Distributed Packet Rewriting

and its Application to Scalable Server Architectures[*]

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

December 1, 1997

Abstract

To construct high performance Web servers, system builders are increasingly turning to distributed designs. An important challenge that arises in distributed Web servers is the need to direct incoming connections to individual hosts. Previous methods for connection routing have employed a centralized node which handles 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. We argue that this approach promises better scalability and fault-tolerance than the centralized approach. 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 under the SPECweb96 benchmark. Finally, we argue that DPR is particularly attractive both for small scale systems and for systems following the emerging trend toward increasingly intelligent I/O subsystems.

Keywords: TCP/IP; TCP Routers; Round Robin DNS; Scalable Web servers; IP Masquerading.

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.

In this paper we show how a number of the challenges involved in building distributed servers can be addressed using *Distributed Packet Rewriting* (DPR). DPR is one of the features of COMMONWEALTH—an

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

architecture and prototype for scalable Web servers being developed at Boston University. The COMMONWEALTH project aims to design, implement, and evaluate a prototype architecture and a set of associated protocols for scalable Web services.

The function of DPR is to direct requests for Web service to one of the hosts in the distributed server. DPR is novel because its function is distributed, that is, all hosts in the system participate in request redirection. To illustrate the need for a function like DPR, consider the sequence of events when a client requests 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 construct scalable Web servers focussed 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 [6] and Network Dispatcher [4].

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. If $m = 1$, then DPR becomes similar to the centralized solutions mentioned above.[1]

The design of DPR was driven by a large set of goals for the server, derived from the goals of the COMMONWEALTH project. These goals (which are not necessarily all compatible) 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. For example, a design that employs a centralized function cannot be scaled up once the capacity of the centralized resource is reached.

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

[1] If $m = 1$, the difference between DPR and the centralized solutions proposed in [2, 6, 4] is that DPR allows *both* packet routing and service to be combined on the same node.

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:* The technique used to assign connections to servers in a cluster 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 of DPR, and the four variants of DPR that we have implemented. In Section 4 we show performance results using DPR, indicating that DPR achieves performance comparable to the best known alternatives. Finally, in Section 5 we conclude.

2 Related Work

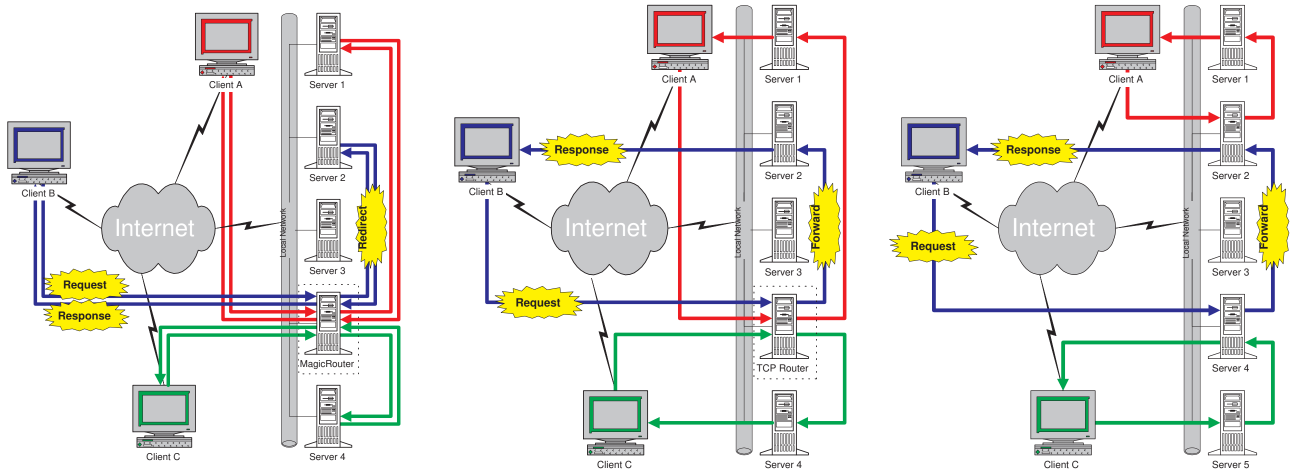
Preliminary work on scalability of Web servers has been performed at NCSA [10] and DEC WRL [13]. In both cases, load is balanced across server hosts by providing a mapping from a single host name to multiple IP addresses. In accordance with DNS standard, the different host IP addresses are advertised in turn [16]. In addition to its violation of the transparency property discussed in the previous section, both the NCSA group and the DEC WRL group 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.

In [13], Mogul empirically characterized the imbalance caused by RR-DNS for a cluster or 3 servers. His measurements suggest that over 25% of the time, the most-loaded server sustained more than 54% of the of the total load in the system compared to a 33% of the total load under perfect balance—an imbalance of $0.54/0.33 = 1.6$. Dias *et al* [6] confirmed these empirical results via simulations. They showed that the peak load on nodes of a cluster can be up to 40% higher than the mean load on all nodes (an imbalance of 1.4), and that this load imbalance is independent of the number of servers in the cluster.

The simulations in [6] unveil another interesting observation. Even if the anomaly caused by DNS caching is resolved, the caching of Host-to-IP translations *at the clients* is enough to introduce significant imbalance. In particular, the experiments of Dias *et al* [6] show that even if DNS caching is turned off—by setting the Time-To-Live (TTL) for cached entries to 0—the load imbalance amongst servers persists due to the burstiness of client requests.[2]

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 [14] 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

[2]A "user click" results in a burst of requests to fetch the page and all its embedded components. Even under RR-DNS, all of these requests would be directed to the same server.



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

Figure 1: Illustration of various architectures for distributed Web Servers

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"—but 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 [4].

An architecture slightly different from that of the MagicRouter is described in [6], 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. This is particularly important when serving large volumes of data (which is the purpose of the system described in [6] for Video Service). To allow for the elimination of packet rewriting from server hosts to clients 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. In [6], the authors sketch various options for distributing the TCP Router functionality. However, no design or implementation details were given with respect to this proposed architecture.

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 amongst all servers in the cluster.

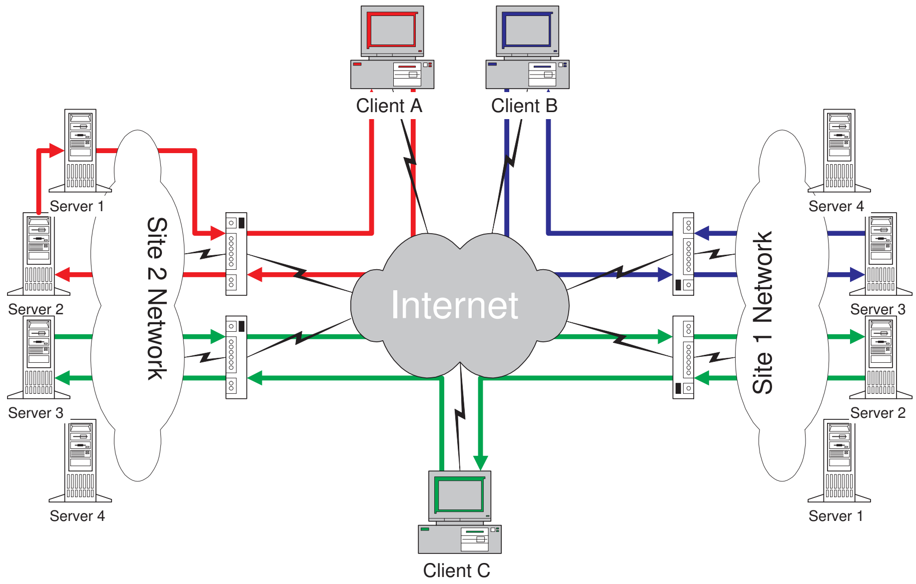


Figure 2: Range of forwarding possibilities under DPR

3 Implementation of DPR

3.1 Design Principles for Distributed Packet Rewriting

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 server. We refer to the first server host to which a packet arrives as the *rewriter*, and the second host as the *destination*.

Figure 2 illustrates the diversity of service paths supported by the DPR architecture. Client A initially contacts Server 2 on Site 2. Server 2 rewrites that connection through Server 1, which then delivers the requested data to Client A. Meanwhile, Client B intially connects to Server 3 on Site 1; although DPR-enabled, it decides not to redirect the connection and so serves the request itself. Finally, Client C's requests visit both Site 2 and Site 1 before finding the correct server.

In each case the packets must traverse the pictured routers. While the centralized schemes attempt to place the rewriting task within these routers or as close to them as possible, DPR 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 (MagicRouter or TCP Router) 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 to 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.

3.2 Design Tradeoffs

Two design issues arise in determining the specific capabilities of a DPR implementation. First, will routing be based on stateless functions, or will it require per-connection state? Second, will DPR routing occur only within a single LAN, or will rewritten packets travel over an internetwork?

The first issue is related to the choice of algorithm used to balance load in the server. 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 port addresses of the 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.

The second issue is concerned with the question of whether all the hosts of the distributed server reside on a single LAN. If so, it is not strictly necessary to encode the IP header consistently for the packet while it is traveling from the rewriting host to the destination host. This allows for a more efficient implementation, but prevents the server from being widely distributed in a geographical sense.

These two design tradeoffs suggest four variants of DPR, all of which we have implemented. In order of the overhead of the implementation, they are: Stateless/LAN, Stateless/WAN, Stateful/LAN, and Stateful/WAN. Each of these was implemented in the Linux 2.0.x kernel. Note that while the generic term "TCP routing" implies decisions made at the TCP layer, it is not generally necessary to modify TCP-layer code to implement TCP routing. Enough information is available at lower levels (the IP level and the device driver level) and implementing at these levels provides greater efficiency.[3] The remainder of this

[3]In Linux, the network "stack" is essentially monolithic anyway, so there is no real layering violation in making DPR-style

section overviews implementation details of these four variants.

3.2.1 Stateless Approaches

In the stateless approach, we use a simple hash function based on the sender's IP address and port number to determine the destination for each packet. Since the IP/port of the sender 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 at balancing load 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. Successive port numbers from the same client should map to different server hosts because of the bursty nature of requests arriving from a single client, as discussed in Section 2.

Stateless/LAN implementation. The Stateless/LAN implementation offers the greatest opportunities for efficiency. Because no state is stored, the additional code and data required is small. Furthermore, because the packet will not travel across an internetwork, it can be retransmitted without any modification. The Stateless/LAN implementation simply changes 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.

Stateless/WAN implementation. The Stateless/WAN implementation forwards packets based on the same hash function, but does so in a way that allows the rewritten packets to travel over an IP network. To do this we use simple IP tunneling, known as IPIP, described in RFC2003 [15]. In this implementation, the rewriting host prepends an additional IP header to the packet with the IP address of the destination, and retransmits the packet. The effect is to transform the old headers and data into the payload of the new packet. The receiver strips the IP header off, and finding an IP packet inside, re-registers the packet with the IP protocol's incoming packet handler. As a result packets in this scheme require slightly more processing time on both the rewriter and the receiver as compared to the LAN variant.

3.2.2 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.

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

routing decisions at the IP level.

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 complete 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 local (**Stateful/LAN**) or the IPIP (**Stateful/WAN**) technique, depending on the network location of the destination. Each destination hosts maintains IP aliases for its rewriters in order to handle streams that do not bear its primary IP address.

We note that independently and at approximately the same time as our work, Clarke has developed a general-purpose TCP forwarding kernel extension based on IP Masquerading [3] which can also be used to support implementation of distributed Web servers.

4 Performance Evaluation

In this section we describe the performance of DPR variants. We restricted our experiments 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.

Since we confine ourselves to single-LAN measurements, our main results concentrate on the Stateless/LAN and Stateful/LAN variants of DPR. However, our measurements of packet processing costs show that the WAN variants are only slightly slower than the LAN variants. For example, the Stateless/LAN variant requires about 75 μs of CPU time to rewrite a packet, and adds less than 10 μs to the processing cost of the packet on the receiver; in comparison, the Stateless/WAN variant also requires about 75 μs of CPU time to rewrite a packet, while adding about 50μs to the processing cost of the packet on the receiver.

We used the SPECweb96 [5] benchmarking tool to measure the throughput and delay characteristics of DPR implementations. 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,[4] while clients ran Windows NT 4.0. 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 simply tests the performance of a single, unmodified server driven by a single client.

[4] In the course of experimentation, we found that MS-DOS filesystems performed very badly when mounted under Linux.

Baseline 2-Host. This experiment consists of two simultaneous copies of the Baseline 11Host experiment. It uses two clients and two servers, and each client sends requests to only one server.

TCP Router. This experiment consists of two clients sending requests to a TCP router (a centralized rewriter) which then distributes the load evenly between two server hosts.

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/LAN 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/LAN 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 TCP 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 TCP 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.

Although our experiments only include two server hosts, we expect that the DPR approach will scale better than the TCP router approach; as a result we believe that our results are conservative and that larger systems would show even more favorable comparisons for DPR.

4.1 Throughput

In Figure 3 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 TCP Router saturate at about 195 GETs/sec.

The fact that the Stateful/Balanced and Stateful/Imbalanced show nearly identical performance indicates that the 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.

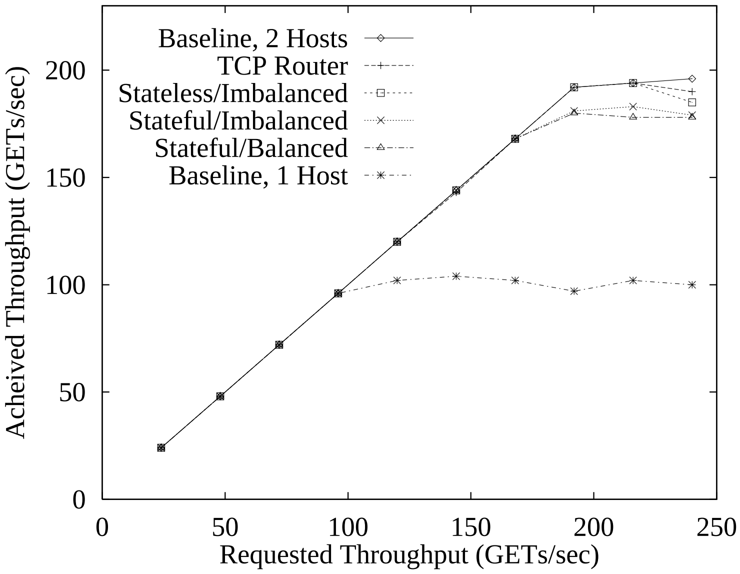


Figure 3: Throughput of DPR Variants and Comparison Cases

Comparing the Stateful and Stateless cases, we see that the Stateless case performs indistinguishably from the TCP 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 the Stateless/LAN implementation is negligible. In addition, the similarity of the Stateless and TCP Router cases suggests that the additional cost of adding a TCP 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 load balancing function.

4.2 Delay

In addition to providing high throughput, it is important to verify that DPR does not add unacceptable delays to the system. In Figure 4 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 3.

Figure 4 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

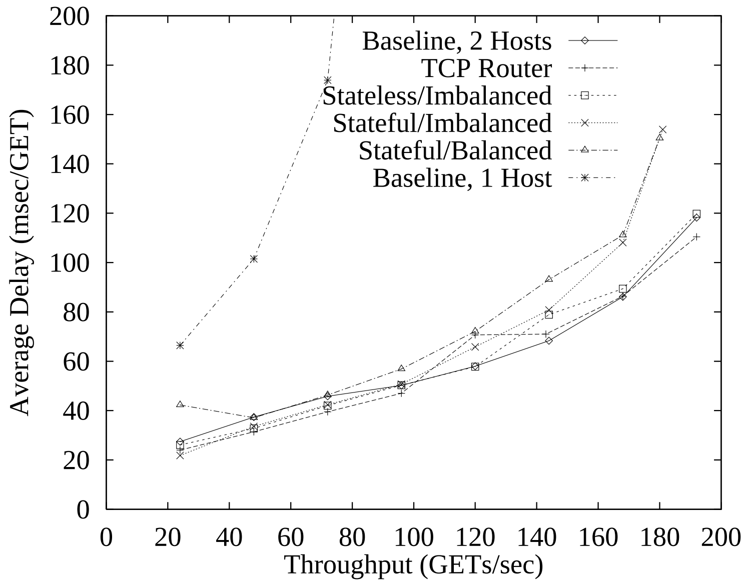


Figure 4: Request Delay of DPR Variants and Comparison Cases

Experiment	Max Throughput Achieved in HTTP GETs/sec	Corresponding Avg Latency in msec/HTTP GET
Baseline 1-Host	104	912.4
Stateful/Imbalanced	183	150.5
Stateful/Balanced	180	150.4
Stateless/Imbalanced	194	131.3
TCP Router	194	126.9
Baseline 2-Hosts	196	124.0

Table 1: Maximum Throughput in Each Experiment.

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.

These results are summarized in Table 1, which shows the maximum throughput achieved in each experiment. This table shows that in terms of both throughput and delay, the Stateless/LAN implementation of DPR achieves performance comparable to a perfectly balanced system with no packet rewriting (Baseline 2-Host) as well as to a system with an additional host whose sole task is packet rewriting (TCP Router). In addition, the Stateful versions of DPR show performance that is close, in both throughput and delay, to the best possible case as well.

5 Conclusion

In this paper we have described a method for balancing load in a distributed server without any centralized resource. Instead of using a distinguished node to route connections to their destinations, as in previous systems, DPR employs *all* the hosts of the distributed system in the connection routing function.

One concern about such an approach is that the addition of connection routing to the responsibilities of the hosts in a server may overburden them with an unacceptable amount of additional work. We have shown that in our implementations, this is not the case. Furthermore, we believe that architectural trends will increasingly favor the colocation of packet routing functions with other system functions in individual hosts. This is because I/O interface hardware, and network interface cards in particular, are rapidly increasing in sophistication. The Intelligent I/O initiative (I_2O) [9] is in fact standardizing hardware and software interfaces for the use of highly intelligent I/O cards in general purpose computing systems. As these trends accelerate, approaches like DPR will become even more attractive.

The functions of DPR do not completely replace those of a TCP router such as Network Dispatcher [4] or Local Director [17]. TCP routers present a single IP address while performing packet rewriting, load balancing, and (potentially) network gatewaying (that is, IP routing). DPR does not present a single IP address, and does not perform network gatewaying. However, we have shown that simple RR-DNS is sufficient for providing the illusion of a single IP address, and standard routers are sufficient (and preferable) for providing gatewaying functions. Although sophisticated load balancing (*e.g.,* based on feedback from server hosts) is more difficult under DPR than under a centralized approach, we have found that static policies can often work well, and we are investigating distributed feedback approaches.

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 also has special value for small scale systems. For example, consider the case in which a Web server needs to grow in capacity from one host to two. Under a centralized approach, two additional hosts must be purchased: the new host *plus* a connection router, even though most of the capacity of the connection router will be unused. 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.H. Ibarra. SWEB: Towards a Scalable World Wide Server on Multicomputers. In *Proceedings of IPPS'96*, April 1996.

[2] Eric Anderson, David Patterson, and Eric Brewer. The MagicRouter: An application of fast packet interposing. submitted to *OSDI 1996*, May 1996.

[3] Steven Clarke. Linux Port Forwarding. See `http://www.ox.compsoc.org.uk/~steve/portforwarding.html`.

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

[5] The Standard Performance Evaluation Corporation. Specweb96. `http://www.specbench.org/org/web96/`.

[6] Daniel M. Dias, William Kish, Rajat Mukherjee, and Renu Tewari. A scalable and highly available web server. In *Proceedings of IEEE COMPCON'96*, pages 85–92, 1996.

[7] Damani et al. ONE-IP: Techniques for Hosting a Service on a Cluster of Machines. In *Proceedings of the Sixth International WWW Conference*, April 1997.

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

[9] I_2O Special Interest Group. See `http://www.i2osig.com`.

[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.L.E. 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] Jeffery C. Mogul. Network behavior of a busy Web server and its clients. Research Report 95/5, DEC Western Research Laboratory, October 1995.

[14] Jeffrey Mogul, Richard Rashid, and Michael 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.

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

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

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