The following paper was originally published
in the Proceedings of the
Tenth USENIX System Administration Conference (LISA X)
Chicago, IL, USA, Sept. 29 - Oct. 4, 1996

IP Multiplexing by Transparent Port-Address Translator

Heon Y. Yeom, Jungsoo Ha, and Ilhwan Kim
Seoul National University

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

IP Multiplexing by Transparent Port-Address Translator

Heon Y. Yeom, Jungsoo Ha, & Ilhwan Kim – Seoul National University

ABSTRACT

There are certain occasions when we want to have more than one computers using a single IP address. One notable case is when we use firewalls for security reasons or when we use private IP addresses and still want to provide global IP connection. Since the computers behind the firewall or using private IP addresses do not have unique IP addresses, they can not participate in global IP network. However, we still want to provide them some means so that they can be used as if they have real IP addresses.

Another example is when we have more than one server and want to make them appear to be just one server. With the proliferation of WWW, popular WWW servers tend to be overloaded and we need a mechanism to balance the load to several servers when needed. One solution being adopted is address translation by DNS (Domain Name Service). IP reuse using network address translator (NAT) is proposed in [TE93] and standardized by RFC1631. For load balancing of the WWW servers, a round robin distribution scheme using DNS is proposed and is used at NCSA [KBM94]. However, as pointed out from the literature, these schemes are based on DNS and there are several shortcomings. We propose a novel technique to perform address translation without a DNS. Our technique is based on the idea that any internet connection is uniquely identified by the socket pair <source IP address, source port, destination IP address, destination port>. By using a address translating bridge, we can provide computers with phony IP addresses real IP connection. The performance of our scheme is investigated using a prototype and the test results show that there is very little overhead in using our scheme.

Introduction

One of the biggest problems we are facing now on the Internet is the depletion of IP addresses. The four byte long IP address is being used up rapidly due to the exponential growth of Internet. We need to either expand the IP address space or devise an efficient way to use the existing IP address space.

Even though a new protocol called IPNG is being investigated to replace the IP, its wide acceptance is still far away and a short-term solution is badly required. We looked at some solutions suggested in the literature such as proxy, socks, and NAT. Among those solutions, NAT is considered to be the most practical since it does not need any changes in the network structure or current applications.

We propose a novel idea which we call port-address translator (PAT) which deals with the shortcomings of NAT that it still need IP addresses as many as the number of active local nodes at any time. By using PAT, only a single IP address is needed and it can support up to 2^{16} active connections.

The main idea behind PAT is that each TCP or UDP connection is uniquely identified by the internet address and port number pairs of the source and destination. Instead of having different IP addresses, one machine with a real IP address can act as a gateway by assigning different port number for each active connection.

To investigate the usability of the proposed PAT, we have conducted some experiments using our prototype. First, we investigated whether our PAT prototype can be used for all the protocols. In other words, the question is that the PAT is really transparent or not. We have found that PAT works well with most of the protocols except the protocols which use the endpoint information at the application level such as FTP, Rlogin and ICMP. In these protocols, the boundary between the communication layers is not clear and the lower layer information is used at the higher layer. To make PAT work with these protocols, we have to attack each protocols one by one and provide application specific services and we have solved the problem with FTP.

Secondly, we have examined the performance of our prototype and found that our prototype shows little overhead when used between two Ethernet segments.

As an additional benefit, PAT can be used as a load balancing gateway. Unlike the DNS based load balancing scheme which fails to balance the load from time to time due to DNS caching, PAT can distribute the loads to multiple servers perfectly at all times.

The rest of this paper is organized as follows: we identify the problem of IP address shortage and look at some of the solutions provided in the literature in section 2. In section 3, we propose our solution by translating IP address-port pair. Design and implementation issues are discussed in section 4 and we report the results of our performance experiments in section 5. Finally, we conclude the paper in section 6.

Background

Waste of IP Address Space

Since there are four bytes in IP address, there can be 2^{32} distinct addresses. However, this address space is wasted due to the following reasons.

- To facilitate routing, IP addresses are grouped into network addresses.
- There are three different classes of networks classified according to their size (class A, B, and C) and each organization is assigned one or multiple contiguous network addresses.
- IP addresses inside a network address can not be used by other organizations.
- The number of nodes in any one network which simultaneously communicate with outside nodes is relatively small.

Think of a class C network which has 150 IP nodes in it. At a glance, at least 106 addresses are wasted because the unused addresses of a class C network cannot be used by the other sites belonging to another network. Moreover, if only 50 local sites tend to access remote sites simultaneously, we don't have to assign global addresses to each local sites all the time.

Thus, we insist that 206 out of 256 IP addresses are wasted in this example. Francis' NAT [TE93] can support this network with only 50 global addresses. In this paper, we present another application level gateway that can support this network with just one global address.

IP address Reuse

It has been already noted that IP addresses can be reused for private networks. In [RFC1597], parts of IP addresses are designated to be used repeatedly for private networks as follows:

- 1 class A network address 10.0.0.0
- 16 class B network addresses from 172.16.0.0 to 172.31.0.0
- 255 class C network addresses from 192.168.0.0 to 192.168.255.0

However, for nodes inside these private networks to work with Internet, there should be some mechanism to give them real IP addresses. Also, in [RFC1519], it has been suggested to use CIDR (Classless Inter-Domain Routing) to avoid IP address waste due to IP partitioning.

Proxy Server

For a local area network where network security is important, a firewall is often used to separate the inside network from outside networks. It is usually implemented by selectively filtering packets coming in and going out. However, the users inside the network still want to use all the Internet services, and it is a major consideration in using a firewall.

One easy solution is to provide a proxy server through which users communicate with outside network. By setting up a proxy server which can be trusted and assigning a real IP address to it, users can go through it when they need outside connection. Refer to Figure 1 for the operation of a proxy server.

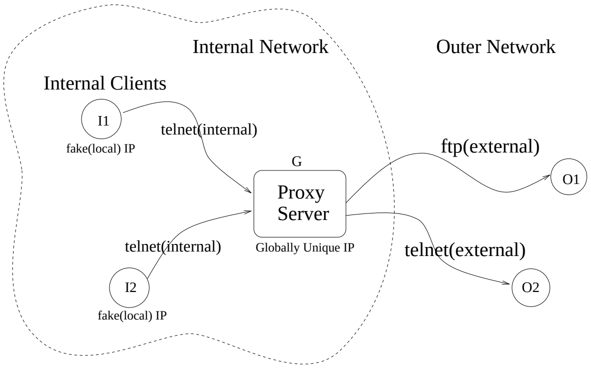

Figure 1: Connecting through a Proxy Server

Socket Server

In UNIX systems, application programs use sockets to access the network and each kernel of the system manages sockets. There are some exceptions where sockets are managed and allocated by a separate server to provide transparent network access where firewall is used. In [KK92], a socket server is introduced which manages the socket of the local machines which do not have direct internet access so that local machines can access outside network. The socket server should have the capability to communicate with outside network and it provides sockets which can be used for outside connection to the internal hosts which can not directly communicate with outside network. Internal nodes can get a socket from the socket server using Rconnect() call. The socket server provides the socket clients its IP port number and transfer packets through the port. Using the socket server and the socksified clients, it is possible to transfer packets between the internal and outside networks.

Even though this scheme provides transparent solution, it requires modification to all the client programs.

Network Address Translator

As we have seen earlier, it is impossible to connect from a regional network using replicated IP address to the global IP network without some tweaking. It is needed to translate the replicated address to a globally unique address to be able to connect to the global IP network since the regional addresses are only unique inside that region. We call these fake IP addresses as replicated addresses.

P. Francis has suggested a scheme for IP address reuse by transparent address translator using DNS in [TE93]. Figure 2 shows the network diagram of a regional network with replicated addresses and the usage of address translator.

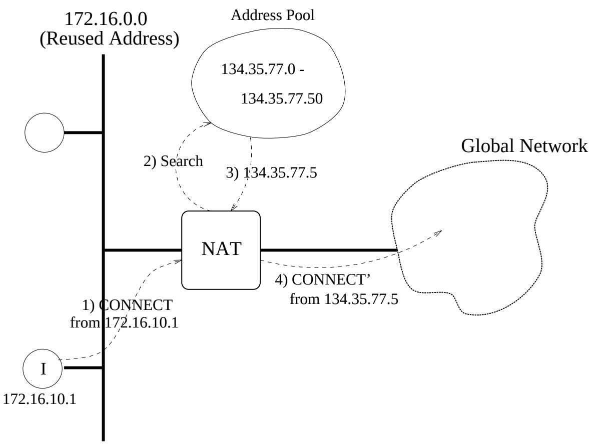

Figure 2: A network with replicated address and NAT

Between the global network and the local network using replicated IP addresses, there is a Network Address Translator (NAT) which has a couple of IP addresses which can be used for outside communication. Whenever a node inside the local network need a global address for outside communication, NAT provides the local node a global address dynamically. A global address, once allocated, can be used until the connection is terminated and the NAT automatically translates between the global address and local address. NAT performs the translation by intercepting all TCP/IP packets to search their header information and refer to the mapping table between the global and local addresses.

As shown in Figure 2, there is a close relationship between NAT, DNS and interdomain router or gateway. Let us look at how NAT works. When an inside node I wants to send a packet to outside network, NAT intercepts this packet and allocates a global IP address and relays it outside. All the packets to the outside network go through the NAT and the packets are examined and proper address translation is performed if needed. Only properly translated packets can be relayed to the outside network.

On the other hand, if there is a connection request from outside to the inside node I, DNS would first notice it and ask the NAT to prepare a global IP address for I. However, there is a set limit on the number of simultaneous outside connection which is the number of reserved global IP addresses the NAT has.

There are certain disadvantages in this scheme.

- It can be applied only to the applications using DNS.
- It is hard to apply it to connectionless protocol such as UDP.
- The number of global IP addresses might not be big enough for outside connections.
- A special consideration is needed for applications which exchanges IP address via the protocol. (FTP, RLOGIN, etc).

In spite of these drawbacks, this scheme enables a transparent address translation by setting up the NAT on the border of the network without a lot of modification. RFC1631 describes the use of NAT.

Transparent Port-Address Translator

Now, we introduce a new technique to re-use address space by transparent port-address translation. We note that there are a lot of ports in a node and only a small portion of them are used at any time. It is possible to provide outside connection by translating local address and port pair to a global IP address and its unused port number.

Let us denote a socket of a node as (address, TCP port number) and a TCP packet as (srcIP, srcPORT, dstIP, dstPORT). Assuming that there is no node with the same IP address on the route, every application using the connection can be uniquely identified in the regional network – in the global internet if there is no replicated addresses – by the pair (srcIP, srcPORT). Furthermore, two pairs of sockets, ((srcIP, srcPORT), (dstIP, dstPORT)) uniquely identify the two endpoints of the communication [RFC793].

Information Needed To Transmit Packets Through Proxy Server

Let us first find out the minimum information to transmit packets between TCP peer entities when proxy server is used. Let us look at Figure 3 which shows a telnet session through proxy server. From node S with replicated address, a remote login connection is established by issuing 'telnet G'. Again from G, 'telnet D' is used to connect to D and we have a indirect telnet connection from S to D. Proxy server G relays all the packets between S and D. As shown in Figure 3, a packet from `(S, 1000)` to `(D, 23)` is received by G and the source address is changed into `(G, 3000)` and then relayed to `(D, 23)`. A packet from `(D, 23)` to `(G, 3000)` is again received by G and it is relayed to `(S, 1000)` after the destination address is changed. This translation is done by the proxy server when the user specified the usage of proxy server. Now, all we have to do is to transparently perform the translation done by the proxy server G.

Operation of Port-address Translator

We can automate the translation the proxy server does by using a special gateway. We call the gateway a Port-address translator (PAT). Figure 4 shows the operation of the PAT.

Since a port number is represented using 16 bits, an IP node can have up to 2^{16} different sockets. Therefore, there can be 2^{16} different entities on an IP node which can be uniquely identified in the Internet address space. In practice, only a small amount of ports are ever being used by applications. The remaining unused ports of a node with global IP address can be allocated to the regional nodes which have replicated addresses.

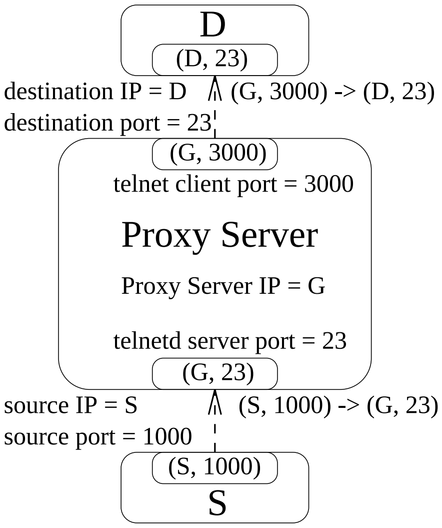

Figure 3: Packet relay by proxy server

G.PORT	I.IPaddress	I.PORT	STATUS
30000	172.16.10.1	1234	SYN-SENT
30001	172.16.10.1	3487	ESTABLISHED
30002	172.16.30.27	12039	CLOSE-WAIT
30003	NULL	NULL	CLOSED
30004	NULL	NULL	CLOSED
:	:	:	:
40000	172.16.68.99	3465	ESTABLISHED

Table 1: An example of port-address mapping table

Let us look at Table 1. This table shows the mapping between sockets `(IPaddr,PORT)` of nodes inside a regional network with stub B class network address 172.16.0.0 and the port numbers of a node G (Gateway node) which has a global IP address. This mapping can be dynamically allocated by:

1. DNS call as in [TE93]
2. Looking for SYN flag in TCP header information and deallocated when the STATUS becomes CLOSED.

Whenever PAT G encounters a packet requiring address translation, this table is looked up to modify the header information and the modified packet is relayed. G monitors all inbound packets and outbound packets to properly modify them.

The source address of the outbound packet is mapped from `(I.Addr, I.PORT)` to `(G.Addr, G.PORT)` according to the port-address mapping table and relayed. Also, inbound packets with destination address `(G.Addr, G.PORT)` is translated into `(I.Addr, I.PORT)` and relayed to the inside regional network. All the outbound packets should be translated before going out of the local network since the local IP addresses are replicated.

PAT as Load Balancing Gateway

With the growth of the internet, certain information servers tend to attract a lot of users and providing service to large number of internet users impose great strain on the popular servers. If the server is overloaded, the server should be upgraded or another server should be added to share the load. When using more than one servers, the question of how to distribute the load always comes up. The most widely used scheme is using DNS based load balancing. However, this scheme does not always work properly due to DNS caching. Using PAT, on the other hand, we can always distribute the load however we want. The load can be distributed round-robin fashion or the server load can be considered by simple programming of the PAT. Figure 5 shows the operation of PAT doing load balancing.

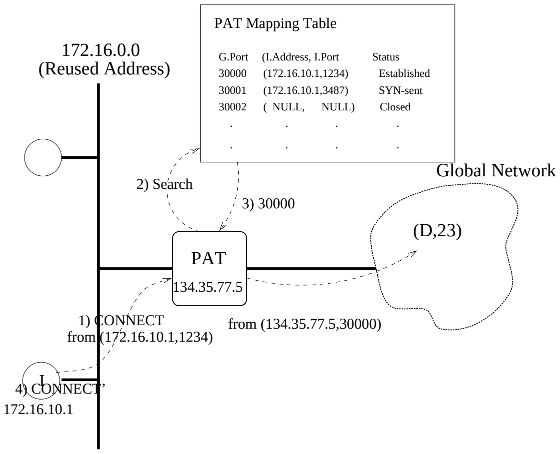

Figure 4: PAT operation as IP address multiplexer

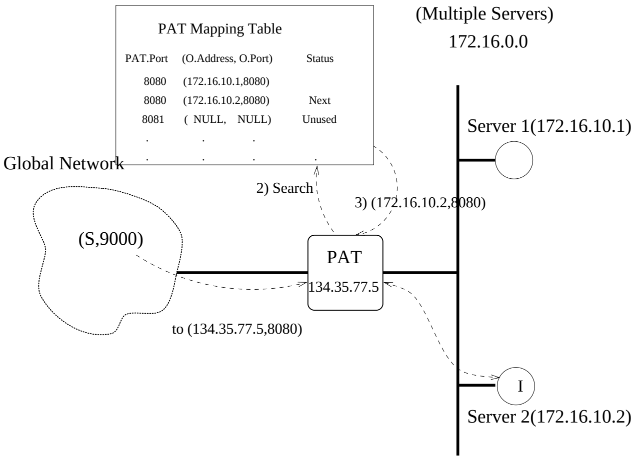

Figure 5: PAT operation as load balancing gateway

Design and Implementation Issues For Port-address Translator

We have implemented a prototype on an IBM compatible PC using packet driver. A lot of skeleton codes are adapted from PC bridge and PC router softwares. Since the port-address translator has more functionality such as address translation, more hardware resource is needed. From the experiment we have done using the prototype, we found that the performance is comparable to that of general PC ethernet bridge.

Some of the issues we have encountered are as follows.

- The checksum of the TCP/IP header should be modified accordingly.
- It is difficult to decide when to allocate and deallocate mapping table entity for UDP protocol.
- A separate mechanism such as static allocation is needed for inbound request to the server with only regional address.
- To deal with application protocols which exchanges packets with IP address in them such as FTP and RLOGIN, the translation should be done on application layer.

These problems are due to the fact that the port-address translator can only gather limited information from the packets. They are similar to the problems encountered with packet filtering gateway which is used to enhance security. However, they are more difficult than the case with the translation using socket server since the information can be obtained from Rbind() call in socket server whereas the port-address translator only looks at the header information.

Calculating checksum for TCP/IP header is pretty simple and it can be efficiently done by a combination of additions and subtractions [RFC1631] [TE93]. We have implemented our prototype using the algorithm shown in [TE93].

Timing for Allocation and Deallocation of Port-address Mapping Table Entry

Two methods have been introduced for the timing of allocation and deallocation of port-address mapping table entry.

First, we can use DNS to dynamically allocate addresses. It has advantages that the address allocation time is specified and inbound service can be supported. However, it tends to be more complex since the address deallocation should be done either by monitoring each session or explicitly specifying the deallocation. Furthermore, for this method to work, all the nodes should be registered to the DNS and DNS caching can cause a severe inconsistency problem.

Second, we can monitor the packet header and its contents and modify them accordingly. It could be complex since we need to trace the regional sockets to follow their TCP STATE change. However, this information can be obtained from the TCP header flag (e.g., SYN, FIN). Figure 6 shows the state transition diagram of the TCP STATE and corresponding header flags. A pseudo code of the algorithm is attached as an appendix.

Supporting UDP

It is more difficult to extend the port-address translator to support UDP protocol. Since UDP is a stateless protocol and there is no sequence number, it is close to impossible to correctly trace a session. What we can do is to use an appropriate timeout value to determine a termination of a session. This idle time threshold based algorithm does not guarantees to work since the proper timeout value is hard to determine.

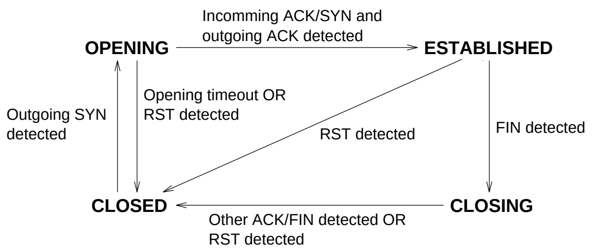

Figure 6: TCP STATE transition diagram from TCP header flag

We have implemented the idle time threshold based algorithm with a timeout value of two minutes. This value is selected since it is relatively large compared to the other timeout values used in NIS (Network Information System) or NFS (Network File System).

A special caution is needed to support UDP applications with different characteristics.

Another problem we have encountered with UDP protocol is the problem with large packets. When a UDP packet larger than 1472 bytes is transmitted through the Ethernet, it is divided into several packets before transmitted. The problem is that only the first packet has the UDP header. The remaining packets do not have the UDP header and the PAT has no way of knowing where to send these packets with no UDP headers. Since the first packet has the continuation flag, PAT can examine the flag whenever it receives a UDP packet and takes appropriate action when the packets with no header follow.

Supporting Inbound Request

Besides the dynamic allocation of the port-address mapping through packet flags, our port-address translator supports static allocations for inbound connection requests. It has a limitation that only one inside server can be specified for each IP address, port pair. In other words, only one server can be activated for each well known protocol if there is only one global IP address. The static allocation is assigned when the port-address translator is launched and stay valid all the time. We can have inside servers for well-known ports to service inbound requests.

For example, if we have a HTTP server at (www.our.domain, 80), packets coming to (gateway.our.domain, 80) is translated into (www.our. domain, 80) and relayed to the HTTP server inside. Of course, our server should be registered as gateway.our.domain in the DNS.

Recently, information servers tend to be configured using a server pool to balance the server load. Our translator can be used for dynamic load balancing for a server pool as shown in the previous section.

Compatibility Issues

Since PAT changes the packet headers without the knowledge of senders or receivers of the packet, there are some network applications which do not work properly with the PAT. We have done extensive testing for various applications and found that most of the applications work well with the PAT. The TCP protocols that work with PAT are Telnet, SMTP, NNTP, HTTP, Finger, and X11. And UDP protocols that work with PAT are TFTP, DNS, and RPC. These protocols have common characteristic that they do not include IP address or port number in application layers. Even if they do, it is just for reference.

Some protocols that do not work with the current implementation of PAT are ICMP, NFS, rlogin and FTP. Most of these protocols use the IP address and port number in their application layers and the use of PAT renders the information useless. We have looked at each protocol to examine why they do not work and discuss some possible solutions.

ICMP

In ICMP protocol, the server is supposed to echo the header the client sends. Since PAT changes the address information in the header of the packet, the information inside the echo packet is different from the information that the client sent and the ICMP client does not recognize the echo from the server. One possible solution is to save a copy of the ICMP request header in PAT and switch the information inside the echo packet when the echo comes from the ICMP server.

NFS

NFS uses RPC and UDP protocol. Since both RPC and UDP work well with PAT, NFS should work with PAT. In our current implementation, UDP packets larger than 1472 bytes are not handled correctly and it renders NFS inoperable through PAT when accessing files of bigger than 1472 bytes. However, we believe NFS should work with PAT once the UDP packet handling is correctly implemented.

Rlogin

When we tried rlogin through PAT, we encountered an authentication failure. The rlogin server asks the client the address when the connection is established. Since the address of the client is different from the client's reply, it results in authentication failure. Again, we need to keep track of the initial negotiation and switch the client's address with the address of PAT to pass the authentication.

FTP

FTP is one of the most widely used protocols and it should be supported for the PAT to be of any use. There are two different types of connections used in FTP. The control connection for setting up the session and sending commands and the data connection for data transfer. The control connection is established when the FTP session is first set up while the data connection is separately established and terminated whenever there is a need for data transfer. When FTP client needs to get some data from the server, it sends a PORT command with its IP address and TCP port number for data connection through the control connection and waits listening to the port [RFC959]. Upon receiving the PORT command, the FTP server establishes data connection using the port number received.

When the client has a replicated address which is meaningless outside its regional network, it still sends a PORT command with its IP address and port number. For the client to be able to establish the data connection correctly, the port-address translator should recognize the packet carrying the PORT command and change the IP address and port number with a correct one before relaying this packet to the server.

This can be done by looking at the packets belonging to a FTP session and look for the PORT command with IP address and port number. Since all data are ASCII coded, it is relatively easy to find them. However, there is a slight problem since modifying the address and port number often results in different string length which causes inconsistent TCP sequence number. The inconsistency in TCP sequence number can render further communication impossible. The sequence number as well as the acknowledgement number of the ACK packet should be properly adjusted to ensure correct communication. In our prototype, we solved this problem by keeping track of the difference between the sequence numbers of the original packet and modified packet and properly adjust all the sequence numbers and acknowledge numbers for the forthcoming packets of the same FTP session.

Other Protocols

Applications like SNMP often uses encrypted IP address and port number. It is impossible to handle this kind of application which encrypts the packet contents at this stage. However, we believe that SNMP packets usually are confined inside its regional network and did not consider SNMP.

To be able to implement a transparent port-address translator, it is necessary to classify each applications and devise appropriate translation schemes for each of them. We are currently working on other applications we have overlooked using our prototype.

Performance Considerations

Test Environment

To measure the performance degradation the PAT might have, we have performed several experiments using our PAT prototype. The PAT prototype is implemented so that it will work between two ethernet segments and it is also possible to make it work between Ethernet and serial port so that it can act as a gateway machine.

First, we measured the Ethernet throughput on an isolated network between two Sparc 1 25MHz machines. Next, the PAT is introduced between two machines and the same set of experiments was performed. For the PAT prototype, we used an old IBM compatible PC with 386SX-25 and a 486DX-50 PC. The cost of the 486 PC used in the setup is about $500 and the 386PC can be had less than $100. It's always a good idea to use computers past their prime which have the highest price/performance ratio. The components needed for the PAT is as follows. (motherboard, CPU, RAM (640K if you can find or more), floppy drive, keyboard, power supply, LANcard X 2)

We have used two different workloads to run the experiments. The first is the TCP workload and the other is FTP workload to examine the effect of the FTP-specific service routine. We did not measure the effect of PAT on UDP workload since UDP is simpler than TCP and it would be less affected by PAT.

Workload T

To measure the TCP performance, we programmed a simple client-server application in which the client requests the server to send random amount of data. The connection is reset after the data transfer and established again until total of about 2MB data is transferred. The size of each request is randomly generated using exponential distribution of average 16KB and 160KB respectively. 16K was the average size of WWW objects observed by the proxy server in SNU. 160KB was used to see the effect of large data transfers.

Workload F

It is used to measure the effect of FTP-specific application. We measured the time to transfer files with total size of 7.2MB.

Two experiments for the workloads T and one experiment with workload F have been performed. For each experiment, we measure the throughput and response time of the transfer. We first measured the time without the PAT for the basis of comparison and measured the time with 486 PAT and 386 PAT respectively. To run the experiments, two sparc 1 workstations were used on an isolated network and the PAT was set up between them.

Test Results

Figures 7, 8, and 9 show the results of the experiments. When transferring large files using TCP, the

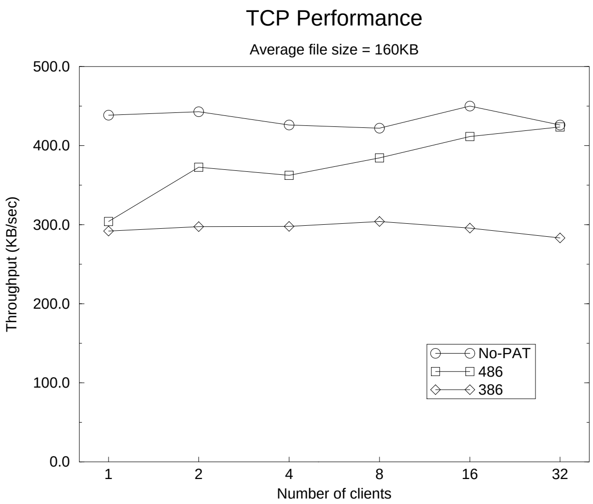

Figure 7: Performance of PAT (TCP, large file)

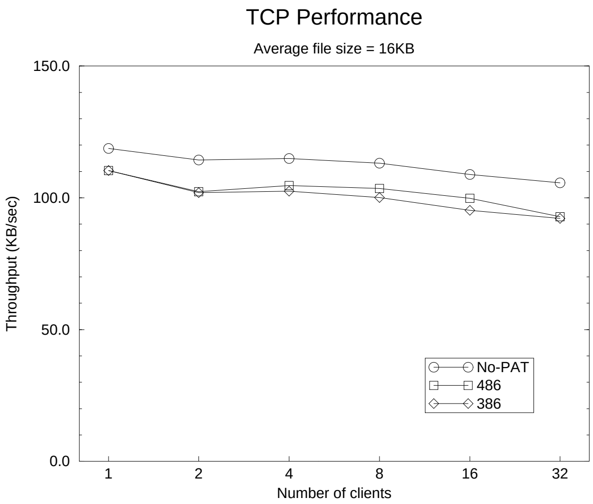

Figure 8: Performance of PAT (TCP, small file)

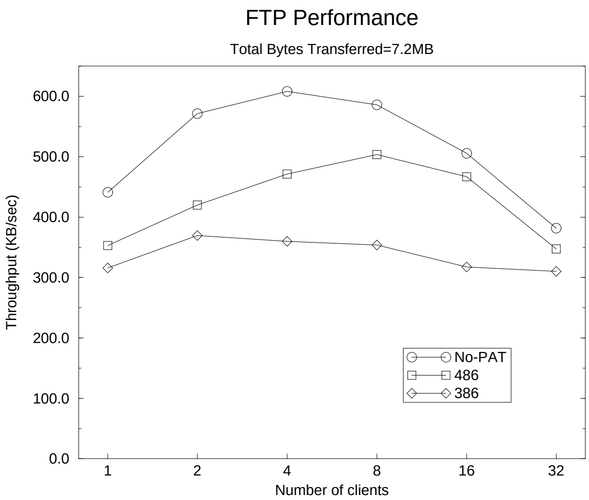

Figure 9: Performance of PAT supporting FTP protocol

386 PAT clearly shows poor performance since the slow gateway imposes great network delay. However, when using 486 PAT, the difference is relatively minor as the number of active connections increases. For small file transfers, there is less than 20% drop in throughput both for 386 and 486 PAT. We believe this is mainly due to the slow machines we used for the experiments. The machines rather than the network were the bottleneck in this experiment. As for the overhead dealing with FTP specific processing, we can see the same trend as with TCP large file transfer and we conclude that FTP processing does not impose big overhead in PAT operation.

We have also measured the round trip time of a small packet with and without PAT. Table 2 shows the result of that test.

In this experiment, we can see that 486 PAT only adds about 5% and 386 PAT has 10% more delay. Considering that it is round trip, for a packet to pass through PAT takes 0.00025 sec for 486 and 0.0005 sec for 386 PAT.

	RTT between Sparc1
no PAT	0.009287 sec
486 PAT	0.009784 sec
386 PAT	0.011348 sec

Table 2: Round trip time of a small packet (4 byte)

Overall, the 486 PAT only has less than 20% overhead with normal network traffic on Ethernet and we believe it would be more than enough to be used as a gateway for small sites connected through T-1 or slower links.

Conclusion

The port-address translator we propose can reuse IP addresses as well as prove load balancing without modifying existing applications or network structure. It is also very inexpensive. Compared to the address translator using DNS which provides one to one mapping between globally unique address and reused address, our scheme can provide a large number of connections using just one globally unique IP address. Ultimately, it can be used to transform a class C network to a network with only one globally unique IP address and reused addresses. We believe that the IP address reuse through port-address translator is a very practical solution and it can be a short term solution to the IP address shortage. The advantages of the port-address translator are summarized as follows:

- The cost of implementing a port-address translator and following network restructuring is comparable to that of packet filtering gateway.
- The port-address translator provide users transparent service without modifying existing applications programs or network topology.
- It is possible to relay UDP packets by careful considerations.
- It is possible to relay inbound requests by reserving ports or using server pool.
- Most of the applications such as FTP can be relayed by application specific gateway.
- It can provide security as a firewall does since it controls all outbound connections.

However, the port-address translator has the following drawbacks. These problems should be addressed before it can be a viable short-term solution for IP address depletion.

- There are some applications which need special consideration and some applications which can not be handled.}
- Various protocols such as ICMP, SNMP, RIP should be considered.}
- It has the single point of failure.

Availability

The prototype implementation of PAT is available from ftp://arari.snu.ac.kr/pub/pat . This version still does not handle long UDP packets correctly.

Acknowledgement

We thank Yongjae Lee for his sleepless nights implementing and debugging the prototype.

Author Information

Heon Y. Yeom received his B.S. degree in computer science from Seoul National University in Seoul, Korea and received his M.S and Ph.D. degree in computer science from Texas A&M University. While at TAMU, he also worked as a systems analyst in Texas Transportation Institute managing all kinds of computers. He currently is an assistant professor at Seoul National University and administers the computer center operations. Reach him electronically at yeom@arirang.snu.ac.kr .

Jungsoo Ha received his B.S. degree in computer science from Seoul National University and is presently working for his M.S. degree. He is one of the student systems administrators in SNU computer center and is responsible for the operation of SUN-SITE Korea and SNU proxy server. Reach him electronically at huggies@arirang.snu.ac.kr .

Ilhwan Kim received his B.S. and M.S degree in computer science from Seoul National University and is presently working for his Ph.D degree. He was one of the best student systems administrator in SNU and is now retired after he realized that it would be hard to earn his doctorate doing systems administration. Still, he intervenes from time to time as he sees fit. Reach him electronically at ilhwan@arirang.snu.ac.kr .

Bibliography

[CB94] W. R. Ceswick and S. M. Bellovin. *Firewalls and Internet Security*, section 3, pages 49-83. Addison-Wesley, 1994.

[KBM94] E. D. Katz, M. Butler and R. McGrath, ‘‘A Scalable HTTP Server: The NCSA Prototype’’, *The 1st International WWW Conference*, May 1994, Geneva, Switzerland

[KK92] D.Koblas and M. R. Koblas. In *USENIX Security Symposium Proceedings III*, September 1992.

[TE93] Paul F. Tsuchiya and Tony Eng. ‘‘Extending the IP Internet Through Address Reuse’’, *ACM Computer Communication Review*, January 1993.

[Stev95] W. R. Stevens, *TCP/IP Illustrated*, Vol. 1, pp 148-151, Oct. 1995.

[RFC792] J. Postel, *Internet Control Message Protocol*, 09/01/1981.

[RFC793] J. Postel, *Transmission Control Protocol*, 09/01/1981.

[RFC821] J. Postel, *Simple Mail Transfer Protocol*, 08/01/1982.

[RFC854] J. Postel, J. Reynolds, *Telnet Protocol Specification*, 05/01/1983.

[RFC959] J. Postel, J. Reynolds, *File Transfer Protocol*, 10/01/1985.

[RFC977] B. Kantor, P. Lapsley, *Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News*, 02/01/1986.

[RFC1034] P. Mockapetris, *Domain Names – Concepts and Facilities*, 11/01/1987.

[RFC1035] P. Mockapetris, *Domain Names – Implementation and Specification*, 11/01/1987.

[RFC1050] Sun Microsystems, Inc, *RPC: Remote Procedure Call Protocol Specification*, 04/01/1988.

[RFC1258] B. Kantor, *BSD Rlogin*, 09/11/1991.

[RFC1288] D. Zimmerman, *The Finger User Information Protocol*, 12/19/1991.

[RFC1350] K. Sollins, *THE TFTP PROTOCOL (REVISION 2)*, 07/10/1992.

[RFC1519] V. Fuller, T. Li, J. Yu, and K. Varadhan, *Classless Inter-domain Routing (CIDR): An Address Assignment and Aggregation Strategy*, September 1993.

[RFC1579] S. Bellovin, *Firewall-Friendly FTP*, 02/18/1994.

[RFC1597] Y. Rekhter and B. Moskowitz, *Address Allocation for Private Internets*, March 1994.

[RFC1631] K. Egevang and P. Francis, *The IP Network Address Translator (NAT)*, May 1994.

[RFC1813] B. Callaghan, B. Pawlowski, P. Staubach, *NFS Version 3 Protocol Specification*, 06/21/1995.

[RFC1945] T. Berners-Lee, R. Fielding, H. Nielsen, *Hypertext Transfer Protocol – HTTP/1.0*, 05/17/1996.