ISSN No. 0976-5697

Volume 1, No. 4, Nov-Dec 2010

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

IP MASQUERADING: A Network Address Translation Technique

Prashant P.J.
M.Tech. (CSE)
M.S.Ramaiah Institute of Technology
Bangalore, India
pras1011@yahoo.co.in

Chiranji Lal Chowdhary*
School of Information Technology and Engineering
VIT Universit
Vellore, India
chiranji.lal@vit.ac.in

Abstract: IP Masquerade, called "IPMASQ" or "MASQ" for short, is a form of Network Address Translation (NAT) which allows internally connected computers that do not have one or more registered Internet IP addresses to communicate to the Internet via the server's Internet IP address. Since IPMASQ is a generic technology, you can connect the server's internal and external to other computers through LAN technologies like Ethernet, Token Ring, and FDDI, as well as dialup connections line PPP or SLIP links. In this paper/work we primarily used Ethernet and PPP connections because it is most commonly used with DSL or Cable modems and dialup connections.

Keywords: Network Address Translation (NAT), Network Address Port Translation (NAPT), Simple Mail Transfer Protocol (SMTP), Internet Service Provider (ISP), Point-to-Point Protocol (PPP), Digital Subscriber Line (DSL), Serial Line Internet Protocol (SLIP), Fiber Distributed Data Interface (FDDI).

I. INTRODUCTION

Network Address Translation (NAT) was originally developed as an interim solution to combat IPv4 address depletion by allowing globally registered IP addresses to be re user shared by several hosts. Although it can be used to translate between any two address realms, NAT is most often used to map IPs from the non routable private address spaces shown below.

Table: I

Class	Private Address Range
A	10.0.0.0 … 10.255.255.255
B	172.16.0.0 … 172.16.255.255
C	192.168.0.0 … 192.168.255.255

These addresses were allocated for use by private networks that either do not require external access or require limited access to outside services. Enterprises can freely use these addresses to avoid obtaining registered public addresses. But, because private addresses can be used by many, individually within their own realm, they are non routable over a common infrastructure. When communication between a privately addressed host and a public network is needed, address translation is required. This is where NAT comes in.

NAT routers sit on the border between private and public networks, converting private addresses in each IP packet into legally registered public ones. They also provide transparent packet forwarding between addressing realms. The packet sender and receiver should remain unaware that NAT is taking place. Today, NAT is commonly supported by WAN access routers and firewalls—devices situated at the network edge.

NAT works by creating bindings between addresses. In the simplest case, a one-to-one mapping may be defined between public and private addresses. Known as static NAT, this can be accomplished by a straightforward, stateless implementation that transforms only the network part of the address, leaving the host part intact. The payload of the packet must also be considered during the translation process. The IP checksum must, of course, be recalculated. Because TCP checksums are computed from a pseudo-header containing source and destination IP address (prepended to the TCP payload), NAT must also regenerate the TCP checksum.

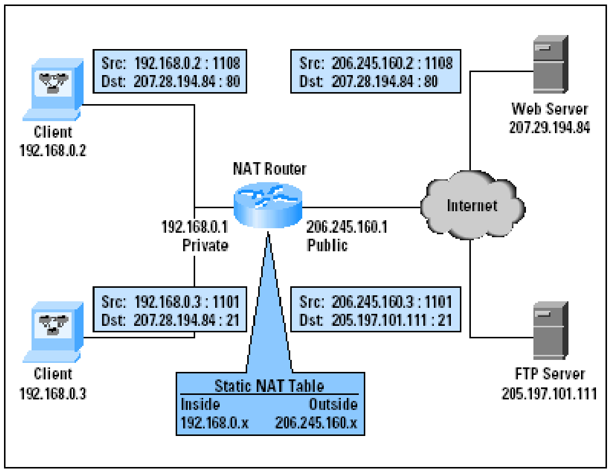

Figure 1: Static NAT

More often, a pool of public IP addresses is shared by an entire private IP subnet or dynamic NAT. Edge devices that run dynamic NAT create bindings "on the fly," building a NAT Table. Connections initiated by private hosts are assigned a public address from a pool. As long as the private host has an outgoing connection, it can be reached by incoming packets sent to this public address. After the connection is terminated or a timeout is reached, the binding expires, and the address is returned to the pool for reuse. Dynamic NAT is more complex because state must be maintained, and connections must be rejected when the pool is exhausted. But, unlike static NAT, dynamic NAT enables address reuse, reducing the demand for legally registered public addresses.

A variation of dynamic NAT known as Network Address Port Translation (NAPT) may be used to allow many hosts to share a single IP address by multiplexing streams differentiated by TCP/UDP port number. For example, suppose private hosts 192.168.0.2 and 192.168.0.3 both send packets from source port 1108. A NAPT router might translate these to a single public IP address 206.245.160.1 and two different source ports, say 61001 and 61002. Response traffic received for port 61001 is routed back to 192.168.0.2:1108, while port 61002 traffic is routed back to 192.168.0.3:1108.

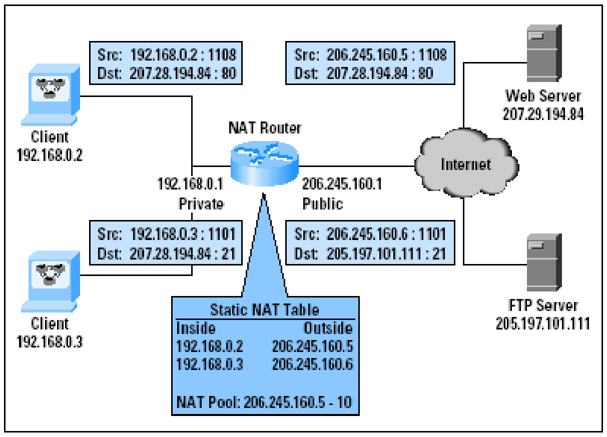

Figure 2: Dynamic NAT

NAPT or IP masquerading is commonly implemented on small Office/ Home Office routers to enable shared Internet access for an entire LAN through a single public address. Because NAPT maps individual ports, it is not possible to "reverse map" incoming connections for other ports unless another table is configured. A virtual server table can make a server on a privately addressed DMZ reachable from the Internet via the public address of the NAPT router (one server per port). This is really a limited form of static NAT, applied to incoming requests.

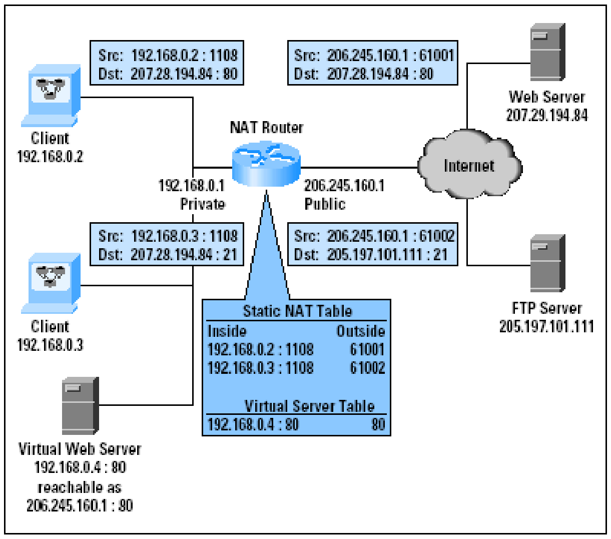

Figure 3: NAPT

In some cases, static NAT, dynamic NAT, NAPT, and even bi directional NAT or NAPT may be used together. For example, an enterprise may locate public Web servers outside of the firewall, on a DMZ, while placing a mail server and clients on the private inside network, behind a NAT-ing firewall. Furthermore, suppose there are applications within the private network that periodically connect to the Internet for long periods of time.

In this case:

[a] Web servers can be reached from the Internet without NAT, because they live in public address space.

[b] Simple Mail Transfer Protocol (SMTP) sent to the private mail server from the Internet requires incoming translation. Because this server must be continuously accessible through a public address associated with its Domain Name System (DNS) entry, the mail server requires static mapping.

[c] For most clients, public address sharing is usually practical through dynamically acquired addresses (either dynamic NAT with a correctly sized address pool, or NAPT).

[d] Applications that hold onto dynamically acquired addresses for long periods could exhaust a dynamic NAT address pool and block access by other clients. To prevent this, long-running applications may use NAPT because it enables higher concurrency (thousands of port mappings per IP address).

A. *Masquerading*

To complete this dynamic NAT section one example for masquerading as it is widely used these days. We have a small office- or home-network consisting of two hosts and a Linux-server. The Linux server possibly provides print- and file services, and it also serves as a NAT-router to the Internet for the other hosts. All internal IPs are translated using the official IP given by the ISP, this IP is the Linux router's IP on the network interface to the ISP.

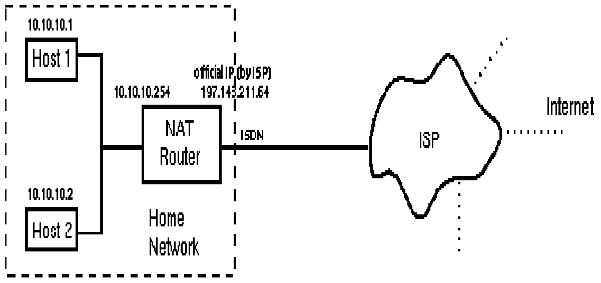

Figure: 4

Linux masquerading is extremely popular and many application specific modules have been written, that (among other things) take care of translating IPs transmitted in the data part of IP packets.

II. IP MASQUERADE NAT FUNCTIONS

Masquerade NAT is used to allow your private network to hide behind, as well as be represented by, the address bound to the public interface. In many situations, this is the address that has been assigned by an Internet Service Provider (ISP), and the address may be dynamic in the case of a Point-to-Point Protocol (PPP) connection. This type of translation can only be used for connections originating within the private network destined for the outside public network. Each outbound connection is maintained by using a different source IP port number.

Masquerade NAT allows workstations with private IP addresses to communicate with hosts on the Internet using iSeries server. iSeries server has an IP address assigned by the local ISP as its Internet gateway. The term locally attached machine is used to refer to all machines on an internal network regardless of the method of attachment (LAN or WAN) and regardless of the distance of the connection.

The term external machines is used to refer to machines located on the In To the Internet, all of your workstations appear to be contained within your iSeries server; that is, only one IP address is associated with both your iSeries server and your workstations. When a router receives a packet intended for your workstation, it attempts to determine what address on the internal LAN should receive the packet and sends it there. Each workstation must be set up so that iSeries server is its gateway and also its default destination. The correspondence between a particular communication connection (port) and a workstation is set up when one of your workstations sends a packet to iSeries server to be sent to the Internet. The masquerade NAT function saves the port number so that when it receives responses to your workstation's packet over that connection, it can send the response to the correct workstation.

The following figure illustrates how Masquerade NAT works.

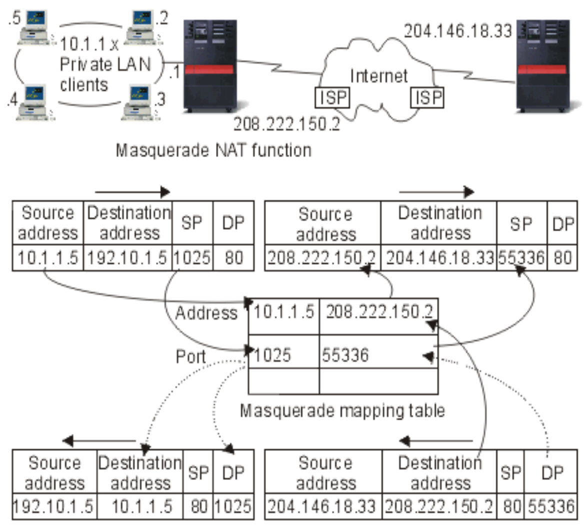

Figure: 5

A record of active port connections and the last access time by either end of the connection is created and maintained by masquerade NAT. These records are periodically purged of all connections that are idle for a predetermined amount of time based on the assumption that an idle link is no longer in use.

All communication between your workstation and the Internet must be initiated by locally attached machines. This is an effective security firewall; the Internet knows nothing of the existence of your workstations, and it cannot broadcast those addresses to the Internet. A key to masquerade NAT implementation is the use of logical ports, issued by masquerade NAT to distinguish between the various communication streams. TCP contains a source and a destination port number. To these designations, NAT adds a logical port number.

A. *Outbound masquerade NAT processing*

The outbound message in the figure above is a packet from the private LAN to the Internet. An outbound message (local to external) contains the source port used by the originating workstation. NAT saves this number and replaces it in the transport header with a unique logical port number. For outbound datagrams, the source port number is the local port number.

[a] Outbound masquerade NAT processing assumes that all IP packets it receives are bound for external IP addresses, and therefore does not check to determine whether a packet should be routed locally.

[b] The set of logical port numbers searches for a match on the transport layer as well as a source IP address and source port. If found, the corresponding logical port number is substituted for the source port. If no matching port number is found, a new one is created, and a new logical port number is selected and substituted for the source port.

[c] The source IP address is translated.

[d] The packet is then processed as usual by IP and is sent to the correct external system.

B. *Inbound masquerade NAT processing (response & other)*

The inbound message in the figure above is a packet from the Internet to your private LAN. For inbound data grams, the destination port number is the local port number. (For inbound messages, the source port number is the external port number. For outbound messages, the destination port number is the external port number.) Response messages returning from the Internet bound for a locally attached machine have a masquerade-assigned logical port number as the destination port number in the transport layer header. The masquerade NAT inbound processing steps are:

[a] Masquerade NAT searches its database for this logical port number (source port). If it is not found, the packet is assumed to be an unsolicited packet, and the packet is returned to the caller unchanged. It is then handled as a normal unknown destination.

[b] If a matching logical port number is found, a further check is made to determine that the source IP address matches the destination IP address of the existing logical port number table entry. If it matches, the original local machine's port number replaces the source port in the IP header. If the check fails, the packet is returned unchanged.

[c] The local matching IP addresses are placed in the packet IP destination.

[d] The packet is then processed, as usual by IP or TCP, and ends up at the correct locally attached machine. Because masquerade NAT requires a logical port number to determine the correct source and destination port addresses, masquerade NAT is incapable of handling unsolicited datagrams from the Internet.

Another IP Masquerading Example

PPP/ETH/etc.

```
+------------+ +-------------+  to ISP provider
| Linux #1 | PPP/ETH/etc. | Anybox |
| | | |
<---------- modem1| |modem2 ----------- modem3| |
| | | |
111.222.121.212 | | 192.168.0.100 | |
+------------+ +-------------+
```

In the above drawing, a Linux box with IP_MASQUERADING is installed as Linux #1 and is connected to the Internet via PPP, Ethernet, etc. It has an assigned public IP address of 111.222.121.212. It also has another network interface (e.g. modem2) connected to allow incoming network traffic be it from a PPP connection, Ethernet connection, etc. The second system which does not need to be Linux connects into the Linux #1 box and starts its network traffic to the Internet. This second machine does NOT have a publicly assigned IP address from the Internet, say 192.168.0.100. With IP Masquerade and the routing configured properly, this second machine "Any box" can interact with the Internet as if it was directly connected to the Internet with a few small exceptions.

"Do not forget to mention that the "ANYBOX" machine should have the Linux #1 box configured as its default gateway (whether it be the default route or just a subnet is no matter). If the "ANYBOX" machine is connected via a PPP or SLIP connection, the Linux #1 machine should be configured to support proxy arp for all routed addresses.

I tell machine ANYBOX that my PPP or Ethernet connected Linux box is its gateway.

When a packet comes into the Linux box from ANYBOX, it will assign the packet to a new TCP/IP source port number and insert its own IP address inside the packet header, saving the originals. The MASQ server will then send the modified packet over the PPP/ETH interface onto the Internet.

When a packet returns from the Internet into the Linux box, Linux examines if the port number is one of those ports that was assigned above. If so, the MASQ server will then take the original port and IP address, put them back in the returned packet header, and send the packet to ANYBOX.

The host that sent the packet will never know the difference.

IP Masquerade, called "IPMASQ" or "MASQ" for short, is a form of Network Address Translation (NAT) which allows internally connected computers that do not have one or more registered Internet IP addresses to communicate to the Internet via the server's Internet IP address. Since IPMASQ is a generic technology, you can connect the server's internal and external to other computers through LAN technologies like Ethernet, Token Ring, and FDDI, as well as dialup connections line PPP or SLIP links. This project primarily uses Ethernet and PPP connections in examples because it is most commonly used with DSL or Cable modems and dialup connections.

IP Masquerade is a networking function in Linux similar to the one-to-many (1: Many) NAT or Network Address Translation servers found in many commercial firewalls and network routers. For example, if the host is connected to the Internet via PPP, Ethernet, etc., the IP Masquerade feature allows other "internal" computers connected to this box (via PPP, Ethernet, etc.) to also reach the Internet as well. IP Masquerading allows for this functionality even though these internal machines don't have an officially assigned IP address. MASQ allows a set of machines to invisibly access the Internet via the MASQ gateway. To other machines on the Internet, the outgoing traffic will appear to be from the IP MASQ server itself. In addition to the added functionality, IP Masquerade provides the foundation to create a HEAVILY secured networking environment. With a well built firewall.

C. Current Status

IP Masquerade has been in the Linux kernels for several years now and is quite mature as the kernel enters the 2.4.x stage. Kernels since Linux 1.3.x have had MASQ support built-in. Today, many individuals and commercial businesses are using it with excellent results.

2.4.x kernel users: The 2.4.x kernel hosts which is both far superior, faster, and more secure than any previous versions written for Linux. Unfortunately, several kernel modules that were written for the 2.2.x kernel to support things like UDP-based RealAudio, etc. have not been ported to 2.4.x yet. Because of this, some people should consider NOT upgrading if these network applications are critical to them. But, at the same time, some of these programs have been updated and now use different, NAT-friendly protocols. Thus special NAT treatment is no longer required. Common network functionalities like Web browsing, telnet, ssh, ping, traceroute, etc. work well over stock IP Masquerade setups.

Other network applications such as **ftp, irc**, and Real Audio work well with the appropriate additional IP MASQ modules loaded into the kernel as modules. Other network-specific programs like streaming audio (MP3s, True Speech, etc) should work too without any special module. Some users on the mailing list also had good results with video conferencing software.

It should be noted that running IP Masquerade with only ONE network card to MASQ between internal and external Ethernet networks is NOT recommended. IP Masquerade works well as a server to other 'client machines' running various operating systems and hardware platforms.

D. Who Can Benefit From IP Masquerade?

If you have a Linux host connected to the Internet and if you have internal computers running TCP/IP connected that are connected to this Linux box via on a network, and if your Linux host has more than one modem and acts as a PPP or SLIP server connected to other computers, and these machines do not have official or public assigned IP addresses (i.e. addressed with private TCP/IP numbers). If you want those other machines to communicate to the Internet without spending extra money to acquire additional Public or Official TCP/IP addresses from your ISP, then you should either configure Linux to be a router or purchase an external router.

E. Who Doesn't Need IP Masquerade?

If your machine is a stand-alone Linux host connected to the Internet (setting up a firewall is a good idea though), or if you already have multiple assigned public addresses for your other machines, and if you don't like the idea of a 'free ride' using Linux and feel more comfortable using expensive commercial tools to perform the exact same functionalities.

III. SYSTEM SPECIFICATION

- What are the minimum hardware requirements?

A 486/66 box with 16MB of RAM was more than sufficient to fill a 1.54Mb/s MASQ has also been known to run quite well on 386SX-16s with 8MB of RAM. Yet, it should be noted that IP Masquerade starts thrashing the system with more than 500 MASQ entries.

The only application that I know which can temporarily break IP Masquerade, is Game Spy. Why? When it refreshes its lists, it creates 10,000s of quick connections in a VERY short period of time. Until these sessions timeout, the MASQ tables become "FULL". There is a hard limit of 4096 concurrent connections each for TCP & UDP. If you want to change the limit - you need to change the PORT_MASQ_BEGIN & PORT_MASQ_END values to get an appropriately sized range above 32K and below 64K.

Software Requirements

The newest Linux kernel 2.6 are now using both a completely new TCP/IP network stack as well as a new NAT sub-system called Net Filter. Within this Net Filter suite of tools, we now have a tool called IPTABLES for the Linux kernel 2.6 much like there was IPCHAINS for the Linux and IPFWADM for the Linux kernels. The new IPTABLES system is far more powerful (combines several functions into one place like true NAT functionality), offers better security (state ful inspection), and better performance with the new Linux TCP/IP stack. But this new suite of tools can be a bit complicated in comparison to older generation Linux kernels 2.6.

Unlike the migration to IPCHAINS from IPFWADM, the new Net Filter tool has Linux kernel modules that can actually support, It basically means that if you want to use IPMASQ or PORTFW functionality under a kernel 2.6, you shouldn't use IPCHAINS rules but IPTABLES ones instead. Please also keep in mind that there might be several benefits in performing a full rule set re-write to take advantage of the newer IPTABLES features like state ful tracking, etc. but that is dependant upon how much time you have to migrate your old rule sets.

Some new Linux kernel 2.6 functionalities include the following:

PROs:

Lots of new protocols modules like: amanda, eggdrop, ipsec, ipv6, portscan, pptp, quota, rsh, talk, and tftp.

TRUE 1:1 NAT functionality for those who have TCP/IP addresses and subnets to use (no more iproute2 commands).

Stateful application level (FTP, IRC, etc.) and stateful protocol level (TCP/UDP/ICMP) network traffic inspection.

This supports for both external and internal traffic. This means that users that have PORTFW for external traffic and REDIR for internal port redirection do not need to use two tools any more.

PORT Forwarding of FTP traffic to internal hosts is now completely supported and is handled in the conn_trak_ftp module.

Full Policy-Based routing features (source-based TCP/IP address routing).

Compatibility with Linux's Fast Route feature for significantly faster packet forwarding (a.k.a Linux network switching).

Fully supports TCP/IP v4, v6, and even DECnet (ack!)

Supports wildcard interface names like "ppp*" for serial interfaces like ppp0, ppp1, etc

Supports filtering on both input and output INTERFACES (not just IP addresses)

Source Ethernet MAC filtering

Other features like traffic mirroring, securing traffic per login, etc.

Net filter is an entirely new architecture thus most of the older 2.2 kernel modules written to make non-NAT friendly network applications work through IPMASQ need to be re-written for the 2.6 kernels. Because of this, if you specifically need functionality from some of these modules , you should stay with a 2.2 kernel until these modules have been either ported or the application has been updated to use NAT-friendly protocol.

IV. ADDRESS TRANSLATION TECHNIQUES

Speaking about NAT we must know that address translation can be done statically or dynamically. In the first case the assignment of NAT-IPs to original IPs is unambiguous, in the latter case it is not. In static NAT a certain fixed original IP is always translated to the same NAT IP at all times, and no other IP gets translated to the same NAT-IP, while in dynamic NAT the NAT IP depends on various runtime conditions and may be a completely different one for each single connection.

In the following sections m,n are defined as follows:

m: number of IPs that need to be translated (original IPs)

n: number of IPs available for translation (NAT IPs)

A. Static Network Address Translation

m:n-Translation, m,n>=1 and m=n (m,n in N)

With static address translation we can translate between IP networks that have the same size (contain the same number of IPs). A special case is when both networks contain just one IP, i.e. the netmask is 255.255.255.255. This NAT strategy is easy to implement, since the entire translation process can be written as one line containing a few simple logic transformations:

new-address = new-network OR (old-address AND (NOT netmask))

In addition, no information about the state of connections that are being translated needs to be kept, looking at each IP packet individually is sufficient. Connections from outside the network to inside hosts are no problem, they just appear to have a different IP than on the inside, so static NAT is (almost) completely transparent.

Example:

NAT rule: translate all IPs in network 138.201.148 to IPs in network 94.64.15, netmask is 255.255.255.0 for both

now 138.201.148.27 is translated to 94.64.15.27, and so on

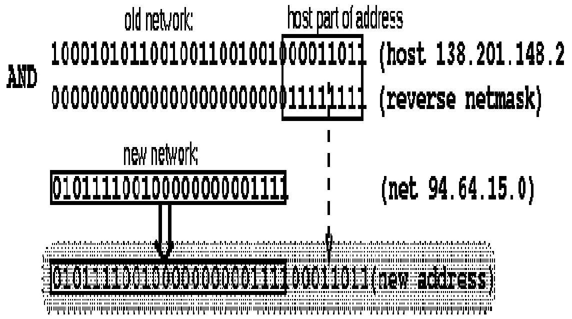
old network:
host part of address
AND
10001010110010011001001000011011 (host 138.201.148.2
00000000000000000000000011111111 (reverse netmask)
new network:
010111100100000000001111 (net 94.64.15.0)
01011110010000000000111100011011(new address)

Figure : 6

B. Dynamic Address Translation

m:n-Translation, m>=1 and m>=n (m,n in N)

Dynamic address translation is necessary when the number of IPs to translate does not equal the number of IPs to translate to, or they are equal but for some reason it is not desirable to have a static mapping. The number of hosts communicating is generally limited by the number of NAT IPs available. When all NAT IPs are being used then no other connections can be translated and must therefore be rejected by the NAT router, for example by sending back 'host unreachable'. Dynamic NAT is more complex than static NAT, since we must keep track of communicating hosts and possibly even of connections which requires looking at TCP information in packets.

As mentioned above, dynamic NAT may also be useful when there are enough NAT IPs, i.e. when m=n. Some people use this as a security measure: it is impossible for someone outside a network to get useful IP numbers to connect to of hosts behind a NAT router doing dynamic address translation by looking at connections that take place, since next time the same host may connect using a completely different IP. In this special case even having more NAT IPs than IPs to be translated (m<n) may make some sense.

Connections from outside are only possible when the host that shall be reached still has a NAT-IP assigned, i.e. if it still has an entry in the dynamic NAT table, where the NAT router keeps track of which internal IP is mapped to which NAT IP. For instance, non-passive FTP sessions, where the server attempts to establish the data-channel, are no problem, since when the server sends its packets to the FTP-client there is already an entry for the client in the NAT-table, and it is extremely likely it still contains the same client-IP to NAT-IP mapping that were there when the client started the FTP-control channel, unless the FTP session has been idle for longer than the timeout of the entry. However, if an outsider wants to establish a connection to a certain host on the inside at an arbitrary time there are two possibilities: the inside host does not have an entry in the NAT-table and is therefore unreachable, or it has an entry, but which NAT-IP must be used is unknown, except, of course, the IP to connect to is known because the internal host is communicating with the outside. In the latter case, however, only the NAT-IP is known but not the internal IP of the host, and this knowledge is valid only while the communication of the internal host takes place plus the timeout of the entry in the NAT routers table.

Example:

NAT rule: dynamically translate all IPs in (class B) network 138.201 to IPs in (class C) network 178.201.112

each new connection from the inside gets assigned an IP from the pool of class C addresses, as long as there are unused addresses left

if a mapping already exists for the internal host this one is used instead

as long as the mapping exists the internal host can be reached via the IP that has been (temporarily) assigned to it

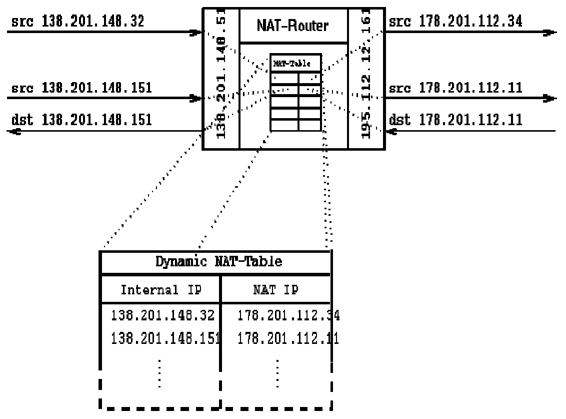

Figure: 7

C. *Masquerading (NAPT)*

m:n-Translation, m>=1 and n=1 (m,n in N)

A very special case of dynamic NAT is m:1-translation, a.k.a. masquerading which became famous under that name because Linux can do it. It is probably the kind of NAT-technique that is used most often these days. Here many IP numbers are hidden behind a single one. In contrast to the original dynamic NAT this does not mean there can be only one connection at a time. In masquerading an almost arbitrary number of connections is multiplexed using TCP port information. The number of simultaneous connections is limited only by the number of TCP-ports available.

A special problem of masquerading is that some services on certain hosts only accept connections coming from privileged ports in order to ensure that it does not come from an ordinary user. The assumption that only the super user can access those ports is not valid, since on DOS or Windows machines everybody can use them, nether the less, some programs rely on this and cannot be used over a masqueraded connection. The Linux implementation uses no privileged ports for masquerading to avoid interfering with 'regular' connections to these ports. Masquerading usually uses ports in the upper range, in Linux this range starts at port 2000 and ends at 2000+4096, This also shows that the Linux implementation by default only allows 4096 concurrent connections. To allow masqueraded connections on ports outside of such a port range requires keeping and managing even more information about the state of connections. Linux, for example, simply treats all packets with *destination IP = local IP* and *destination port is inside the range used for masquerading* , as packets that have to be demasqueraded, i.e. they are answers to packets that have been masqueraded on their way out.

Incoming connections are impossible with masquerading, since even when a host has an entry in the masquerading table of the NAT device this entry is only valid for the connection being active. Even ICMP-replies that belong to connections *(host/port unreachable)* do not get through to the sender automatically but must be filtered and relayed by the NAT-routers software. While it is true that incoming connections are impossible we can take additional measures to enable them, but they are not part of the masquerading code. We could, for an example, set up the NAT-device so that it relays all connections coming in from the outside to the telnet-port to a host on the inside. However, since we have just one IP that is visible outside for enabling incoming connections for the same service but for different hosts on the inside we must listen on different ports on the NAT-device, one for each service and internal IP. Since most applications listen on well-known ports that cannot be easily (and transparently!) changed, this is quite inconvenient and often no option, especially not for public services. The only solution is to have as many (external) IPs as the number of services that shall be provided. An external IP can still be shared by different services, and then be remapped to different internal IPs using NAT, but that is not part of masquerading.

Example:

NAT rule: masquerade the internal network 138.201 using the NAT routers own address

for each outgoing packet the source IP is replaced by the routers (external) IP, and the source port is exchanged against an unused port from the range reserved exclusively for masquerading on the router

if the destination IP of an incoming packet is the local router IP and the destination port is inside the range of ports used for masquerading on the router, the NAT router checks its masquerading table if the packet belongs to a masqueraded session; if this is the case, the destination IP and port of the internal host is inserted and the packet is sent to the internal host.

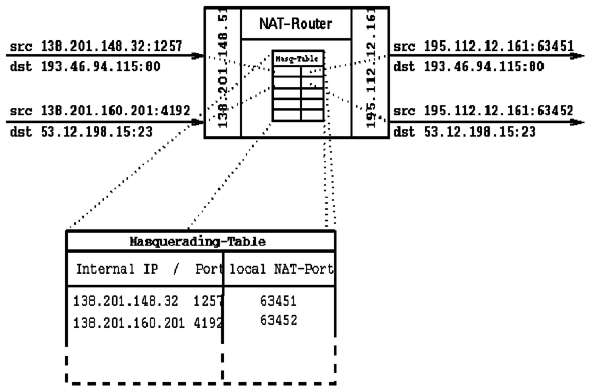

Figure: 8

The greatest advantage of masquerading for many people is that they only need one official IP-address but the entire internal network can still directly access the Internet. This is so important because IP addresses have become quite expensive. As long as there are application level gateways we do not need any IPs or any kind of NAT and one IP is still enough, but for some protocols, e.g. all UDP based services, there is just no gateway so direct IP connectivity is necessary.

At the time of this writing there existed an Internet Draft (which I should not reference here, since it is just a draft) from the same people who wrote RFC 1631 (NAT). It explains masquerading, that they call Network Address Port Translation (NAPT), in great depth. There is no IETF-paper (none that I could find, at least) on more recent forms of NAT like the ones introduced in the following chapters, although there are (commercial) implementations of them. It seems like for the IETF NAT only exists for helping to solve the classical Internet address space shortage problems described above.

D. How does IP Masquerade differ from Proxy and NAT

PROXY:

Proxy servers are available for: Win95, NT, Linux, Solaris, etc.

All applications behind the proxy server must both SUPPORT proxy services (SOCKS) and be CONFIGURED to use the Proxy server - Screws up WWW counters and WWW statistics A proxy server uses only 1 public IP address, like IP MASQ, and acts as a translator to clients on the private LAN (WWW browser, etc.).

This proxy server receives requests like TELNET, FTP, WWW, etc. from the private network on one interface. It would then in turn, initiate these requests as if someone on the local box was making the requests. Once the remote Internet server sends back the requested information, it would re-translate the TCP/IP addresses back to the internal MASQ client and send traffic to the internal requesting host. This is why it is called a PROXY server.

Note: ANY applications that you might want to use on the internal machines *MUST* have proxy server support like Netscape and some of the better TELNET and FTP clients. Any clients that don't support proxy servers won't work. Another nice thing about proxy servers is that some of them can also do caching (Squid for WWW). So, imagine that you have 50 proxied hosts all loading Netscape at once. If they were installed with the default homepage URL, you would have 50 copies of the same Netscape WWW page coming over the WAN link for each respective computer. With a caching proxy server, only one copy would be downloaded by the proxy server and then the proxied machines would get the WWW page from the cache. Not only does this save bandwidth on the Internet connection, it will be MUCH MUCH faster for the internal proxied machines.

E. IP MASQUERADING :

IP Masq is available on Linux and a few ISDN routers such or as the Zytel Prestige128, Cisco 770, NetGear ISDN routers , etc.

Functions :

Only 1 IP address needed.

Doesn't require special application support.

Uses firewall software so your network can become more secure.

Incoming traffic cannot access your internal LAN unless the internal LAN initiates the traffic or specific port forwarding software is installed. Many NAT servers CANNOT provide this functionality.

Special protocols need to be uniquely handled by firewall redirectors, etc. Linux has full support for this (FTP, IRC, etc.) capability but many routers do NOT (Net Gear DOES). Masq or 1: Many NAT is similar to a proxy server in the sense that the server will perform IP address translation and fake out the remote server (WWW for example) as if the MASQ server made the request instead of an internal machine.

The major difference between a MASQ and PROXY server is that MASQ servers don't need any configuration changes to all the client machines. Just configure them to use the linux box as their default gateway and everything will work fine. You WILL need to install special Linux modules for things like RealAudio, FTP, etc. to work) Also, many users operate IP MASQ for TELNET, FTP, etc. and also setup a caching proxy on the same Linux box for WWW traffic for the additional performance.

F. NAT:

NAT servers are available on Windows 95/NT, Linux, Solaris, and some of the better ISDN routers.

Very configurable and No special application software needed ,Requires a subnet from your ISP Network Address Translation is the name for a box that would have a pool of valid IP addresses on the Internet interface which it can use. Whenever the Internal network wanted to go to the Internet, it associates an available VALID IP address from the Internet interface to the original requesting PRIVATE IP address. After that, all traffic is re-written from the NAT public IP address to the NAT private address. Once the associated PUBLIC NAT address becomes idle for some pre-determined amount of time, the PUBLIC IP address is returned back into the public NAT pool. The major problem with NAT is, once all of the free public IP addresses are used, any additional private users requesting Internet service are out of luck until a public NAT address becomes free.

V. DESIGN

IP Masquerading multiple internal networks

Masquerading more than one internal network is fairly simple. You need to first make sure that all of your networks are running correctly (both internal and external). You then need to enable traffic to pass to both the other internal interfaces and to be MASQed to the Internet. Next, you need to enable Masquerading on the INTERNAL interfaces. This example uses a total of THREE interfaces: EXTIF stands for the eth0 interface which is the EXTERNAL connection to the Internet. INTIF stands for the eth1 interface and is the 192.168.0.0 network. Finally, INTIF2 stands for the eth2 interface and is the 192.168.1.0 network. Both INTIF and INTIF2 will be MASQed out of interface eth0 or EXTIF. In your rc.firewall-* ruleset next to the existing MASQ at the very end of the ruleset, add

The following:

A. *iptables support for multiple internal lans*

l

# 2.6.x and 2.4.x kernels with IPTABLES

#

# The following rules build upon the rc.firewall-iptables-stronger ruleset.

# Please see that ruleset in Section 6 for how all variables get set, etc.

#Enable internal interfaces to communication between each other

#

$IPTABLES -A FORWARD -i $EXTIF -o $INTIF2 -m state --state ESTABLISHED,RELATED \

-j ACCEPT

$IPTABLES -A FORWARD -i $INTIF -o $INTIF2 -m state --state ESTABLISHED,RELATED \

-j ACCEPT

$IPTABLES -A FORWARD -i $INTIF2 -o $INTIF -m state --state ESTABLISHED,RELATED \

-j ACCEPT

$IPTABLES -t nat -A POSTROUTING -o $EXTIF -j SNAT --to $EXTIP

B. *ipchains support for multiple internal lans*

# 2.2.x kernels with IPCHAINS

#

# The following rules build upon the rc.firewall-ipchains-stronger ruleset.

# Please see that ruleset in Section 6 for how all variables get set, etc.

#Enable internal interfaces to communication between each other

$IPCHAINS -A forward -i eth1 -d 192.168.0.0/24 -j ACCEPT

$IPCHAINS -A forward -i eth2 -d 192.168.1.0/24 -j ACCEPT

#Enable internal interfaces to MASQ out to the Internet

$IPCHAINS -A forward -j MASQ -i eth0 -s 192.168.0.0/24 -d 0.0.0.0/0

$IPCHAINS -A forward -j MASQ -i eth0 -s 192.168.1.0/24 -d 0.0.0.0/0

C. *Port Forwarding with IPMASQ*

The programs offer generic TCP and/or UDP port forwarding for IP Masquerade. This project is typically used with or as a replacement for specific IP MASQ modules to get a specific network traffic through the MASQ server.

With port forwarders, you can redirect data connections from the Internet to an internal, privately addressed machine behind your IP MASQ server. This forwarding ability includes network protocols such as TELNET, WWW, and SMTP. Protocols such as FTP, legacy ICQ.

internal machines typically CANNOT use the same "external" PORTFWed IP address to access a given internal" machine. To put it another way, PORTFW was only intended to be used with "external" computers on the Internet. If this is an issue

for you, you can also use the REDIR tool for older 2.2.x and 2.0.x kernels to let internal machines get redirected to the internal servers too.

The forwarding of non-NAT friendly traffic such as FTP server traffic to an internal MASQed FTP server, known as

PORTFW FTP,

"Port Forwarding is only called within masquerading functions so it fits inside the same IPFWADM/IPCHAINS rules. Masquerading is an extension to IP forwarding. Therefore, ipportfw only sees a packet if it fits both the input and masquerading ipfwadm rule sets."

What that means in English is that if you have a strong packet firewall running, PORTFW doesn't directly bypass any of that security.

l 2.0.x users will need to apply a simple kernel option patch to have access to then enable this via the normal kernel "make"

procedures.

D. *IPMASQADM-based PORTFWD'ing*

First, make sure you have the newest 2.2.x kernel uncompressed into /usr/src/kernel/linux. If you haven't already done this,

Next, you'll need to compile the 2.2.x kernel as shown in Section 3.2.2 section. Be sure to say "YES" to the IPPORTFW option when

you configure the kernel. Once the kernel compile is complete and you have rebooted, return to this section.

Now, compile and install the IPMASQADM tool:

cd /usr/src

tar xzvf ipmasqadm-x.tgz

cd ipmasqadm-x

make

make install

Now, for this example, we are going to allow ALL WWW Internet traffic (port 80) hitting your Internet TCP/IP address to be

forwarded to the internal Masqueraded machine at IP address 192.168.0.10.

PORTFW FTP: As mentioned above, there are two solutions for forwarding FTP server traffic to an internal MASQed PC. The first

solution *IS* a BETA level IP_MASQ_FTP module for 2.2.x kernels to PORT Forward FTP connections to an internal MASQed

FTP server. It should also be noted that the FTP kernel

module also supports the adding of additional PORTFW FTP ports on the fly without the requirement of unloading and reloading the

IP_MASQ_FTP module and thus breaking any existing FTP transfers. There are also examples and some additional information about PORTFWed FTP

connection below in the 2.0.x. kernel section.

NOTE: Once you enable a port forwarder on port 80, that port can no longer be used by the Linux IP Masquerade server. To be

more specific, if you have a WWW server already running on the MASQ server, a port forward will now give all Internet users the

WWW pages from the -INTERNAL- WWW server and not the pages on your IP MASQ server.

Anyway, to enable port forwarding for HTTPd:

l Edit the /etc/rc.d/rc.firewall-* ruleset and ENABLE the "Optional" "HTTP" sections in both the INPUT and OUTPUT

subsections.

l Add the following lines shown below JUST BELOW the "ipchains -P forward DENY" rule (in the "Optional

FORWARD section"). Be sure to replace the "$EXTIP" variable's contents with your EXTERNAL Internet IP address on the

IPMASQ server.

NOTE #2: If you get a dynamically assigned TCP/IP address from your ISP (PPP, DSL, Cable modems, etc.), you CANNOT load

this strong ruleset upon booting. You will either need to reload this firewall ruleset EVERY TIME you get a new IP address or make

your /etc/rc.d/rc.firewall-ipchains-stronger ruleset more intelligent. To do this for various types of connections such as PPP or DHCP

users,

/etc/rc.d/rc.firewall-*

http://www.ecst.csuchico.edu/~dranch/LINUX/ipmasq/ m-html/ipmasq-HOWTO-m.html (119 of 179)11/16/2005 5:51:53 PM

Linux IP Masquerade HOWTO

#echo "Enabling IPPORTFW Redirection on the external LAN.."

#

# This will forward ALL port 80 traffic from the external IP address

# to port 80 on the 192.168.0.10 machine

#

PORTFWIP="192.168.0.10"

/usr/sbin/ipmasqadm portfw -f

/usr/sbin/ipmasqadm portfw -a -P tcp -L $EXTIP 80 -R $PORTFWIP 80

That's it! Just re-run your /etc/rc.d/rc.firewall-* ruleset and test it out!

If you get the error message "ipchains: setsockopt failed: Protocol not available", you AREN'T running your new IPPORTFW

enabled kernel. Make sure that you moved the new kernel over, re-run LILO, and then reboot again. If you are sure you are running

your new kernel, run the command "ls /proc/net/ip_masq" and make sure the "portfw" file exists. If it doesn't, you must have made

an error when configuring your kernel. Try again.

PORTFW Redirection of Internal requests:

It should be mention that this IPMASQ HOWTO currently does *NOT* provide any explination or examples on how to use the

REDIR tool. If you need help setting it up for 2.2.x kernels, send me an email. For those who want to understand why PORTFW

cannot redirect traffic for both external and internal interfaces (what the REDIR tool fixes), here is an email from Juanjo that better

explains it.

From Juanjo Ciarlante

--

>If I use:

>

> ipmasqadm portfw -a -P tcp -L 1.2.3.4 80 -R 192.168.2.3 80

>

>Everything works great from the outside but internal requests for the same

>1.2.3.4 address fail. Are there chains that will allow a machine on

localnet

>192.168.2.0 to accesss www.periapt.com without using a proxy?

Actually not.

I usually setup a ipmasqadm rule for outside, *AND* a port

redirector for inside. This works because ipmasqadm hooks before

redir will get the eventual outside connection, _but_ leaves things

ok if not (stated by APPROPIATE rules).

The actual "conceptual" problem comes from the TRUE client (peer) IP

goal (thanks to masq) being in the same net as target server.

The failing scenario for "local masq" is :

client: 192.168.2.100

masq: 192.168.2.1

serv: 192.168.2.10

1)client->server packet

http://www.ecst.csuchico.edu/~dranch/LINUX/ipmasq/ m-html/ipmasq-HOWTO-m.html (120 of 179)11/16/2005 5:51:53 PM

Linux IP Masquerade HOWTO

a) client: 192.168.2.100:1025 -> 192.168.2.1:80 [SYN]

b) (masq): 192.168.2.100:1025 -> 192.168.2.10:80 [SYN]

(and keep 192.168.2.1:61000 192.168.2.100:1025 related)

c) serv: gets masqed packet (1b)

2)server->client packet

a) serv: 192.168.2.10:80 -> 192.168.2.100:1025 [SYN,ACK]

b) client: 192.168.2.100:1025 -> 192.168.2.10:80 [RST]

Now take a moment to compare (1a) with (2a).

You see, the server replied DIRECTLY to client bypassing masq (not

letting masq to UNDO the packet hacking) because it is in SAME net, so

the client resets the connection.

hope I helped.

Warm regards

Juanjo

VI. TESTING IP MASQUERADE

Finally, it's time to give IP Masquerading an official try after all this hard work. If you haven't already rebooted your Linux box, do so to make sure the machines boots ok, executes the /etc/rc.d/rc.firewall-* ruleset, etc. Next, make sure that both the internal LAN connection and connection of your Linux hosts to the Internet is okay.

Testing internal MASQ client PC connectivity

Step one : Testing internal MASQ client PC connectivity

From an internal MASQed computer, try pinging its local IP address (i.e. ping 192.168.0.10). This will verify that TCP/IP is correctly working on the local machine. Almost ALL modern operating systems have built-in support for the "ping" command. If this ping doesn't work, make sure that TCP/IP is correctly configured on the MASQed PC .the output should look something like the following (hit Control-C to abort the ping):

```
------------------------------------
masq- client# ping 192.168.0.10
PING 192.168.0.10 (192.168.0.10): 56 data bytes
64 bytes from 192.168.0.10: icmp_seq=0 ttl=255
time=0.8 ms
64 bytes from 192.168.0.10: icmp_seq=1 ttl=255
time=0.4 ms
64 bytes from 192.168.0.10: icmp_seq=2 ttl=255
time=0.4 ms
64 bytes from 192.168.0.10: icmp_seq=3 ttl=255
time=0.5 ms
^C
--- 192.168.0.10 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet
loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
```

Testing internal MASQ client to MASQ server connectivity

Step Two: Testing internal MASQ client to MASQ server connectivity

Next, from the same internal MASQed computer, try pinging the the IP address of the Linux MASQ server's INTERNAL interface (i.e. ping 192.168.0.1). This will verify that TCP/IP is correctly working on both the local and Linux MASQ machine. Almost ALL modern operating systems have built-in support for the "ping" command. If this ping doesn't work, make sure that TCP/IP is correctly configured on the MASQed Server. The output should look something like the following (hit Control-C to abort the ping):masq-client# ping 192.168.0.1

```
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=255
time=0.8 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255
time=0.4 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=255
time=0.4 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=255
time=0.5 ms
^C
--- 192.168.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet
loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
```

If the ping failed, check the network connection between the MASQ server and the PC. If it's a DIRECT Ethernet connection (no hub or switch), you MUST have a "Ethernet cross-over cable". These cables are common and can be found at any computer store. Without this cable, the NICs (network cards) will not give you a "LINK" light. If you are using a hub or switch, make sure the ports connected to the MASQ server and MASQed client machine have a LINK light. If they do and the pings STILL don't work or there is a lot of packet loss, try different ports on the hub/switch (it not all that uncommon to have hub/switch ports die). Finally, if things still don't work perfectly, try replacing each of the NICs in the machines.

Testing internal MASQ server connectivity

Step Three : Testing internal MASQ server connectivity

On the MASQ server, ping the internal IP address of the MASQ server's network interface card (i.e. ping 192.168.0.1). If this ping doesn't work, make sure that TCP/IP is correctly configured on the MASQed Server. The output should look something like the following (hit Control-C to abort the ping):

```
--------------------------------------
masq-server# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=255
time=0.8 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255
time=0.4 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=255
time=0.4 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=255
time=0.5 ms
^C
--- 192.168.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet
loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
---------------------------------------
```

Testing internal MASQ server to MASQ client connectivity

l Step Four : Testing internal MASQ server to MASQ client connectivity

Next from MASQed server, try pinging the IP address of one of the internal MASQ client computers (i.e. ping 192.168.0.10). This will verify that TCP/IP is correctly working on both the local server machine and on the MASQ client machine. If this ping doesn't work, make sure that TCP/IP is correctly configured on the MASQed PC. If the ping does work, the output should look something like the following (hit Control-C to abort the ping):

```
------------------------------------
masq-server# ping 192.168.0.10
PING 192.168.0.10 (192.168.0.10): 56 data bytes
64 bytes from 192.168.0.10: icmp_seq=0 ttl=255
time=0.8 ms
64 bytes from 192.168.0.10: icmp_seq=1 ttl=255
time=0.4 ms
64 bytes from 192.168.0.10: icmp_seq=2 ttl=255
time=0.4 ms
64 bytes from 192.168.0.10: icmp_seq=3 ttl=255
time=0.5 ms
^C
--- 192.168.0.10 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet
loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
------------------------------------
```

Testing External MASQ server Internet connectivity

Step Five : Testing external MASQ server to Internet connectivity

From the MASQ server, ping the external IP address of the MASQ server's EXTERNAL network interface that is connected to the Internet. This address might be a Ethernet interface, a PPP interface, etc. connection to your ISP. If you don't know what this external IP address is, run the Linux command "/sbin/ifconfig" on the MASQ server itself to get the Internet address. The output should look something like the following (we are looking for the IP address of eth0):

```
------------------------------------
eth0 Link encap:Ethernet HWaddr 00:08:C7:A4:CC:5B
inet addr:12.13.14.15 Bcast:12.13.14.255
Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST
MTU:1500 Metric:1
```

```
RX packets:6108459 errors:0 dropped:0 overruns:0 frame:0
TX packets:5422798 errors:8 dropped:0 overruns:0 carrier:8
collisions:4675 txqueuelen:100
Interrupt:11 Base address:0xfcf0
------------------------------------
```

As you can see from the above, the external IP address is "12.13.14.15" for this example. So, now that you have your IP address after running the "ipconfig" command, ping your external IP address. This will confirm that the MASQ server has full network connectivity. The output should look something like the following (hit Control-C to abort the ping):

```
-------------------------------------
masq-server# ping 12.13.14.15
PING 12.13.14.15 (12.13.14.15): 56 data bytes
64 bytes from 12.13.14.15: icmp_seq=0 ttl=255 time=0.8 ms
64 bytes from 12.13.14.15: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 12.13.14.15: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 12.13.14.15: icmp_seq=3 ttl=255 time=0.5 ms
^C
--- 12.13.14.15 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
-------------------------------------
```

If either of these tests doesn't work, you need to go back and double check your network cabling and verify that the two network interfaces on the MASQ server are seen in "dmesg". An example of this output would be the following towards the END of the "dmesg" command:

```
-------------------------------------
.
.
PPP: version 2.3.7 (demand dialling)
TCP compression code copyright 1989 Regents of the University of California
PPP line discipline registered.
3c59x.c:v0.99H 11/17/98 Donald Becker
http://cesdis.gsfc.nasa.gov/linux/drivers/vortex.html
eth0: 3Com 3c905 Boomerang 100baseTx at 0xfe80, 00:60:08:a7:4e:0e, IRQ 9
8K word-wide RAM 3:5 Rx:Tx split, autoselect/MII interface.
MII transceiver found at address 24, status 786f.
Enabling bus-master transmits and whole-frame receives.
eth1: 3Com 3c905 Boomerang 100baseTx at 0xfd80, 00:60:97:92:69:f8, IRQ 9
8K word-wide RAM 3:5 Rx:Tx split, autoselect/MII interface.
MII transceiver found at address 24, status 7849.
Enabling bus-master transmits and whole-frame receives.
Partition check:
sda: sda1 sda2 < sda5 sda6 sda7 sda8 >
sdb:
..-------------------------------------
```

Also be sure that the cabling is correct (Ethernet: the NICs connecting the external MASQ server to your ISP has the "link" light lit up). Finally, make sure that TCP/IP is correctly configured on the MASQed Server.

Testing internal MASQ client to external MASQ server connectivity

Step Six : Testing internal MASQ client to external MASQ server connectivity

From an internal MASQed computer, ping the IP address of the MASQ server's EXTERNAL TCP/IP address obtained in Step FIVE above. This address could be from your Ethernet, PPP, etc. interface which is ultimately the address connected to your ISP. This ping test will prove that Linux masquerading (ICMP Masquerading specifically) and IP forwarding is working.

If everthing thing is working correctly, the output should look something like the following (hit Control-C to abort the ping):

```
-------------------------------------
masq-client# ping 12.13.14.15
PING 12.13.14.15 (12.13.14.15): 56 data bytes
64 bytes from 12.13.14.15: icmp_seq=0 ttl=255 time=0.8 ms
64 bytes from 12.13.14.15: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 12.13.14.15: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 12.13.14.15: icmp_seq=3 ttl=255 time=0.5 ms
^C
--- 12.13.14.15 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
-------------------------------------
```

If this test doesn't work, first make sure that the "Default Gateway" on the MASQed PC is pointing to the IP address on the MASQ -SERVERs- INTERNAL NIC. Also double check that the /etc/rc.d/rc.firewall-* script was run without any errors. Just as a test, try re-running the /etc/rc.d/rc.firewall-* script now to see if it runs OK. Also, though most kernels support it by default, make sure that you enabled "ICMP Masquerading" in the kernel configuration and "IP Forwarding" in your /etc/rc.d/rc.firewall-* script.

If you still can't get things to work, take a look at the output from the following commands run on the Linux MASQ SERVER: "ifconfig" : Make sure the interface for your Internet connection (be it ppp0, eth0, etc.) is UP and you have the correct IP address for the Internet connection. An example of this output is shown in STEP FIVE above. m "netstat -rn" : Make sure your default gateway (the column with an IP address in the Gateway column) is set. An example of this output might look like:

```
-------------------------------------
masq-server# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
192.168.0.1 0.0.0.0 255.255.255.255 UH 0 16384 0 eth1
12.13.14.15 0.0.0.0 255.255.255.255 UH 0 16384 0 eth0
12.13.14.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
192.168.0.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
127.0.0.0 0.0.0.0 255.0.0.0 U 0 16384 0 lo
0.0.0.0 12.13.14.1 0.0.0.0 UG 0 16384 0 eth0
-------------------------------------
```

Notice the very LAST line that starts with 0.0.0.0? Notice that it also has an IP address in the "Gateway" field?

You should specify an IP address for your specific setup in that field (this is typically done automatically when your Internet connection is enabled).

m "cat /proc/sys/net/ipv4/ip_forward" : Make sure it says "1" so that Linux forwarding is enabled m Run the command "/sbin/ipchains -n -L" for 2.2.x users or "/sbin/ipfwadm -F -l" for 2.0.x users. Specifically, look for the FORWARDing section to make sure you have MASQ enabled. An example of an IPCHAINS output might look like for users using the SIMPLE rc.firewall-* ruleset:

```
------------------------------------
.Chain forward (policy REJECT):
target prot opt source destination ports
MASQ all ------ 192.168.0.0/24 0.0.0.0/0 n/a
ACCEPT all ----l- 0.0.0.0/0 0.0.0.0/0 n/a
```

VII. CONCLUSION

Masquerade NAT is used to allow your private network to hide behind, as well as be represented by, the address bound to the public interface. In many situations, this is the address that has been assigned by an Internet Service Provider (ISP), and the address may be dynamic in the case of a Point-to-Point Protocol (PPP) connection. This type of translation can only be used for connections originating within the private network destined for the outside public network. Each outbound connection is maintained by using a different source IP port number.

Masquerade NAT allows workstations with private IP addresses to communicate with hosts on the Internet using iSeries server. iSeries server has an IP address assigned by the local ISP as its Internet gateway. The term locally attached machine is used to refer to all machines on an internal network regardless of the method of attachment (LAN or WAN) and regardless of the distance of the connection.

VIII. REFERENCES

[1] Phifer, L., “IP Security and NAT: Oil and Water?”, ISP-Planet,June 15, 2000. http://www.isp-planet.com/technology/nat_ipsec.html

[2] Phifer, L., “Realm-Specific IP for VPNs and Beyond”, ISP-Planet, June 23, 2000. http://www.isp-planet.com/technology/rsip.html

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

[4] http://www.ibiblio.org/pub/linux/docs/howto/other-formats/pdf/IP-Masquerade-HOWTO.pdf

[5] http://www.bglug.ca/articles/nat_and_ip_masquerade.pdf

[6] http://estigia.fi-b.unam.mx/Linux/pdf/prared17.pdf

[7] http://www.comptechdoc.org/independent/networking/guide/netipmasq.html

[8] http://www.comptechdoc.org/independent/networking/guide/netipmasq.html