A Client Oriented, IP Level Redirection Mechanism*

Sumit Gupta
A. L. Narasimha Reddy
Dept. of Elec. Engg.
Texas A & M University
College Station, TX 77843-3128

Abstract

This paper introduces a new approach for implementing transparent client access to network services. Ever increasing load on the Internet has made it essential to design services that are fast, reliable, easily manageable, transparent to access, and that can scale gracefully with load. A common way of achieving this has been replicating services across multiple servers and redirecting clients to different servers depending upon various criteria. Existing schemes are either entirely server or network based. This scheme involves the client network layer actively in redirection. The paper describes the redirection protocol in detail and the basic implementation of the testbed. The performance of the mechanism is measured by experiments on the testbed and analyzed. The advantages and disadvantages of client based network level redirection are discussed and some useful applications that it enables are described.

1 Introduction

The recent explosive increase in the size and popularity of the Internet is straining both networks and servers. Most Internet sites were originally not designed to handle the request traffic that they are experiencing. It has become critical to find solutions to alleviate the network and the server level congestion at these sites with high traffic. A common solution to solve these problems is to use multiple servers to offer services - a scheme called **Service Replication**. Schemes have been proposed that transparently "redirect" a client's service request to one of the replicated servers. It is possible to improve the quality of the delivered service by redirecting the client more intelligently based on the server load or network throughput. These methods perform client redirection at

*This work was supported in part by a Texas ATP grant and by an NSF Career Award

various levels in the protocol stack and at various points in the network. Each method has its advantages and limitations.

The objective of this paper is to introduce a new approach for transparent redirection of clients to replicated servers. Almost all existing schemes take an entirely server based approach i.e., all the work is done by the specialized servers or routers. This work implements a client-oriented redirection scheme that operates in the Internet Protocol (IP) [1, 2] in which the client IP layer is actively involved in the redirection of a local client. A client based IP level scheme not only performs well for traffic redirection, but with slight modifications can be cast into a variety of potentially useful scenarios and applications.

Section 2 describes some of the existing schemes for client redirection. Section 3 introduces our approach. Section 4 goes into the advantages and disadvantages of our scheme as compared to the ones described in section 2. Section 5 explains the implementation in detail and section 6 discusses the results. Section 7 touches upon how such a scheme can be used to enhance the features of a variety of applications on the Internet. Section 8 discusses future work and concludes the paper.

2 Background and Related Work

There are several implementations of client redirection mechanisms available today. The design space for providing transparent access to scalable network services includes clients, network, routers, and the service site. All the existing mechanisms can be broadly categorized according to the level in the protocol stack where redirection is implemented and the entity which does most of the work.

DNS Server Based Solutions: An example of a DNS (Domain Name Service) [3, 4, 2] level redirection mechanism is the HTTP [5] server prototype developed by the National Center for Super-computing Applications [6]. In this scheme, one host-name alias is used for the site. The authoritative DNS server for the domain maintains a one-to-many mapping of the server name to the IP address of each of the servers. When clients send name resolution queries, the name-server returns each of the addresses in a round-robin fashion. Clients are served by one of the servers and the round-robin name resolution may result in balancing the load among the servers. Local caching by DNS servers makes the load less than perfectly balanced among the servers. Another problem with round robin DNS schemes is that as DNS servers know nothing about network topology, server availability or server capacity (powerful multiprocessor or a low end PC), users might be connected to a distant, unavailable, or overloaded server.

Server Based IP Level Solutions: Both the Magic-router [7] and Local Director [8] fall under this category. In these schemes, services are replicated across a cluster of machines on a single subnet with a modified router. Only one logical IP address is advertised for the site and the router maps the incoming client requests to one of the hidden servers by inspecting and modifying

the destination IP addresses of all packets going through it. Both techniques provide transparent redirection and load balancing. The TCP router (Transmission Control Protocol Router) [9] takes a very similar approach but configures the cluster nodes so that they can send packets directly back to the client. The front end router is a single point of failure and also can easily become a bottleneck as the number of clients increases. Moreover, all the servers actually providing the service have to be geographically localized to be on the hidden internal network which does not give too much improvement in the quality of service to a distant client if the bandwidth between the site and the client is the bottleneck.

Protocol Specific Redirection: This refers to redirection mechanisms that use features of particular protocols to achieve redirection. For instance, the Distributed Director [10], in its HTTP mode, forces redirection of HTTP requests by issuing a HTTP "302 Temporarily Moved" status code to the client, along with the URL for the server. HTTP Redirect Mechanism [5] also provides a protocol specific redirection mechanism.

Active Networks These schemes place the responsibility and mechanisms for redirection on the network [11, 12, 13]. Intermediate routers are kept up-to-date about the availability, load etc. of replicated servers and are designed to perform customized computations and modifications on packets flowing through them. The routers transparently re-route packets meant for the original server to one of the replicas. However, depending on the network topology and the distance of the translation point from the client and the replicated server, the packets might undergo dog-leg routing. Moreover, it is difficult to provide information about servers for each application in the routers, not to mention the amount of memory needed in the router to store this information along with translation tables for all the flows through it.

Smart Clients: This refers to schemes where the intelligence for redirection is built into the client applications themselves. One current implementation [14] consists of an API that provides a level of abstraction between a logical service and the physical servers. When a user requests a service, a bootstrapping mechanism is used to retrieve a service specific Java applet which has information about servers offering that service, their load, and their availability etc. The current versions include a web based front end for Telnet and FTP (File Transfer Protocol) to a network of workstations. This scheme can provide good load balancing, fault tolerance and has good features like generic naming. However, smart client schemes are highly application specific. A new front end has to be designed and deployed for each application.

In our scheme, only one IP address is advertised for the site. The server with this IP address acts as the redirctor and also provides service to some clients. When a client requests service, the server determines if the client supports our redirection mechanism. If the client has support for this scheme, then it is redirected to another server using our protocol. If it does not support redirection, the redirector itself provides the requested service. The redirection protocol works at the IP layer and is quite versatile. The exact mechanism is described in detail in the following sections.

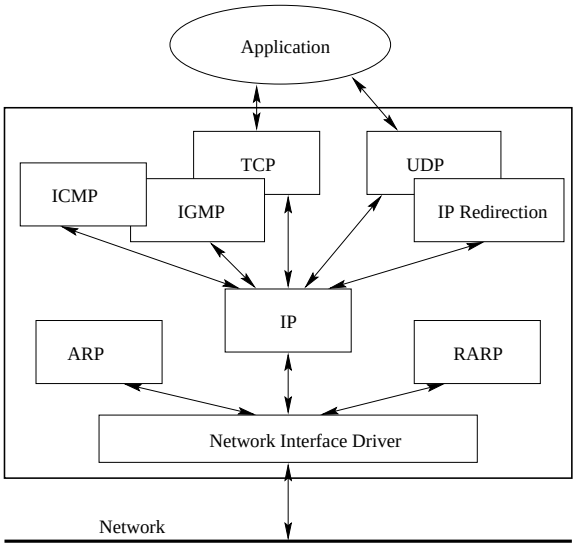


Figure 1: The Protocol Stack

3 The Redirection Protocol

3.1 The Big Picture

In this scheme, both the clients and redirector have to be running the IP Redirection protocol (IPRP), which exists at the same level as the Internet Control Message Protocol (ICMP) [15, 2] or the Internet Group Management Protocol (IGMP) [16, 2]. It is conceptually a part of IP, but has its own protocol number, just like any other transport protocol. Figure 1 shows the logical positioning of different protocols in a stack and the place where this redirection protocol fits in.

Broadly, the redirection handshake works as follows: All supporting clients have the ability to create and maintain redirection bindings, which specify how a local application should be redirected. When the client contacts the redirector for service, the redirector requests the client IP, through a request packet, to create a new binding and map packets destined for the redirector to a particular server. The client then confirms this binding with the redirector by sending a check message that includes a randomly generated key which must be echoed by the redirector for the client to confirm the binding. Once the redirector confirms the binding, all packets of that flow in either direction are translated by the client IP and directly travel between the client and the redirected server. The actual message format is discussed in the next subsection. Figure 2 shows the redirection handshake and the level in the protocol stack where each message is processed.

Figure 3 shows the format of the redirection messages. The *Type* field is a 16 bit field containing

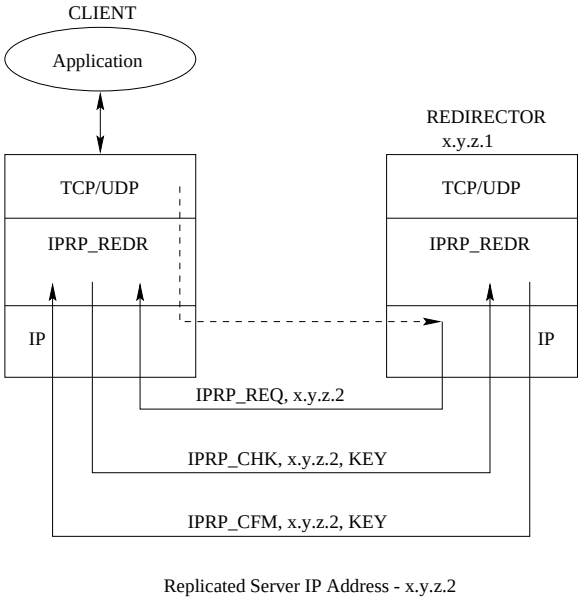


Figure 2: The Redirection Handshake Procedure

a constant that identifies the type of the message - IPRP_REQ (Redirection Request), IPRP_CHK (Redirection Check), IPRP_CFM (Redirection Confirm) or IPRP_ALM (Alarm). *Protocol Number* is the protocol number specifying whether the redirected connection is a TCP or UDP connection. *Client Port* is the TCP or UDP port at the client and *Server Port* is the port at the redirector and the server (which also represents the service being requested). The *Replicated Server IP Address* field contains the 32 bit IP address of the server that the client is being redirected to and *Key* contains the 32 bit secure key used for redirection.

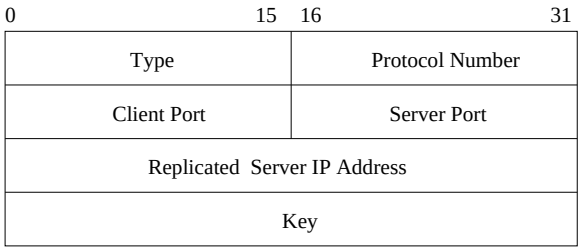


Figure 3: Redirection Message Format

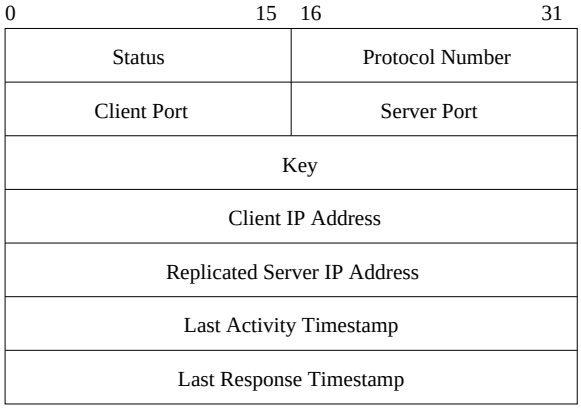


Figure 4: Redirector's Table Entry Format

3.2 The Redirector

The redirector maintains two tables. A *Redirection table* stores entries identifying clients that are in the process of being redirected. Similarly, an *Accept table* keeps track of clients for whom the redirector decides to provide the requested service itself. These might be clients that don't support redirection or for whom the redirection process failed for some reason.

Figure 4 shows the structure of an entry in the redirector's tables. The *Status* field denotes the redirection state of the connection from the server's point of view, and is either IPRP_SENT (redirection request sent) or IPRP_CONFIRMED (redirection binding confirmed with key). As in the message format, *Protocol Number* denotes whether the connection is TCP or UDP. *Server Port* identifies the requested service. The redirector uses *Client IP Address*, *Client Port*, and *Protocol Number* to uniquely identify a client. The redirector also stores the random key used in redirection in the *Key* field. The server may use this key later to send control messages to the client (if need be), without having to agree on a new key. *Replicated Server IP Address* is the address of the server that this particular client was redirected to. The *Last Activity Timestamp* is used by the redirector to judge if a connection has been successfully redirected or has died, and if it is time for the entry to be deleted. The *Last Response Timestamp* field stores the last time a response was received from the client. This is used for retransmissions.

The accept table has exactly the same structure as the redirection table. In fact, entries are moved from the redirection table to the Accept table if the redirection attempt fails. However, not all the fields in an entry structure are relevant in the context of the accept table. Besides these two tables, the redirector also maintains an array of the IP addresses of replicated servers for each of the services that it wishes to redirect.

The redirector functions in the following way. If the received packet's service (port number) is not configured to be redirected or if the redirector has accepted to provide service to this client (in the accept table), the packet is passed up to the transport layer.

Else, if an entry exists in the redirection table, it indicates that the connection is in the process of being redirected. In this case, the packet is forwarded to the replicated server after suitable modifications of destination address and the TCP/UDP checksums. The arrival of this packet might also be due to the fact that earlier redirection request or confirmation packets did not reach the client. If the state of the entry in the redirection table is IPRP_SENT, a duplicate redirection request is sent to the client and if the state is IPRP_CONFIRMED, a duplicate confirmation is sent.

Else, this packet is the first packet from a new client and the redirector begins a redirection attempt. A replicated server is chosen from the list, and the client is sent an IPRP_REQ message requesting redirection to that server. Then, a new entry is inserted in the redirection table for this client.

When the redirector receives an IPRP_CHK - a check message from a client requesting confirmation of a redirection request, the redirector checks if a matching IPRP_SENT entry is present in the redirection table. If there is a matching entry, the random key is copied from the check message into the entry. Then, an IPRP_CFM message is sent to the client that includes the key, and the state of the corresponding entry is changed to IPRP_CONFIRMED. If no entry is found, an IPRP_ALM message is sent indicating a fake redirection attempt at the client site.

The IPRP protocol also has a slow time-out function which is called periodically (every 500 ms). This function deletes entries from accept and redirection tables which have expired last activity timestamps. On the other hand, if only the last client response timer has expired, then the entry is moved to the accept table because that indicates that the client cannot support IPRP.

3.3 The Client

Like the redirector, the clients also maintain a *Redirection table* that has entries identifying clients have either been redirected successfully or are in the process of being redirected. The structure of an entry in the client's redirection table is similar to the redirector's entry shown in fig 4 except for the *Client IP Address* field is replaced by *Original Server IP Address* which maintains the IP address of the redirector. The *Status* field denotes the redirection state of the connection from the client's point of view, and is either IPRP_CHECKING (unconfirmed entry) or IPRP_CONFIRMED (confirmed entry). Besides its redirection table, a client also maintains a queue of buffers. These buffers are used to temporarily hold packets that arrive from a replicated server if the entry in the redirection table corresponding to that flow is not yet confirmed.

A client takes the following actions on receipt of different kinds of packets. When a client receives an IPRP_REQ from the redirector, it ignores this packet if there is an existing entry matching the received packet in the IPRP_CHECKING state. If a matching entry does not exist in the redirection table, a new one is created and marked IPRP_CHECKING. The client generates a random key, stores it in the *Key* field of the entry, and sends an IPRP_CHK message to the redirector with the key. This step is required to avoid malicious hosts masquerading as a redirector.

While a redirection is being confirmed, the client can potentially receive packets from the replicated server as a response to the initial packets sent by the client. The client IP layer buffers all such packets in a common buffer queue to reduce connection establishment latency. Buffered packets are released to the transport layer only when the redirection is confirmed to avoid security problems.

The client compares the keys in its redirection table and the received packet on receipt of an IPRP_CFM message. If the keys match, then the entry's state is marked IPRP_CONFIRMED. If the confirmation is a valid one, the client searches through the queue of buffered packets, and forwards any packets of this flow to the transport layer. If an IPRP_ALM message is received, the corresponding entry is immediately deleted from the table since it is a spurious one.

Similar to the redirector, the client IPRP protocol has a slow time out function which periodically checks the redirection table and expires entries that have not seen any activity for more than the timeout period.

Once a session's redirection is confirmed, the IP layer begins translating packets in either direction. If a confirmed binding is found for an outgoing packet, the destination address in the IP header is changed to the replicated server address in the entry and the checksums are appropriately modified. Similarly, an incoming packet belonging to that session from the replicated server is translated. The source address in the IP header is changed to the original redirector's address and the checksums are modified. Since all the translation happens in the IP layer, the client transport and higher layers are unaware of any underlying redirection, thus achieving the important goal of transparency. The translation mechanism is abstracted in figure 5.

4 Advantages and Disadvantages

This section compares our scheme with the existing methods described in section 2. It also describes some new features possible by using client based mechanisms and highlights its drawbacks. There are several advantages of using a client based IP Redirection scheme.

Incremental Scalability: In this architecture, more servers can be added to deal with extremely high demands by simply adding the address of a cooperating server to the list maintained

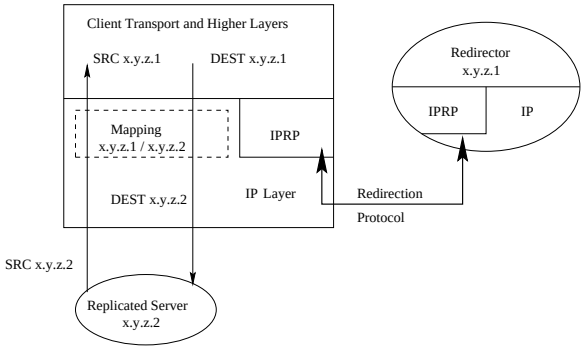


Figure 5: Post-Redirection Translation

by the Redirector.

Transparency: The redirection procedure itself is transparent to the clients since it is performed at the IP layer and even the transport layers are unaware of it. This implies that existing applications need not be modified to benefit from this method. For example, our implementation can transparently redirect regular Telnet and FTP clients to connect to and retrieve data from a replicated server.

Service Specific Redirection: This mechanism has the flexibility to allow service specific redirection. This implies that clients can be redirected to specialized servers depending upon the kind of service they request. For instance, an FTP request could be redirected to a host maintaining all FTP data and an HTTP request might be redirected to one maintaining all the HTML documents. It is not possible to do this in DNS based schemes since name resolution does not reveal the nature of the client application.

Cascaded Redirection: It is easy to extend this scheme to incorporate cascaded redirection. This means that the client can be repeatedly redirected transparently from one server to another if the server chooses to do so. Although a practical implementation of this feature has to deal with issues regarding the transfer of server side TCP state or application state, it is a distinctive feature that has many potential uses. It would simplify the transparent migration of services. A migrating service can first be installed on a host server, the clients connected to the old server redirected, and the old host can then transparently be decomissioned.

Since the central redirector is involved only in the initial redirection and does not process every packet, it does not become a bottleneck with increased load which is a problem faced by schemes where a single server does all the work. Also, since each client maintains its own list of redirection bindings, the Redirector needs to maintain bindings only for non-redirected clients which requires

much less memory compared to schemes where the server needs to maintain the mapping for each client accessing the site.

Unlike the Magic Router [7], this scheme does not place any topological restriction on the servers to be geographically localized. Since redirection is performed for each client, this scheme eliminates the problem of cached bindings being used by multiple clients causing skewed loads on the servers, as is the case in DNS based redirection [6].

An inherent advantage of performing redirection at the IP layer is that it is protocol or application independent. It is a general mechanism that an application can use without having a customized redirection mechanism in place. As is discussed in section 7, the mechanism can be massaged to serve well in a variety of scenarios.

Client oriented IP redirection has the following drawbacks:

Deployment: Since this scheme requires client operating systems to be modified to support redirection, It is relatively difficult to deploy. However, many popular operating systems like Windows NT, Windows 95 and Linux allow the TCP/IP stack to be loaded like any driver module without having to recompile or reinstall a new operating system. For these systems, a compliant stack can be released as an independent, upgraded module. Moreover, the scheme does not render the service unavailable to legacy clients, but they do not derive any special benefit from it either.

Connection Latency: Clients encounter longer connection establishment latencies with redirecting servers. However, the average delay is much less once the client is redirected to a nearer or faster server. A legacy client would not experience any additional connection establishment delay since the redirector would not redirect that client if the type of service field is used to indicate the lack of redirection support.

Processing Overhead: Since the host IP layer maintains the redirection bindings of local clients and implements the translation of source and destination addresses, all packets undergo some extra processing in the IP layer. This overhead can be minimized by efficient implementation and as is discussed in section 6, results show that the overhead is not substantial.

5 Implementation

For the purpose of taking measurements, a testbed was implemented which consisted of Pentium based PCs running FreeBSD, Release 2.2.2 [17]. One machine was configured as the redirector and the others as clients supporting redirection. The actual servers that the clients were redirected to were a mix of workstations running Solaris and PCs running FreeBSD or Linux. FreeBSD kernel was modified to support IPRP. This section highlights the key portions of the implementation of

the testbed. It mentions the main areas of the FreeBSD source that were modified. It also briefly discusses the client and redirector protocol module implementations.

5.1 FreeBSD IP Processing

In the FreeBSD kernel, the IP processing for an incoming packet is implemented in the *ip_input* function and the output processing in the *ip_output* function. The *ip_input* function is called by the device drivers, and is passed an *mbuf* containing the packet. An *mbuf* is a kernel data structure in FreeBSD used to manipulate packets [18]. Similarly, the *ip_output* function is called by any module that wants to send a packet over the network. A global *inetsw* array is used by *ip_input* to de-multiplex incoming packets and pass them to the appropriate protocol input functions [18].

For the clients, both *ip_input* and *ip_output* were modified to add support for redirection. For the redirector, only *ip_input* was changed. The redirection protocol itself was implemented by adding another entry to the *inetsw* array with pointers to the protocol input function *ipredr_input* and the slow time out function *ipredr_timo*. A protocol number of 99, defined by the constant IPPROTO_REDR, was used. Essentially, the redirection protocol was implemented like a higher level protocol but conceptually remains an extension to IP. The actions performed by these functions vary in the clients and the redirector.

Once a binding has been confirmed, the input and output IP layer processing of a client modifies the source and destination fields in the IP headers of packets of that session respectively. The redirector IP input layer also needs to modify the destination in the initial packets from a client and forward them to one of the replicated servers. Changing these headers requires that the the IP checksum as well as the TCP and UDP checksums be updated. Instead of recomputing the entire checksum, which is expensive, the checksums can be incrementally updated. A procedure is described in [19] for incrementally updating Internet checksums which was corrected and improved by [20] and [21]. Our implementation is based on the method described in [21].

5.2 Redirector Implementation

The most important data structures that a redirector maintains are its lookup tables - the redirection table and the accept table. To facilitate efficient insertions, deletions and searches, the tables are implemented as hash tables with doubly linked lists as collision resolution chains. The sizes of the hash table arrays are configurable parameters. Each entry uniquely identifies a session. Entries are hashed into the tables using the IP address of the client as the key.

Figure 6 shows the way different functions in the redirector kernel code interact with each other. An incoming packet is processed by *ip_input*. It uses the function *hash_search* to search for entries in the accept table and the redirection table. A redirection protocol message is passed

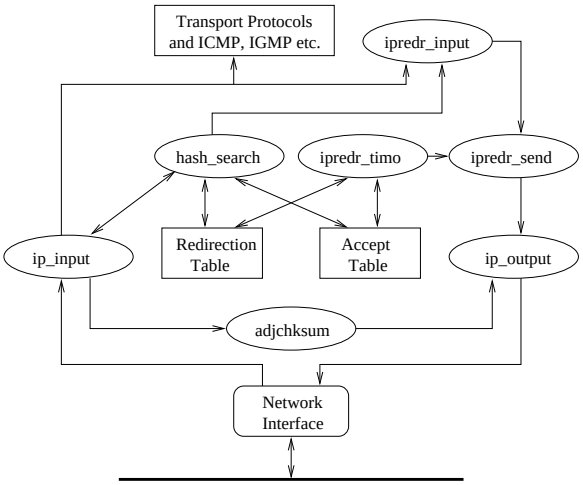


Figure 6: Redirector Implementation Functions

on to the protocol input function *ipredr_input*, which processes it as specified by the protocol. *ipredr_input* sends any redirection message through the *ipredr_send* function which calls *ip_output*. The function *ip_input* also may call *ip_output* after calling *adjchksum* to forward initial packets from the client to the replicated server. The function *ipredr_timo* is the protocol timeout function which is called periodically and deletes expired entries from the tables. It may also call *ipredr_send* for retransmitting redirection messages.

5.3 Client Implementation

A client's redirection table is also implemented as a hash table with doubly linked lists for collision resolution. Entries are hashed into the table using the replicated server's IP address as the key. Figure 7 depicts the way different functions in the client kernel code relate to each other. The interaction is quite similar to the redirector with some changes. However, the processing *inside* each of the functions is different in the clients and the redirector.

An incoming packet is processed similarly as in the redirector. The redirection table is keyed by the replicated server's address since only packets from a replicated server are mapped on input. Whenever a packet is passed to *ip_output* from the transport layers for output processing, the function uses the linear search method *linear_search* to check the redirection table for a binding. If a confirmed binding is found, the destination of the packet is mapped according to the field in the entry and the appropriate checksums are updated using *adjchksum* before the packet is sent to the interface driver. The output processing has to search the redirection table linearly instead of using a hash search method since entries in the table are hashed with the replicated server address

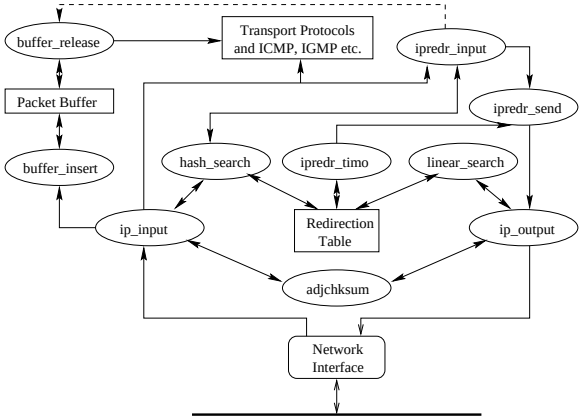


Figure 7: Client Implementation Functions

as the key and before actually searching the table, the replicated server to which this packets should be mapped is not known. This can be overcome by implementing the redirection table as a "two-dimensional" hash table.

The clients also maintain a queue to buffer packets that arrive from a replicated server before the corresponding entry is confirmed. This is implemented as a singly linked list represented by the global *bufq*. The *ip_input* function uses the function *buffer_insert* to insert an mbuf containing such a packet into the queue. Buffered packets are only released when the entry is confirmed through a confirmation message from the redirector and the protocol input function *ipredr_input* calls *buffer_release*. The function *buffer_release* simply removes the packet from the chain and passes it to the transport layers just like the IP layer would.

6 Performance Measurements

The performance of any applications based on IPRP will depend upon the overhead involved in redirection. The main focus of this section is to measure and discuss the overheads involved in IPRP based redirection.

6.1 Processing Overhead

The redirection mechanism requires support in the client IP layers which involves searches through the redirection table, periodic timeouts, protocol message processing and address translations etc. Since extra processing means additional delays, it is important to measure the IP input and output processing time for a redirected connection and compare it to that for a packet going through a

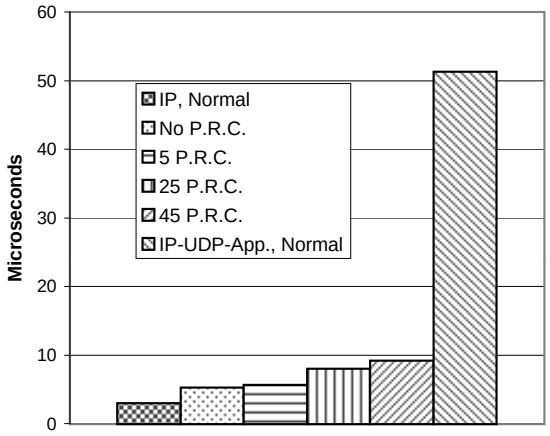


Figure 8: IP Input Processing Time for a Non-Redirected Session with Parallel Redirected Connections (P.R.C.)

regular stack. Since packets that belong to non-redirected sessions also have to go through some of this processing, the overhead introduced for incoming and outgoing packets for non-redirected sessions is measured as well.

Figure 8 shows the average time spent in the IP layer by an incoming packet of a non-redirected flow while there are multiple parallel redirected telnet sessions from the same machine. The left-most bar is the time a packet for such a flow would have spent in an unmodified IP layer, which serves as a comparison base. The second bar from the left is the time it takes even when there are no parallel redirected sessions. The difference of this value from the regular processing time is the minimum overhead every non-redirected packet goes through (2.33μs on an average), irrespective of the number of parallel sessions. As the graph shows, the overhead increases with increased number of parallel redirected sessions. This is expected since a higher number of redirected sessions implies that more entries in the redirection table need to be searched. However, the increase is not linear because hash based searches are fairly efficient. Even with 45 parallel redirected sessions (which is fairly high and is unlikely to be typical), the overhead is not substantial (6.23μs average). In-fact, the extra time spent in the IP layer is negligible when compared to the total time spent by the packet in going from the IP layer to the application layer, as shown by the right-most bar. Moreover, the right-most bar is the time through a UDP stack, which involves much less processing than in TCP. Overhead for a non-redirected flow can be further reduced by increasing the number of hash chains.

Figure 9 shows the overhead experienced by a packet from a non-redirected session in the IP layer output processing. The left-most bar is the average output processing time for a packet in

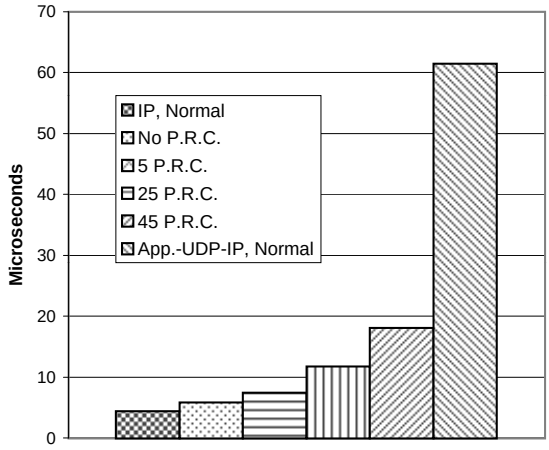


Figure 9: IP Output Processing Time for a Non-Redirected Session with Parallel Redirected Connections (P.R.C.)

an unmodified IP layer and the right-most one is the time taken in going from the application layer to the end of IP output processing in a redirection compliant kernel. The difference between the first and the second bars from the left is the minimum overhead involved (1.42μs average) and as expected, the overhead increases with increasing number of parallel redirected sessions. However, the overhead is higher for output processing than in input processing. This is because the redirection table is searched in a linear fashion during output processing since the key i.e. the replicated server address is not known a priori. A reduction in both the average overhead and the rate of increase in overhead can be achieved by implementing the client redirection table as a "two-dimensional" hashed chain structure. This would facilitate efficient searches during output also. The overhead is not substantial compared to the total time spent in the UDP/IP stack, and would be even less so for a TCP packet.

Packets that actually belong to a redirected session undergo more processing in both input and output at the IP layer than non-redirected session packets since they need to be modified. Figure 10 shows both the average input and output processing time for a redirected packet and compares it against the processing time in a regular IP layer. The time taken from the application to IP layer through UDP (and reverse direction, for input packets) in a supporting kernel is also shown in figure 10. These measurements are for a single redirected connection with no parallel redirected session running, and thus the overhead for the redirected packets represents the overhead due to packet translation only (average overhead of 3.70μs for input processing and 4.65μs for output processing). The overhead is not substantial and is worth the benefits that can be derived from redirection.

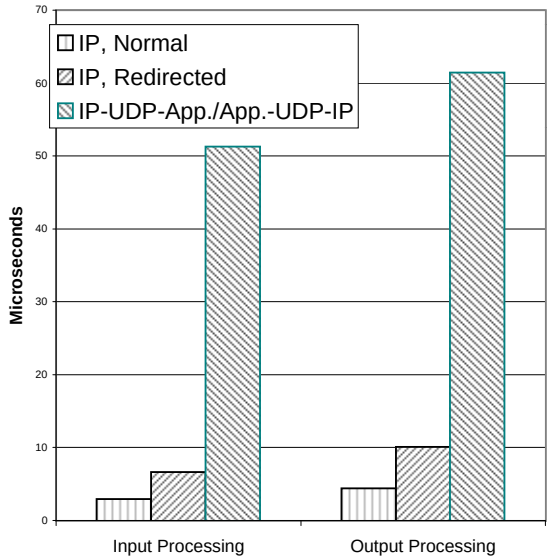


Figure 10: IP Processing Time for Single Redirected Session

6.2 Connection Establishment Latency

The redirector begins the redirection handshake for every new client. The extra processing and propagation delays involved in the message exchange introduces latency in the establishment of a connection. It is important to measure this delay since it is critical to the client's perception of a fast service.

6.2.1 Latency for Supporting Clients

Figure 11 plots TCP connection establishment times with varying redirector distance. In all cases, the replicated servers where the clients were redirected to were on the same subnet as the clients to isolate the impact of latency of redirection. Due to experimental setup restrictions, it was not possible to actually place the redirector and the clients far apart so forced delays were introduced in the redirector kernel to simulate propagation delays. Thus, distance is expressed in terms of the round trip time (RTT) between the client and the redirector.

In each of the clusters in figure 11, the left bar represents the regular TCP connection establishment time between a client and a server that are separated by varying distances. Although the establishment of a TCP connection involves a three-way handshake between the client and the server, the client considers the connection established when it receives an acknowledgment for the

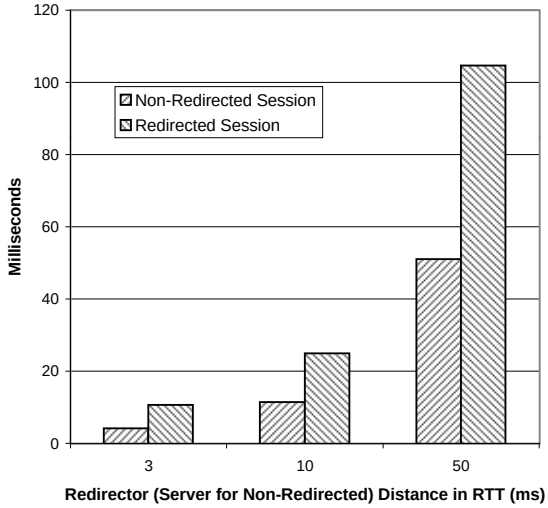


Figure 11: TCP Connection Establishment Time for Non-Redirected Session and Redirected Session with Forwarding and Buffering

initial SYN packet. This takes one round trip time and that is why the connection establishment latencies for a regular TCP connection in figure 11 are close to the RTT values for all server distances (the extra time is the processing delay). These measurements serve as the base case for comparison as these would be seen for any normal connection.

In figure 11, the redirected connection establishment times (with forwarding and buffering) also increase with increasing redirector distance as expected but remain of the same order of magnitude as the RTT. The measurements strongly bring out the benefit of forwarding and buffering. Without forwarding and buffering, the TCP timeout mechanism determines the connection establishment latency and hence typically resulted in a 3 second delay. Redirected connection establishment latency always exceeds that for a regular TCP connection by approximately one RTT. This is because redirection requires an extra check and confirmation message to be exchanged, which takes an extra round trip time. Redirection proceeds in parallel with connection establishment. Any packets from the replicated server before the redirection is confirmed are buffered at the client and thus connection establishment time is lower bounded only by twice the round trip propagation delay between the client and the redirector.

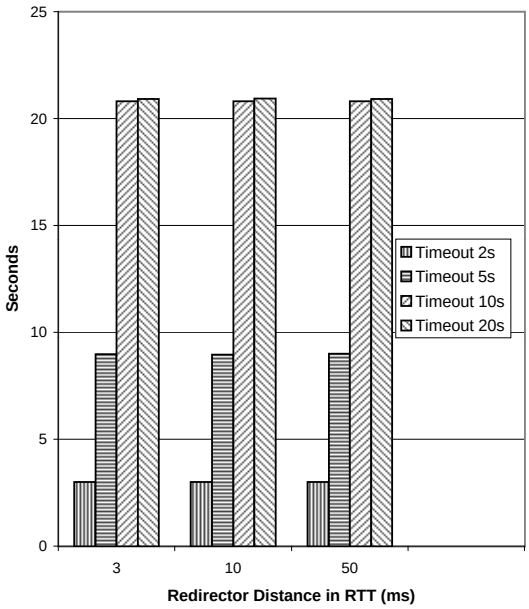
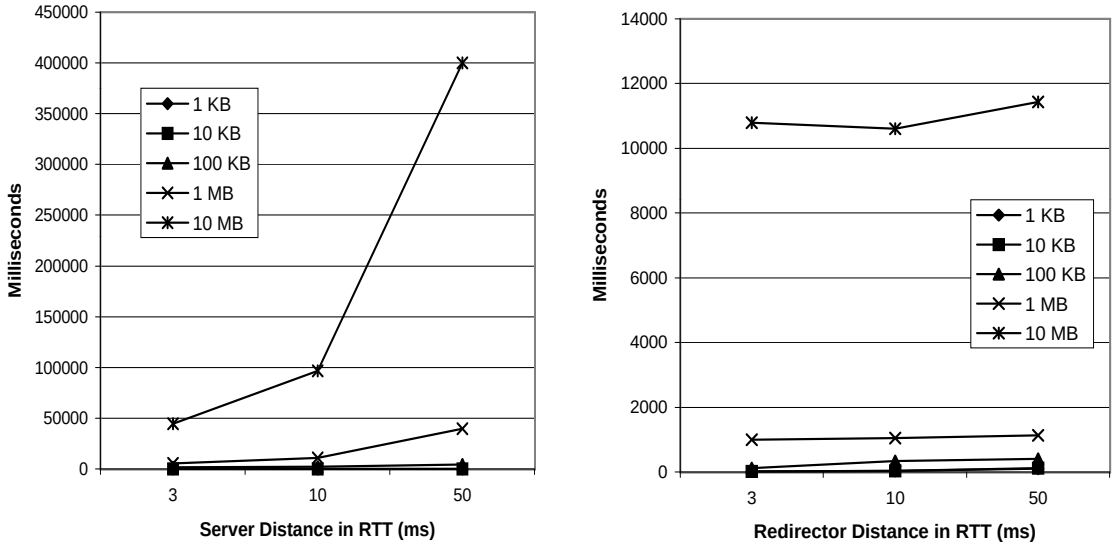


Figure 12: TCP Connection Establishment Latency for a Non-Redirectable Client with Varying Client Response Timeout

6.2.2 Latency for Legacy Clients

The redirector also attempts redirection for clients that do not support redirection and accepts the service requests itself when it realizes that the client is a legacy client. This happens through the value of the *last Client Response* field in the redirector's redirection table. The timeout value can be configured by the site administrator. Figure 12 shows the connection establishment latency of a non-supporting client requesting service from the redirector. Since TCP retransmits the initial SYN segments after approximately 3ms, 9ms and 21ms, the latency is the maximum of the timeout value and the retransmission period for a TCP connection.

An improvement over this approach is the use of the *type of service* field in the IP header to indicate whether a client supports redirection or not. The redirector then would not attempt redirection for a legacy client. This would result in legacy clients seeing no extra connection establishment latency.



(a) Non-redirected (b) Redirected

Figure 13: Bulk Transfer Time Over a TCP Connection

6.3 Bulk Transfer Time Measurements

Figure 13(a) shows the time taken to transfer files of varying sizes from a remote server by an unmodified client over a TCP connection. The transfer times are less than a second for smaller files but increase sharply as the file size and the server distance increases. In fact, for a server for which the RTT is 50 ms, it takes time of the order of several minutes to download a 10 MB file. This is because each packet experiences higher propagation delay with increasing server distance.

Similarly, figure 13(b) shows the bulk transfer times over a redirected connection. The redirector distance is varied but the replicated server is always maintained on the client's network to study the impact of redirection latency. This is a best case analysis since the replicated server could be distant. The figure shows marked improvement over the non-redirected case. The transfer times are slightly higher for smaller files since the overhead involved in redirection and translation is more than the transfer time itself. However, as the file size and the redirector distance increases, the transfer times are much less than with no redirection. This shows the benefits of redirection and providing service close to the client.

7 Applications

This section discusses some useful applications that a client based IP level redirection scheme can be cast into and some future enhancements that are possible. As mentioned earlier, IP level redirection is a fairly general mechanism that can be massaged to fit into different scenarios. We outline a number of possible applications of IP level redirection.

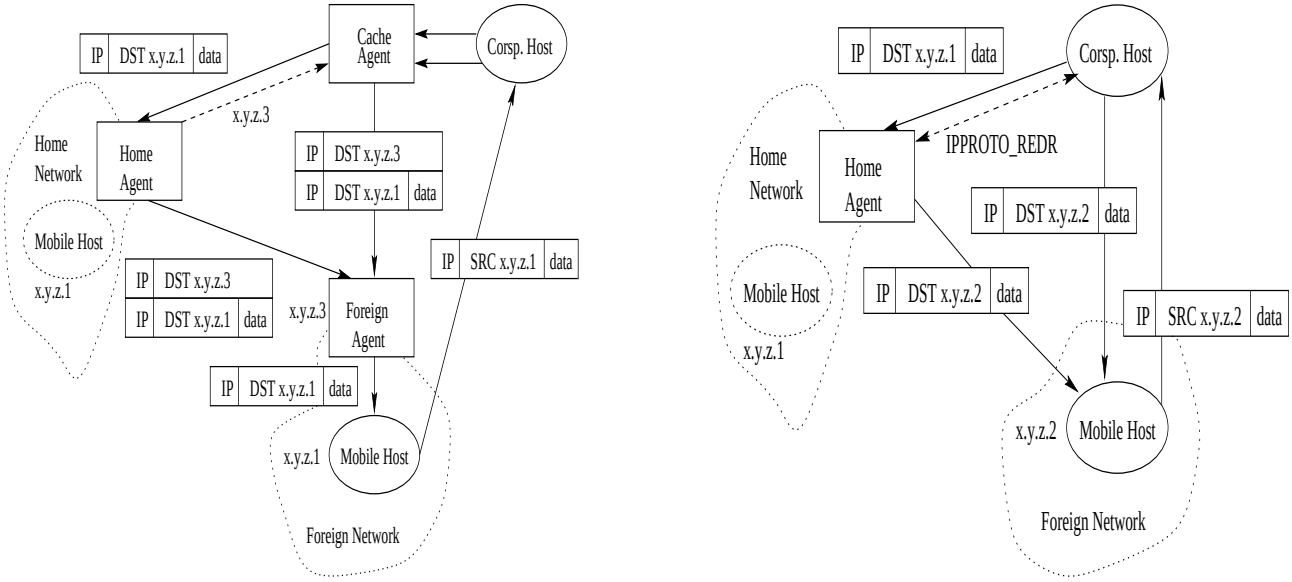


Figure 14: Support for Mobility

7.1 Mobile IP Implementations

Most proposed schemes and implementations share a common approach to IP mobility. A *mobile host* has a constant *home IP address* regardless of its current location. A *correspondent host* is a host communicating with the mobile host, which could itself be mobile. The *home agent* is a stationary entity which is responsible for keeping track of the mobile host's current location. A *foreign agent* is an entity assumed to be present in every network that the mobile host visits, which is aware of all guest hosts in its network and also serves as a temporary *point of attachment* for them.

A mobile host uses the IP address of the foreign agent of the network it is presently in as its

proxy address and informs its home agent of this address. Since a correspondent host is unaware of the location of the mobile host, it sends packets on the home IP address of the host. The home agent intercepts these packets and *tunnels* them to the foreign agent using IP-in-IP encapsulation [22]. The foreign agent decapsulates the packet and passes it directly to the mobile host using its physical interface address. Packets from the mobile host can go directly to the correspondent host. A basic scheme such as this suffers from dog-leg routing for packets from the correspondent host to the mobile host. To avoid dog-leg routing, protocols like the Internet Mobile Host Protocol (IMHP) [23] suggest the use of *cache agents*. These are entities which reside either on the correspondent host (if the host is IMHP compliant) or on intermediate routers. Cache agents maintain bindings about the current point of attachment of mobile hosts and instead of forwarding packets to the home agent, perform encapsulation themselves and send packets to the foreign agent directly. This is depicted in figure 14(a). This eliminates the dog-leg to a certain extent.

An implementation based on IP redirection would work as follows. Every time a correspondent host tries to contact a mobile host at its home address, the home agent, which has the functionality of a redirector, invokes the redirection handshake to create a binding at the client. Thus the correspondent host IP layer translates future packets destined for the home IP address to the current location of the mobile host, eliminating the dog-leg in the forward direction. The mobile host sends outgoing packets with its current address as the source. These go directly to the correspondent host where they are translated before being handed to the upper layers to seem as if they came from the home IP address. This mechanism would work even if the mobile host joins a network with a firewall since packets in either direction use the local, dynamically assigned address. Figure 14(b) shows this idea. Another advantage of our implementation over earlier schemes is that packets are not encapsulated with another IP header, but the header itself is translated. The procedure is more efficient than encapsulation, does not increase the size of the packet, and avoids unnecessary additional fragmentation.

7.2 Reliable Multicasting

RMTP [24] is a reliable multicast protocol based on a hierarchical structure in which receivers are grouped into local regions and each region has a special receiver called the *designated receiver*. The designated receiver sends acknowledgments periodically to the sender, processes messages and acknowledgments from receivers in its region and retransmits lost packets to them. Since retransmissions are done locally, latency is low and as the sender only receives a common acknowledgment from the designated receiver, *acknowledgment implosion* is not a problem. Receivers choose the closest designated receiver through periodic polls and if a designated receiver dies, the next closest one is chosen. RMTP requires that clients be made aware of designated receivers and support the extra functionality required to determine the closest live designated receiver.

An IP level redirection based approach, as shown in figure 15, works as follows: The receivers

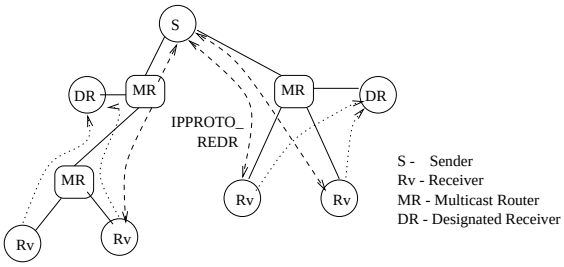


Figure 15: Hierarchical Reliable Multicasting

are initially aware only of the sender. When the first response from a receiver arrives at the sender, it invokes the redirection protocol and redirects the receivers to send their future responses and retransmission requests to the designated receiver closest to them, which the sender determines. The receivers don't send any packets to the designated receiver explicitly, but that happens transparently. Another advantage of doing this is that if the current designated receiver decides to leave the group, it can redirect the receivers in its domain to another or next-in-line designated receiver before leaving. The receivers would not detect any changes and would still send requests and acks to the sender's address. This illustrates that the same IP level redirection protocol can be used to enhance a multicast protocol into a hierarchical multicast protocol.

7.3 Interactive On-Line Games

On line games between geographically separated players are very popular on the Internet. There are a number of sites that users can log on to and play against other present users. In almost all applications though, all packets between the two players have to go through the site. However, if two users want to play a highly interactive game on-line, and if there exists a direct, shorter path between them, they could experience a much better playing environment if packets were to traverse directly to and fro, avoiding the additional delay in going through the central site. The players should not be asked to reconnect to their opponent, though, so transparency is important. All this is possible if the clients support redirection since the redirector can be configured to redirect opponents to each other. Naturally, the gaming applications have to be designed to use this facility.

7.4 Location Independent Video Transmission

IPRP allows an application to be redirected to a different server while the application is in progress (with necessary support at the servers). The cascaded redirection feature possible with this scheme allows a user to be transparently redirected repeatedly if required. For example, a user can contact a central site for live video. When the live video needs to be transmitted from a different location (from a site in city A to another site in city B in a new year celebration video, for example), the

client can again be redirected to expect the packets from the second server. This is not possible in most live video transmissions on the Internet currently, which are restricted to one location primarily. This idea can be used to provide location dependent services in a mobile environment by redirecting the client to a different server as he/she moves around.

7.5 Layered Video Transmission

Some schemes for streaming video on the Internet use a layered encoding mechanism where a base encoding layer provides the minimum acceptable picture quality and an enhancement layer improves the quality. Typically, the base and enhancement layers are streamed over parallel sessions but from the same server. This can cause congestion on the network since video streams requires large amounts of bandwidth. An IPRP based scheme could redirect the sessions such that the two layers are delivered by two different servers. The streams take different paths to the client reducing the chances of congestion and improving the overall throughput. The application would however have to take care of synchronization issues.

8 Conclusions and Future Work

The paper introduced the idea of involving the client IP layer actively in redirecting a local application. This approach is different from most other redirection approaches which rely entirely on the server or the network. The main drawbacks of this approach have been discussed and ways to overcome the problems have been suggested. The performance of the protocol is measured in terms of bulk transfer times and connection latencies. The impact of a redirection-supporting kernel and multiple parallel redirected sessions on regular connections is also studied. The results show that the overhead introduced in the client IP layers due to redirection is not significant. The connection establishment latencies are also acceptable for redirected connections because of redirector forwarding and client buffering. Latency for a legacy client can also be minimized by using the TOS field. The paper also presented various possible Internet based applications where a client based network level redirection scheme can be used.

There are ways that the current implementation can be made more versatile and efficient. The redirector can be modified to periodically poll the replicated servers to detect if they are alive. If it detects a dead server, it could try and redirect clients that were initially sent to this server. Adding this would involve issues of service migration.

A "wild-card" redirection mechanism can be incorporated. This refers to the situation where the redirector might want to redirect *all* clients on certain hosts to a particular server. Instead of redirecting each client individually, redirection could occur only once. Moreover, if the client IP detects an existing wild-card entry for a site, it should start mapping packets for a new session

automatically, without even sending the first packets to the original site and triggering another redirection handshake.

In the current implementation, entries in the client redirection table are hashed using the replicated server address as the key. As suggested earlier, output processing can be made more efficient by organizing the redirection table as a two-dimensional hash table, which would allow searches using either the redirector's address or the server's address as a key. Current implementation only supports a weak security model which depends upon the exchange of a random key. It is still open to attacks from malicious hosts that might be snooping on the network and who might pick up the random key. A better implementation could support a more secure authentication mechanism.

Another possible enhancement could be the use of the type of service field in the IP header for clients to indicate to the redirector if they support redirection. The redirector would not attempt redirection for a client that does not support it, thus avoiding higher connection establishment latency for the client.

A number of new applications as suggested in section 7 are being currently implemented to validate the versatility of IPRP.

References

[1] J. Postel, "RFC 791: Internet Protocol," Sept. 1981. ftp://ftp.internic.net/rfc/rfc791.txt, accessed on July 15, 1997.

[2] R. W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, Menlo Park, CA, 1994.

[3] P. Mockapetris, "Domain names - implementation and specification," Tech. Rep. RFC 1035, Network Working Group, Nov. 1987. http://info.internet.isi.edu/in-notes/rfc/files/rfc1035.txt, accessed on July 25, 1997.

[4] D. B. Terry, M. Painter, D. W. Riggle, and S. Zhou, "The Berkeley Internet name domain server," in *Proceedings of the USENIX Association and Software Tools Users Group Summer Conference: Salt Lake City, UT, USA* (Software Tools Users Group, ed.), pp. 23–31, Summer 1984.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "RFC 2068: Hypertext Transfer Protocol - HTTP/1.1," Jan. 1997. ftp://ftp.internic.net/rfc/rfc2068.txt, accessed on Aug 1, 1997.

[6] E. Katz, M. Butler, and R. McGrath, "A scalable HTTP server: The NCSA prototype," tech. rep., National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign, Il, 1994. http://www.ncsa.uiuc.edu/InformationServers/Conferences/CERNwww94/www94.ncsa.html, accessed on July 20, 1997.

[7] E. Anderson, D. Patterson, and E. Brewer, "The magicrouter, an application of fast packet interposing." http://www.cs.berkeley.edu/~eanders/magicrouter/osdi96-mr-submission.ps, accessed on July 21, 1997.

[8] Local Director, http://www.cisco.com/warp/public/751/lodir/literature.shtml, accessed on Sept. 10, 1997.

[9] C. Attanasio and S.E.Smith, "A virtual multiprocessor implemented by an encapsulated cluster of loosely coupled computers," tech. rep., IBM RC18442, April 1992.

[10] Distributed Director, http://www.cisco.com/warp/public/751/distdir/literature.shtml, accessed on Sept. 10, 1997.

[11] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, pp. 80–86, Jan. 1997. http://www.tns.lcs.mit.edu/publications/ieeecomms97.html, accessed on July 20, 1997.

[12] D. Wetherhall and D. Tennenhouse, "The active IP option," in *7th ACM SIGOPS European Workshop, Connemara, Ireland, September 1996*, Sep 1996.

[13] H. Chawla and R. Bettati, "Replicating IP services," tech. rep., Department of Computer Science, Texas A&M University, September 1997.

[14] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler, "Using smart clients to build scalable services," in *1997 Annual Technical Conference, January 6-10, 1997. Anaheim, CA* (USENIX, ed.), (Berkeley, CA, USA), pp. 105–117, USENIX, Jan. 1997.

[15] J. Postel, "RFC 792: Internet Control Message Protocol," Sept. 1981. ftp://ftp.internic.net/rfc/rfc792.txt, accessed on Oct. 11, 1997.

[16] W. Fenner, "RFC 2236: Internet Group Management Protocol, version 2," Nov. 1997. ftp://ftp.internic.net/rfc/rfc2236.txt, accessed on Oct. 11, 1997.

[17] FreeBSD, http://www.freebsd.org, accessed on Sept. 3, 1997.

[18] W. R. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*. Addison Wesley, Menlo Park, CA, 1994.

[19] B. Braden, D. Borman, and C. Partridge, "RFC 1071: Computing the Internet checksum," Sept. 1988. ftp://ftp.internic.net/rfc/rfc1071.txt, accessed on Nov. 15, 1997.

[20] T. Mallory and A. Kullberg, "RFC 1141: Incremental updating of the Internet checksum," Jan. 1990. ftp://ftp.internic.net/rfc/rfc1141.txt, accessed on No. 15, 1997.

[21] A. Rijsinghani, "RFC 1624: Computation of the Internet checksum via incremental update," May 1994. ftp://ftp.internic.net/rfc/rfc1624.txt, accessed on Nov. 15, 1997.

[22] W. Simpson, "RFC 1853: IP in IP tunneling," Oct. 1995. ftp://ftp.internic.net/rfc/rfc1853.txt, accessed on Oct. 15, 1997.

[23] C. Perkins, A. Myles, and D. Johnson, "The Internet mobile host protocol (IMHP)." ftp://ftp.mpce.mq.edu.au/pub/elec/dist/mobile/imhp/jenc94/inet94.ps, accessed on Apr. 10, 1998.

[24] J. C. Lin and S. Paul, "RMTP: A reliable multicast transport protocol," in *IEEE INFOCOM 1996*, pp. 1414–1424, 1996.