implementation. First, will routing decisions be based on stateless functions, or will it require per-connection state? Second, how should rewritten traffic be carried on the server network? The following sections investigate these questions and describe decisions made in our various implementations extending the Linux 2.0.30 kernel with DPR support.

Stateless vs Stateful Routing:
It is possible to balance load across server hosts using a stateless routing function, *e.g.,* a function that computes a hash value based on the source and destination IP and TCP port addresses of the original packet. On the other hand, more sophisticated load balancing policies may require more information than what is contained in the packets, for example, knowledge of load on other hosts. In this case, each rewriting host must maintain a routing table with an entry for each connection that is currently being handled by that host.

Stateless Approach: In the stateless approach, we use a simple hash function on the client's IP address and port number to determine the destination of each packet. Since the client's IP/port forms a unique key for requests arriving at the server, this function is sufficient to distribute requests.

Using server logs from the BU Web site in simple simulations, we have verified that our hash function is effective



(a) MagicRouter/LocalDirector (b) TCP Router (c) Distributed Packet Rewriting

Figure 1. Illustration of various architectures for distributed Web Servers

at balancing load (in terms of hits per server over time) for actual client request arrivals. An important factor in this success is the use of the client port number as an input to the hash function; the client's TCP layer indirectly chooses the final server when it selects its ephemeral TCP port. Successive port numbers from the same client should map to different server hosts, dispersing each burst across servers, and thus alleviating the imbalance due to the burstiness of client requests [7].

Although stateless implementations are lightweight and the resulting server load distributions are acceptable, we must keep in mind the inability of the rewriter to route a connection based on other factors (such as end-server load, distance, availability, or the necessity that successive requests be routed to the same host for correct session semantics).

Stateless/LAN Implementation: This variant takes the simplicity and speed of stateless MAC address rewriting to an extreme. Because no state is stored, the additional code and data required is small. The Stateless/LAN implementation simply overwrites the MAC address of the packet and retransmits it on the LAN. The simplicity of the transformation allows rewriting to occur in the context of the network device driver, namely, in the kernel routine that device drivers use to register incoming packets. This implementation thus receives, rewrites, and retransmits packets all within a single interrupt service cycle; furthermore, no device-specific modifications are required.

While rewriting the entire request in the interrupt routine provides performance virtually indistinguishable from that of a dedicated rewriting router (see Section 4), our Stateless/LAN implementation is not practical. When Stateless/LAN processes fragmented packets, only the first IP fragment contains the necessary TCP port information to ensure proper delivery and subsequent fragments are misrouted. Because of this shortcoming, we used this implementation only as an indication of an upper bound on the performance that can be achieved with DPR-style techniques.

Stateful Approach: In the stateful approach, the packet routing decision is based on more information than is contained in the packet. For example, a stateful approach is necessary in order to route connections based on the current load on each server host. Most of our efforts have concentrated on this approach.

Stateful Implementation: In the stateful method, rewriters must track TCP connection establishment and termination. A table of connections currently being rewritten is maintained by each host and is consulted in order to rewrite each packet. In implementing these functions we were able to adapt features from code already present in the Linux kernel that supports *IP Masquerading*. IP Masquerading [12] was developed to allow multiple hosts to function behind a firewall without valid IP addresses. Thus, IP Masquerading supports connections that are initiated *from* the "hidden" hosts. In order to support a distributed server, we need to support connections connecting *to* the hidden hosts.

Using the IP Masquerading functions adapted to support a distributed server, the rewriter has considerable freedom to choose a destination when it receives the first packet of a client's TCP stream. After noting its decision in a state table, it then forwards each packet associated with the connection using either the MAC rewriting or IPIP encapsulation technique, depending on the network location of the destination.

At present, the routing decision for a newly observed connection is made by simply obtaining the next entry in a ring of server addresses. This ring is extended to user space through a `setsockopt(2)` system call. By populating the ring intelligently, a user daemon can adjust the rewriting policy as server conditions change.

We note that independently and at approximately the same time as our work, Clarke developed a general-purpose TCP forwarding kernel extension based on IP Masquerading [5] which can also be used to support implementation of distributed Web servers.

Addressing Techniques:

There are two approaches to addressing packets bound for another host in a multiple-server environment, depending on whether the original destination IP address must be communicated from the rewriter to the destination host.

The first approach is appropriate when there is only one published IP address for the whole Web server cluster (as would be the case when a centralized connection router is used). In this case, the original packet's destination IP address (IP_1) is replaced with that of the new destination (IP_2) and then the packet is routed normally. When the new destination transmits packets to the client, it must be careful to replace its IP source address (IP_2) with that of the rewriter (IP_1), because the client believes its connection to be with the rewriter (*i.e.* IP_1). Every host in the Web server cluster knows that its outbound traffic should always bear the source address IP_1, so the address IP_1 need not explicitly appear in rewritten packets.

One consequence of this technique is that the IP and TCP checksums need to be adjusted when rewriting, since they both depend on the destination IP addresses in the packet. (In practical terms, only the IP checksum is important, since IP routers do not examine the payload of IP packets they encounter [3]. However, firewalls and other types of "smart routers" might in fact examine the TCP checksum, so it is advisable to recompute it as well.)

The second approach applies to systems with more than one published IP address, as in DPR. In a DPR system, a mechanism is needed to communicate both the original address (IP_1) and the rewritten address (IP_2) in packets sent between the rewriter and the destination hosts so that the destination knows how to populate the IP source address field. The most efficient method we used was to rewrite the MAC address of the packet and retransmit, leaving the original packet's IP addresses and checksums undisturbed. (This is how Internet hosts normally send packets through a gateway.) Although fast, the method only works if both servers are located on the same LAN. If the servers are on different LANs, then IP-level routing is necessary. In this case we tunnel the original packet to IP_2 with IPIP encapsulation as described in RFC2003[15]. When the packet arrives at IP_2, the outer IPIP header is discarded and the inner header is interpreted.

Whether MAC rewriting or IPIP encapsulation is used, the server with primary address IP_2 eventually receives and processes an IP packet bearing the original destination address IP_1. Therefore, each server must be configured to respond to all of the possible original destination addresses (such as IP_1) in addition to its own primary address. In our Linux implementation, this was just a matter of adding loopback alias devices with the extra addresses.

4. Performance Evaluation

In this section we describe the performance of DPR variants. We have two goals: first, to characterize the overheads present in DPR; and second, to study the scalability of DPR as compared to centralized connection routing.

To address these two goals we ran two series of experiments. The first series used a small network in determining the performance overhead of Stateful DPR and Stateless/LAN DPR when compared to a centralized connection router, and to baseline cases involving no connection routing. For this set of experiments we used the SPECweb96 [6] benchmarking tool because it places relatively smooth loading on the server over time.

The second series of experiments concentrated on exploring the scalability of Stateful DPR compared to centralized connection routing. Since our goal in this section was to explore how DPR would behave under realistic conditions, we used the Surge reference generator [4] to provide the server workload. Surge is a tool developed as part of the COMMONWEALTH project that attempts to accurately mimic a fixed population of users accessing a Web server. It adheres to six empirically measured statistical properties of typical client requests, including request size distribution and interarrival time distribution. As a result, it places a much burstier load on the server than does SPECweb96. In addition, while SPECweb96 uses an open system model (requested workload is specified in GETs/sec), Surge adopts a closed system model (workload is generated by a fixed population of users, which alternate between making requests and lying idle). As a result, Surge's workload intensity is measured in units of User Equivalents (UEs).

In both series of experiments we restricted our configurations to a single LAN in order to provide repeatable results. Although the LAN was not completely isolated during our measurements, only a negligible amount of unrelated traffic (mostly ARP requests) was present.

4.1. Performance Overhead of DPR

As described above, SPECweb96's principal independent parameter is the requested throughput, measured in HTTP GETs per second. The measured results of each experiment are the achieved throughput (which may be lower than what was requested) and the average time to complete an HTTP GET (measured in msec/GET). For each experiment, we ran SPECweb96 for the recommended 5 minute warmup, after which measurements were taken for 10 minutes. System hosts (both clients and servers) consisted of Hewlett-Packard Vectra PCs, each having a 200MHz Pentium Pro processor, 32 MB of memory, and a SCSI hard drive. Servers ran Linux 2.0.30 on Linux ext2 filesystems, while clients ran Windows NT 4.0. We used the NT Performance Monitor to ensure that

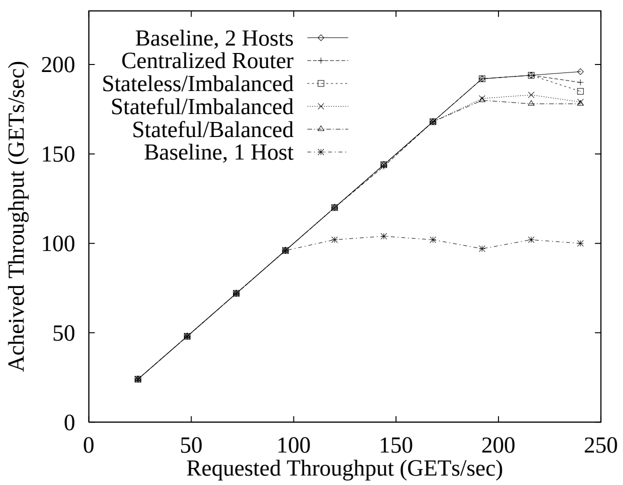


Figure 2. Throughput of DPR Variants

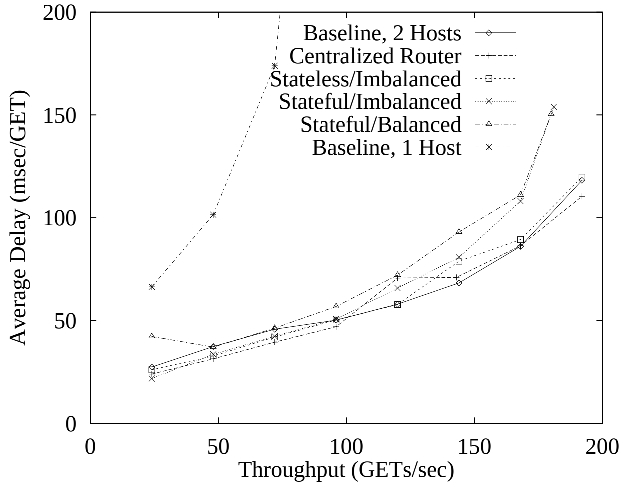


Figure 3. Request Delay of DPR Variants

our clients had capacity to spare when our servers became saturated. The LAN was a 100 Mbit/sec Hewlett-Packard AnyLAN switch; this star network is frame-compatible with Ethernet, but it also uses a round-robin schedule together with a client sensing mechanism so that packet collisions do not occur. The Web servers used were Apache 1.2.4.

We describe the results of six experiments:

Baseline 1-Host. This experiment tests the performance of a single, unmodified server driven by a single client.

Baseline 2-Host. This experiment consists of two simultaneous copies of the Baseline 1-Host experiment. It uses two clients and two servers, and each client sends requests to only one server.

Centralized Router. This experiment consists of two clients sending requests to a centralized connection router, which then distributes the load evenly between two servers. The Centralized Router implementation is our Stateful DPR configured to redirect all connections to the other two servers (i.e. the routing function does not compete with local web service).

Stateless/Imbalanced. This experiment uses the Stateless/LAN variant of DPR, running on two hosts. Two clients generate requests, but they send *all* requests to one of the server hosts, which then redistributes half of them.

Stateful/Imbalanced. This experiment uses the Stateful variant running on two hosts. Again two clients generate requests, sending all requests to one host, which redistributes half of them.

Stateful/Balanced. This experiment again uses the Stateful variant, but now the two clients generate equal amounts of requests for each server host. Each host then redistributes half of its requests, sending them to the other server.

Baseline 1-Host and Baseline 2-Host define the range of possible performance for the systems under study, with Baseline 2-Host defining the best performance that might be expected from a 2-host server system. The Centralized Router results represent the performance of the most common alternative to DPR, and show the effect of removing the packet rewriting function from the server hosts. Note that each packet travels through two server nodes in the DPR and Centralized Router cases, and through only one server node in the Baseline cases.

The Stateless/Imbalanced and Stateful/Imbalanced experiments serve to show the worst possible performance of DPR, *i.e.,* when the arriving request load is maximally imbalanced (all requests to one host). The Stateful/Balanced experiment allows comparison of the best and worst possible load arrival distributions for DPR.

Throughput:

In Figure 2 we show the achieved throughput of each experimental system as a function of the requested throughput. The Baseline 1-Host case saturates at about 100 GETs/sec, and the Baseline 2-Host case at the corresponding level of about 200 GETs/sec. In between the experiments fall into two groups: the Stateful experiments saturate at about 180 GETs/sec, while the Stateless/Imbalanced and Centralized Router saturate at about 195 GETs/sec. The fact that the Stateful/Balanced and Stateful/Imbalanced show nearly identical performance indicates that when requests arrive in a highly imbalanced way and all packet rewriting occurs on only one host, DPR is still able to achieve good throughput. This comparison indicates that the performance demand of packet rewriting is quite moderate, and so adding a packet rewriting function to a host already performing Web service does not represent a significant additional burden.

Comparing the Stateful and Stateless cases, we see that the Stateless case performs indistinguishably from the Centralized Router case, and they both are equivalent to the Baseline 2-Host case (in which no packet rewriting is taking place at all). The similarity of the Stateless to the Baseline 2-Host case shows that the performance cost of packet rewriting in

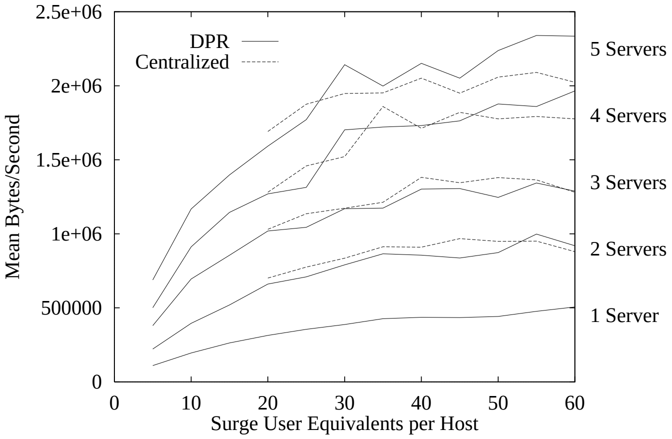


Figure 4. Throughput Comparison

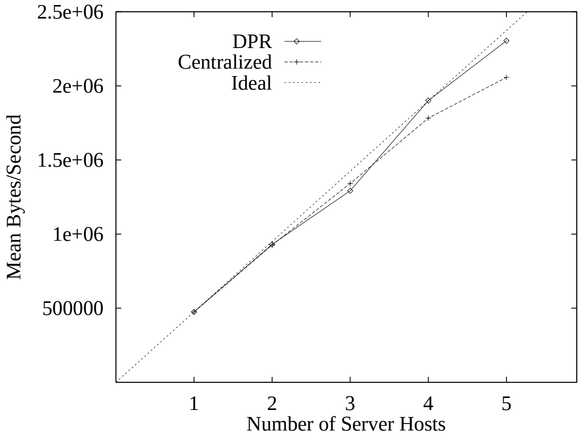


Figure 5. Scalability Comparison

the Stateless/LAN implementation is negligible.

An important implication of the similarity of the Stateless and Centralized Router cases is that the centralized connection routing architecture is not cost-effective. This is because an entire node has been allocated to the connection routing task (there are three nodes in the Centralized Router case and only two in the DPR cases). Thus the additional cost of adding a specialized connection router to a small system may not be justified. It is just as efficient, and cheaper, to use the server hosts already present to perform the connection routing function. This point will be reinforced by our results in Section 4.2 on the scalability of the DPR architecture compared to the centralized routing architecture.

Delay:
In addition to providing high throughput, it is important to verify that DPR does not add unacceptable delays to the system. In Figure 3 we show the average response time of an HTTP GET (in msec/GET) as a function of system throughput, for the same six experiments. In this figure we only plot those points for which achieved and requested throughput are nearly equal, so throughput does not reach quite the same maximum values as in Figure 2. Figure 3 shows that the experiments break into the same groupings as before. Again, the Stateful/Balanced and Stateful/Imbalanced cases show approximately similar performance. Furthermore the Stateless case shows approximately similar delays to the TCP Router and the Baseline 2-Host cases.

Since packets travel through an additional server node in the DPR and TCP Router cases as compared to the Baseline 2-Host case, there is a potential for greater delay in those cases. However, it appears that the additional delays induced by the additional hop are small compared to the average response time for an HTTP GET. The response time of an average HTTP GET under SPECweb96 is in the range of 25 to 150 milliseconds on a LAN. Were the system serving packets over the global Internet, response times would be even greater since the added round-trip times would be tens to hundreds of milliseconds as well. The addition of additional packet processing due to Stateless/LAN DPR, which appears to be on the order of tens to hundreds of microseconds, is a negligible additional cost for a Web server application.

4.2. Scalability of DPR

The previous section showed that the overheads of DPR were no greater than that of a centralized connection router, and that even when connection routing load was completely unbalanced, system performance did not suffer. These results suggest that DPR should show good scalability, but it is still necessary to evaluate DPR's scalability in practice. For comparison purposes we also evaluate the centralized connection routing case.

The scalability series of experiments took place on different equipment than the performance overhead experiments. All of the Web/DPR servers were Dell Dimension PCs with 64 MB of memory and IDE hard drives running Linux 2.0.30 and Apache 1.2.1. Four of the Web servers had 200 MHz Pentium Pro processors, and one had a 233 MHz Pentium II. The latter system appears in our results as the fourth Web server in both DPR and centralized connection router experiments. Our fastest system, a 266 MHz Pentium II, was used only as a connection router. Both clients and servers used Linux ext2 file systems. The LAN was a 12-port 3Com SuperStack II Switch 3000 10/100 running Ethernet in full duplex at 100Mb/s. The total bandwidth measured during the experiments show that network capacity was not a limiting factor; we also observed that our clients were able to saturate our servers before reaching their own capacity.

As described above, for these experiments we used the Surge load generator. In the DPR cases, we configured Surge so that equal amounts of traffic were directed at each server host. In an N host system, each DPR host serves $1/N$ of the requests locally and distributes $(N-1)/N$ of the requests equally to the other hosts in the system. In adopting this routing policy, our results for DPR are quite conservative.

A better policy that is still quite practical would be for each server host to only redirect requests that arrive when the host is loaded above the system average; in that case, a fraction of requests much smaller than $(N-1)/N$ would be redirected, and the overall performance of the DPR system would be better than that reported here.

In our experiments we compare N-host DPR systems to centralized routing configurations consisting of N server hosts *plus* a connection router. By doing so, we emphasize the scalability difference between the architectures. However these tests do not compare equivalent systems in terms of hardware costs; as described above, it is more cost-effective to organize a system of N hosts in a DPR architecture than to set aside one host solely for connection routing.

In order to scale the demand placed on the server systems as the number of server hosts grows, it is necessary to proportionally increase the number of User Equivalents used in Surge. For this reason we report results in terms of User Equivalents per server host.

The achieved throughput for a range of both DPR systems and centralized routing systems is shown in Figure 4. This figure shows that for small systems (2-3 hosts), DPR and centralized routing behave approximately equivalently. However for larger systems (4-5 hosts), the centralized approach seems to show lower maximum throughout than the DPR approach. This evidence that the centralized node is beginning to become a bottleneck is supported by the fact that the difference between the two systems becomes more pronounced as the demand grows.

To illustrate the onset of a bottleneck effect in the centralized routing case, we show the peak throughput achieved as a function of the size of the system in Figure 5. Peak throughput was obtained in each case by averaging the throughput over the range 50-60 UEs per host (which in each case was where system saturation was judged to have set in). The figure also shows the "ideal" throughput obtained by simply scaling up the throughput obtained by a single unmodifed host.

This figure shows that DPR obtains near-perfect speedup for server systems up to five hosts in size. In contrast, the centralized routing architecture seems to show signs of inefficiency at larger sizes; on four hosts, maximum throughput under centralized routing has dropped to 94% of ideal, and on five hosts the centralized system is only 86% efficient.

5. Summary

In this paper we have proposed and experimentally evaluated a protocol for routing connections in a distributed server without employing any centralized resource. Instead of using a distinguished node to route connections to their destinations, as in previous systems, Distributed Packet Rewriting (DPR) involves *all* the hosts of the distributed system in connection routing. The benefits that DPR presents over centralized approaches are considerable: the amount of routing power in the system scales with the number of nodes, and the system is not completely disabled by the failure of any one node. DPR allows more cost-effective scaling of distributed servers, and as a result more directly supports the goals of the COMMONWEALTH project.

References

[1] D. Anderson, T. Yang, V. Holmedahl, and O. Ibarra. SWEB: Towards a Scalable World Wide Server on Multicomputers. In *Proceedings of IPPS'96*, April 1996.

[2] E. Anderson, D. Patterson, and E. Brewer. The MagicRouter: An application of fast packet interposing. `http://HTTP.CS.Berkeley.EDU/~eanders/projects/magicrouter/osdi96-mr-submission.ps`, May 1996.

[3] F. Baker. IETF RFC1812: Requirements for IP Version 4 Routers. See `http://ds.internic.net/rfc/rfc1812.txt`.

[4] P. Barford and M. Crovella. Generating representative workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98*, pages 151–160, Madison, WI, June 1998.

[5] S. Clarke. Port Forwarding in Linux. See description at `http://www.ox.compsoc.org.uk/~steve/portforwarding.html`.

[6] T. S. P. E. Corporation. Specweb96. `http://www.specbench.org/org/web96/`.

[7] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available web server. In *Proceedings of IEEE COMPCON'96*, pages 85–92, 1996.

[8] S. Garfinkel. The Wizard of Netscape. *WebServer Magazine*, pages 58–64, July/August 1996.

[9] IBM Corporation. The IBM Interactive Network Dispatcher. See `http://www.ics.raleigh.ibm.com/netdispatch`.

[10] E. D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. In *Proceedings of the First International World-Wide Web Conference*, May 1994.

[11] K. Law, B. Nandy, and A. Chapman. A Scalable and Distributed WWW Proxy System. Technical report, Nortel Limited Research Report, 1997.

[12] Linux IP Masquerade Resource. See `http://ipmasq.home.ml.org`.

[13] J. Mogul, R. Rashid, and M. Accetta. The Packet Filter: An Efficient Mechanism for User-level Network Code. In *Proceedings of SOSP'87: The 11th ACM Symposium on Operating Systems Principles*, 1987.

[14] J. C. Mogul. Network behavior of a busy Web server and its clients. Research Report 95/5, DEC Western Research Laboratory, Oct. 1995.

[15] C. Perkins. IETF RFC2003: IP Encapsulation within IP. See `http://ds.internic.net/rfc/rfc2003.txt`.

[16] R. J. Schemers. lbnamed: A Load Balancing Name Server in Perl. In *Proceedings of LISA'95: The 9th Systems Administration Conference*, 1995.

[17] C. Systems. Scaling the Internet Web Servers: A white Paper. `http://www.cisco.com/warp/public/751/lodir/scale_wp.htm`, 1997.