The IP Network Address Translator (Nat): Preliminary Design

Paul F. Tsuchiya

Bell Communications Research

tsuchiya@thumper.bellcore.com

Abstract

The two most compelling problems facing the IP Internet are IP address depletion and scaling in routing. This paper discusses the characteristics of one of the proposed solutions—address reuse. The solution is to place Network Address Translators (Nat) at the borders of stub domains. Each Nat box has a small pool of globally unique IP addresses that are dynamically assigned to IP flows going through Nat. The dynamic assignment is coordinated with the Domain Name Servers. The IP addresses inside the stub domain are not globally unique—they are reused in other domains, thus solving the address depletion problem. The pool of IP addresses in Nat is from a subnet administered by the regional backbone, thus solving the scaling problem. The main advantage of Nat is that it can be installed without changes to routers or hosts. This paper presents a preliminary design for Nat, and discusses its pros and cons.

1.0 Introduction

The two most compelling problems facing the IP Internet are IP address depletion and scaling in routing. Numerous solutions have been proposed, such as increasing the size of the IP address, using completely flat IP addresses, changing the structure of IP addresses, and even abandoning IP and switching to OSI [Ch]. Unfortunately, all of these solutions require changes to routers, hosts, or both.

Among the proposed solutions is the notion of address reuse. This solution takes advantage of the fact that a very small percentage of hosts in a stub domain[1] are communicating outside of the domain at any given time. Indeed, many (if not most) hosts never communicate outside of their stub domain. Because of this, IP addresses inside a stub domain, that need not be globally unique or known externally, can be dynamically

1. A stub domain is a domain, such as a corporate network, that only handles traffic originated by or destined to hosts in the domain.

translated into a small pool of IP addresses that are globally unique when outside communications is required.

This solution has the disadvantage of taking away the end-to-end significance of an IP address, and making up for it with increased state in the network. There are various work-arounds that minimize the potential pitfalls of this. Indeed, connection-oriented protocols are essentially doing address reuse at every hop.

The huge advantage of this approach is that it can be installed incrementally, without changes to either hosts or routers[2]. The only required changes are to Domain Name System (DNS) server implementations in the stub domain. As such, this solution can be implemented and experimented with quickly. If nothing else, this solution can serve to provide temporarily relief while other, more complex and far-reaching solutions are worked out.

2.0 Overview of Nat

The design presented in this paper is called Nat, for Network Address Translator. Nat is a box or router function that can be configured as shown in figure 1. The upper configurations require no host or router modifications. The lower configuration requires a modification to the stub border router.

Nat's basic operation is as follows. The addresses inside a stub domain can be reused by any other stub domain. For instance, a single Class A address could be used by many stub domains. At each exit point between a stub domain and backbone, Nat is installed. Each Nat is assigned a small pool of globally unique IP addresses (each Nat has a separate pool). These IP addresses are dynamically assigned to IP "flows" going through Nat.

For instance, in the example of figure 2, both stubs A and B internally use class A address 42.0.0.0. Stub A's Nat is assigned the class C address 198.76.29.0, and Stub B's Nat is assigned the class C address 198.76.28.0[3]. The class C addresses are globally unique—no other Nat boxes can use them.

When stub A host 42.33.96.5 wishes to exchange packets with stub B host 42.81.13.22 ("al.nxb.com"), it sends a Domain Name System (DNS) query to the DNS in stub B (1). DNS knows that the internal address

2. A few unusual applications may require changes, and hosts that communicate outside their domain to hosts that do not have permanent assignments must use DNS.

3. In actuality, for the purposes of scaling in the backbones, the Nat address would probably be subnetted class B addresses.

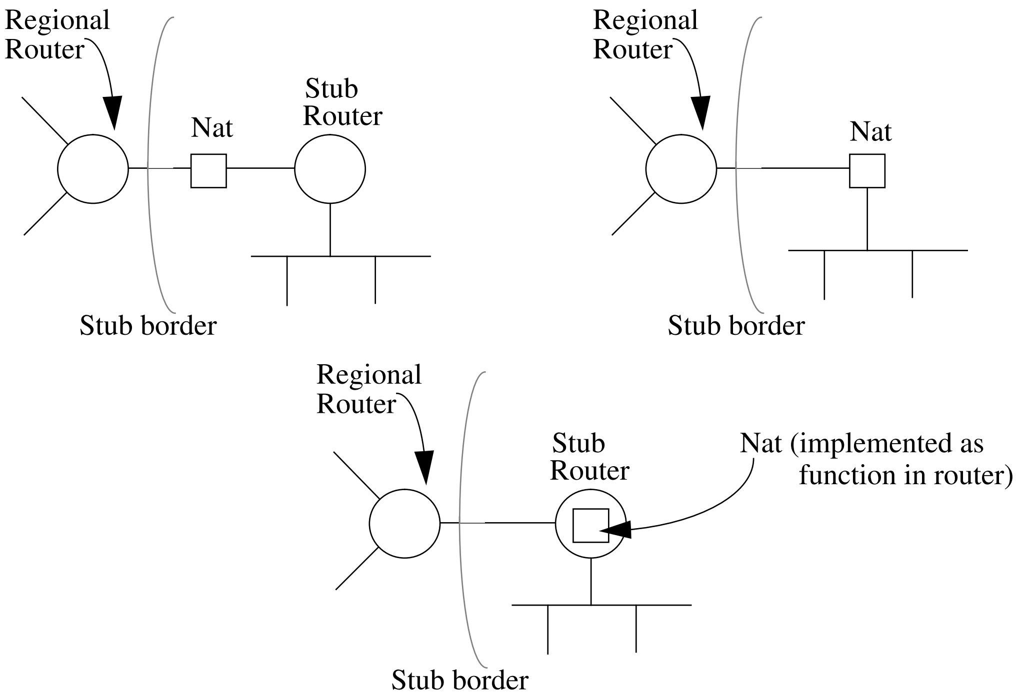

Figure 1: Nat Configurations

for "al.nxb.com" is 42.81.13.22, but let's assume that there is no external address assigned for "al.nxb.com". DNS would then send a query to Nat asking to have an address assigned (2). Nat finds an unassigned address in its pool, 198.76.28.4, and returns it to DNS (3), that in turn uses this address to answer the original DNS query (4).

42.33.96.5 then sends a packet to "al.nxb.com" with destination address 198.76.28.4. When this packet reaches stub A's Nat, it assigns an externally unique address (198.76.29.7) from the pool to 42.33.96.5, and translates the source address of the IP header with the new address[4]. This packet is routed through the backbones to the stub B Nat, which translates the destination address of the IP header to be the internally known address, and the packet is sent to "al.nxb.com". Likewise, IP packets on the return path go through similar address translations.

4. Note that both the IP and TCP checksums must be modified. This does not require a complete recalculation—only an incremental recalculation.

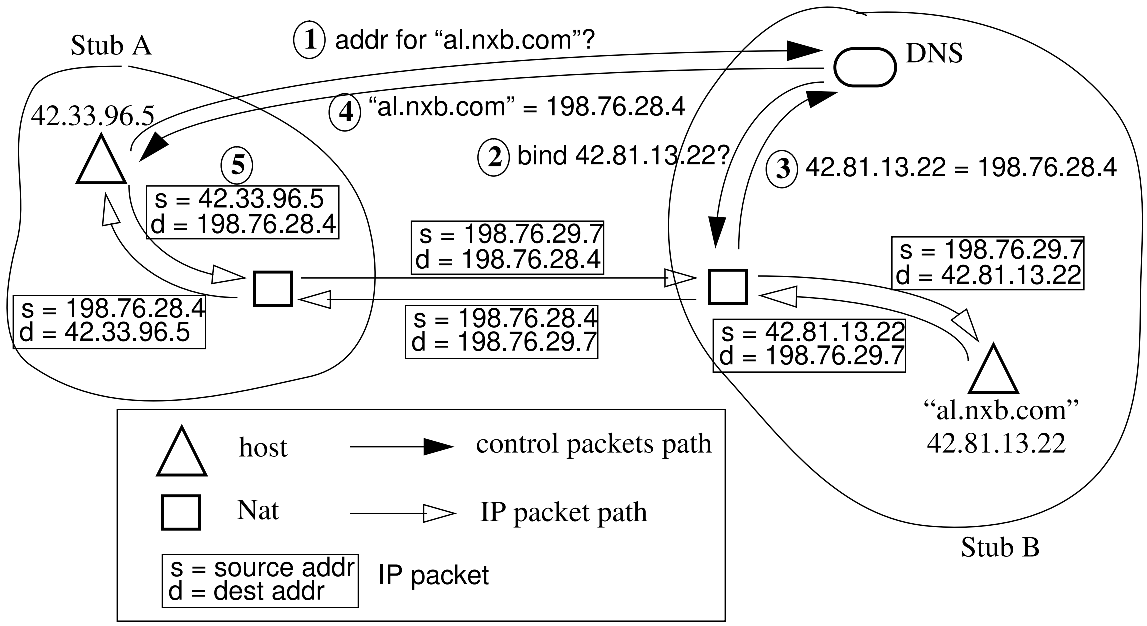

Figure 2: Basic Nat Operation

Notice that this requires no changes to hosts or routers. For instance, as far as the stub A host is concerned, 198.76.28.4 is the address used by "al.nxb.com". The address translations are completely transparent. Only the Domain Name Server requires modification.

Of course, this is just a simple example. There are numerous issues to be explored. In the next section, we discuss various aspects of Nat.

3.0 Various Aspects of Nat

3.1 Address Spaces

Partitioning of Reusable and Non-reusable Addresses

For Nat to operate properly, it is necessary to partition the IP address space into two parts—the reusable addresses used internal to stub domains, and the globally unique addresses. We call the reusable address *local addresses*, and the globally unique addresses *global addresses*. Any given address must either be a local address or a global address. There is no overlap.

The problem with overlap is the following. Say a host in stub A wished to send packets to a host in stub B, but the local addresses of stub B overlapped the local addressees of stub A. In this case, the routers in stub A would not be able to distinguish the global address of stub B from its own local addresses.

Initial Assignment of Local and Global Addresses

Theoretically all stubs could use the same class A address locally. However, existing stubs already have unique addresses assigned internally. It is difficult and takes time to change all addresses in a stub. Therefore, at least initially, existing address assignments should be defined as local addresses. A block of unassigned class B addresses should be defined as global addresses. These would be assigned to Nat boxes (on a subnetted basis). A single class A address should also be defined as local. This class A would be given to new stubs, who would be expected to install Nat when they connect to the IP Internet. Over time, existing stubs should install Nat and transition their existing address to the class A address. Once the transition was complete, the stub could give back its old addresses, which would then become global.

Scaling

One assignment strategy for global addresses goes as follows. Each regional (bottom level backbone) would be assigned a global class B address. The regional would then subnet the class B address among Nat boxes that connected to it. If, for instance, each Nat box required an average of 250 global addresses in its pool (not at all an unreasonable estimate), then the class B address could be subnetted among 250 or Nat boxes. Even if most stubs had two connections to the regional (thus requiring two Nat boxes, and two pools of addresses), 125 stubs could be subnetted into one class B address. The regional could then advertise one class B address to other backbones, rather than 125 separate addresses, as it does now. This would shrink current routing tables from several thousand entries to tens of entries.

3.2 Address Assignments

Permanent Address Assignments

Not all address assignments made by Nat should be dynamic. Any hosts that communicate outside the stub frequently should be given permanent assignments. Such hosts include DNS servers, email distributors, and anonymous FTP repository hosts. Indeed, stubs that implement security by limiting outside communications to a small number of “secure” hosts do not need dynamic assignment at all. Of course, the

DNS servers **must** have permanent assignments, because it is through the DNS servers that dynamically assigned global addresses become known.

The global address pool (or just pool), then, is partitioned into two parts, the static pool and the dynamic pool. The assignments from the static pool are configured in both the Nat box and the DNS server. That way, the DNS server does not need to query the Nat box for the assignment.

Choosing an Assignment

When Nat receives a request from DNS for an assignment, or when a packet arrives from the stub destined for the backbone that does not have an assignment, Nat must choose an assignment from the dynamic pool. The task of choosing an assignment from the pool can be tricky. The goal is to 1) minimize prematurely destroying assignments while 2) maximizing address utilization and 3) minimizing complexity. The primary goal is the first one, provided of course that we can make significant efficiency gains in address utilization over current practice.

The simplest algorithm for choosing an assignment is to choose the address that has not been in active use the longest. To do this, Nat would save the time that the last packet was seen for each of its assignments. When a new assignment is needed, Nat chooses the one with the oldest time. We call this algorithm *IP-based*, because it bases its activity estimate purely on the timing of individual IP packets.

The IP-based algorithm assumes (incorrectly, of course) that the usage behavior of all assignments is the same. For instance, it would choose the assignment for a TCP connection that has been idle for two hours but never issued a close, over a TCP connection that issued a close one hour ago and has been idle ever since. Clearly, it would be preferable to destroy the one-hour old closed connection over the two-hour old open connection[5]. However, since the IP-based algorithm does not monitor connection status, it cannot know this.

If applications where a valid assignment may remain idle for long periods of time exist, then the IP-based algorithm must maintain a large pool to insure that the oldest assignment is not still active. (If a limited number of hosts have these kinds of applications, then a possible solution would be to give these hosts permanent assignments.)

---

5. TCP does not have a keep-alive function, so a TCP connection can remain open without exchanging packets indefinitely.

An alternative approach would be an algorithm that monitors connection status and partitions assignments into two classes—those that have seen an end-of-connection indication and those that have not. The ended-connection assignments would become available for re-assignment after a relatively short idle time, say one hour. The open assignments would become available for re-assignment after a long idle time, perhaps one week. One way to implement this would be to consider the idle time of an ended-connection association to be x times the actual idle time, and then compare all idle times directly. We call this the *connection-based* algorithm.[6]

It is impossible to know which assignment algorithm to use without knowing how applications behave. Since Nat will initially be experimental, both algorithms can be experimented with. Since each Nat box can independently choose it's own algorithm, multiple algorithms can be experimented with simultaneously.

Incorrect Assignments

In the simple mechanism described in section 2, the participation of DNS was limited to only absolutely necessary functions. Further, each Nat box translates addresses and routes packets without discriminating on the backbone-side address. In other words, the reuse of global addresses by Nat is not tightly coordinated with the use of addresses by hosts. As a result, it is entirely possible for a host A to use an address that it thinks is for host B, but that in fact Nat has reassigned to host C.

Note that even without address reuse, there always exists the possibility that a host uses the wrong address. For instance, DNS can be incorrectly configured so that the IP address it returns does not belong to the host named in the query. Usually, the application protocol or the human user will discover the error. Still, Nat increases the probability of misdelivering packets.

Notice that a protocol such as X.25 is basically an address reuse scheme similar in some respects to Nat. In this case, the X.121 address corresponds to the Domain Name, and the Virtual Circuit Identifier (VCI) corresponds to the assigned IP address. X.25 for all practical purposes does not have the mis-addressed packet problem like Nat does, because the assignment of VCIs is tightly coordinated by all components on the path, including hosts. It follows, then, that the problem of mis-addressed packets in Nat is not a problem with address reuse per se, but a problem of the style of implementation forced upon us by the decision to not modify hosts.

6. Connection-less applications can only use the IP-based algorithm

All cases of mis-addressed packets are the result of hosts (or DNS) caching addresses longer than Nat. This comes about because Nat, through DNS, can give a host an address to use, but cannot later remove it. DNS should therefore always set the Time-to-Live (TTL) to a value slightly smaller than the minimum time that Nat will maintain an assignment [Lo]. This of course does not prevent a host from caching the assignment for a longer period of time. For instance, the host may start up an application that runs for a long time, sends very occasional UDP packets, and uses the IP address that it was originally invoked with during its existence. A small TTL only prevents a DNS server from storing the query beyond this time, thereby preventing it from giving the assignment to another DNS server after it has expired.

Another style of implementation, that involves much more coordination with DNS but that reduces the probability of mis-addressed packets, is briefly described as follows. Instead of distinguishing assignments by only stub-side addresses, Nat distinguishes on both sides. In other words, if a packet does not have an expected source *and* destination address, it is dropped.

In order for the Nat boxes on both ends to learn both stub- and backbone-side addresses, DNS gets involved on both ends. When a host H_a wants to send packets, it queries its local DNS server D_a. Rather than immediately send a query to the destination DNS server, it queries the Nat box N_a in its domain and gets an assignment A_a. This assignment is then conveyed in the query to the DNS server D_b on the destination side. D_b then conveys the address A_a to the destination Nat box N_b, which associates the source global address A_a with the destination global address A_b that it assigns. When D_b answers the query, D_a informs N_a of the address A_b, so now both Nat boxes have both A_a and A_b associated with their assignments. As a result, hosts that previously used either A_a or A_b will now not be able to use them, and packets will not be mis-delivered (just dropped). To successfully communicate, these hosts will have to again query their DNS server.

If a third host H_c wants to send packets to host H_b with global address A_c, DNS server D_b will need to inform N_b of the new address A_c so that N_b will now associate both A_a and A_c with A_b. Note that if only one side has implemented Nat, the DNS server on the Nat side will need to query the DNS server on the non-Nat side to learn the expected IP address of other side. This extra query is needed because DNS queries normally only carry the domain name of the query originator, not the IP address.

This two-sided design involves more overhead and complexity than the one-sided design, and may turn out not to be necessary. Its inclusion in Nat is a matter for further debate and experimentation. Notice that the two-sided technique is not helpful with permanent assignment.

3.3 Running Routing Algorithms Across Nat

Of course, in order for Nat to be transparent to the border routers of the backbone and the stub, the border routers must be believe that they are exchanging routing information with each other in the usual way. In other words, the stub border router must think that it is sending routing information about its internal (local) addresses to the backbone border router. However, Nat must intercept the routing information from the stub border router and replace the local address information with address information reflecting its global pool. However, global information that Nat receives from the stub border router must be passed through unchanged. Routing information from the backbone border router to Nat should always be passed through unchanged.

3.4 Multiple Nat Boxes and DNS Servers

All of the previous descriptions assume that each stub has just one Nat box and one DNS server. However, each stub/backbone entry/exit point needs to have a Nat box. In many, if not most cases, an IP packet can potentially travel through more than one border router. For this to work in the context of Nat, every Nat box that might potentially handle the packets of a given connection must know the assignment.

In addition, any given host may have multiple DNS servers (primary and backup, for instance). In what follows, we describe the mechanisms necessary to make multiple Nat boxes and DNS servers work.

Need for Nat Cliques

The Nat boxes of a stub are partitioned into one or more possibly overlapping groups, each with one or more Nat boxes. We call these groups Nat cliques. A Nat clique is a group of Nats such that a packet addressed with an assignment from one of the Nats in the clique can potentially be routed through any of the Nats in the clique. Therefore, the formation of a Nat clique depends on both intra-domain and inter-domain routing, primarily inter-domain.

There can be many reasons why such cliques form. For instance, assume that a stub has two attachments each to both NSFNET and MILNET. Assume also that the addresses assigned to the Nat boxes corresponding to each backbone are hierarchically formed to imply routing through that backbone.

Therefore, packets assigned by the NSFNET-attached Nats will go through NSFNET, and packets assigned by the MILNET-attached Nats will go through MILNET. In this case, there are two Nat cliques.

The Nats in a Nat clique are divided into two types—those that can assign and receive addresses for the clique, and those that only receive addresses for the clique. We call these clique-assigning Nats and clique-receiving Nats. Although it is not absolutely necessary, every Nat should be a clique-assigning Nat for at least some clique (for instance, for robustness in case all other Nats in a clique fail).

The reason for having two types of Nats is as follows. Consider the previous example, but modify it so that it is possible to alternate route packets with the MILNET-derived assignments through NSFNET, but it is not possible to alternate route packets with the NSFNET-derived assignments through MILNET. In this case, the NSFNET Nats must be aware of the MILNET assignments, in case such packets are alternate routed through them, but the MILNET Nats do not need to be aware of the NSFNET Nat assignments. There are therefore two cliques. The "NSFNET" clique has just the two NSFNET-attached Nats, and both are clique-assigning Nats. The "MILNET" clique has all four Nats, but the MILNET-attached Nats are clique-assigning Nats, and the NSFNET-attached Nats are clique-receiving Nats.

Operation of Nat Cliques and DNS Cliques

Nat and DNS cliques require the following configuration information. The DNS server contains a list for each Nat clique. Within each list is the IP address of the clique-assigning Nats in the clique. Each Nat contains a list for each Nat clique for which it is a clique-assigning member. This list contains the IP address of every Nat in the clique (both clique-assigning and clique-receiving). Clique-assigning Nats must also contain a list for each DNS clique. A DNS clique is a group of DNS servers that can potentially answer a query for the same hosts. Although it is only necessary as a configuration-correctness mechanism, each Nat may contain a list for each Nat clique for which it is a clique-receiving member. This list contains the IP address for every clique-assigning member of the clique. All of the configuration information must be stored in non-volatile memory (or be learnable upon booting).

Every Nat box has one or more unique pools of global addresses from which it can make assignments. By not having Nat boxes share global addresses, we eliminate the need for coordination of address assignment where one Nat box has to check with others to make sure that a particular assignment can be made. This has the negative effect of requiring a larger pool in each Nat box than what otherwise would be necessary (to insure that any Nat box doesn't run out of addresses to assign). However, the increase is not that much (certainly not linear with the number of Nat boxes), since assignments can be spread over the Nats in a clique.

The reason that a Nat box may have multiple pools is because a backbone may assign multiple address prefixes to a single stub for the purposes of policy routing. For instance, assume that a regional backbone is attached to both MILNET and NSFNET. If the regional network maintains two address prefixes, and advertises one of them to MILNET and the other to NSFNET, then packets with the MILNET-advertised address will be routed through MILNET and vice versa. By receiving multiple addresses in a query, a host (or user) has the power to choose the backbone network [Ts1].

When a DNS server receives a DNS query, it sends a Nat-assignment query to one clique-assigning Nat in each Nat clique. It should round-robin the queries among the clique-assigning Nats in each clique to evenly spread the assignment load. The Nat receiving the query makes an assignment, returns the assignment to the DNS server, sends an assignment notification message to all DNS servers that are members of the DNS cliques that the requesting DNS server belongs to, and sends an assignment notification message to all of the Nats in its cliques indicating the new assignment. On rare occasions, the Nat receiving the query may have no addresses available for assignment. In this case, Nat returns a NULL assignment, and the DNS may either query another Nat box in the clique, or return a failure in its DNS response.

When the assignments are returned to the DNS servers, they have expiration times associated with them. The assigning Nat boxes must not reassign the address until the expiration time has elapsed. The DNS servers must not set the TTL (Time-to-Live) field in the DNS response to longer than the shortest expiration time. If the DNS servers choose to cache the assignment, they must remove the cache entry by the shortest expiration time. Note that it is not necessary for the DNS servers to cache the entry, because if another query for the same host comes, the DNS server will query Nat boxes and receive the same assignments.

When Nat boxes receive assignment notifications, they keep the assignments until notified otherwise. This will occur when the assigning Nats reassign the addresses.

Nat assignment notifications must be reliable, because there is no refreshing (or timing out) of assignments by receiving Nats. Therefore, assignment notification messages must be acknowledged, and resent if no acknowledgment is received. Of course, if a receiving Nat has crashed, then no acknowledgment can be sent. Therefore, Nat boxes must be able to mark other Nat boxes as down after a number of attempted assignment notifications. Also, when Nat boots (comes up after crashing), it must contact all assigning

Nats in its cliques and receive all current assignments. This must also happen if Nats in a clique have been partitioned from each other, and the partition heals. Note that each Nat must have enough memory to hold all of the assignments of all of the Nats in all of their cliques.

Private Networks that Span Backbones

In many cases, a private network (such as a corporate network) will be spread over different locations and will use a public backbone for communications between those locations. In this case, it is not desirable to do address translation, both because large numbers of hosts may want to communicate across the backbone, thus requiring large global address pools, and because there will be more applications that depend on configured addresses, as opposed to going to a name server. We call such a private network a *backbone-partitioned stub*.

Backbone-partitioned stubs should behave as though they were a non-partitioned stub. That is, the routers in all partitions should maintain routes to the local address spaces of all partitions. Of course, the (public) backbones do not maintain routes to any local addresses. Therefore, the border routers must tunnel through the backbones using encapsulation. To do this, each Nat box will set aside one global address from the pool for tunneling. When a Nat box x in stub partition X wishes to deliver a packet to stub partition Y, it will encapsulate the packet in an IP header with a destination address from the pool of Nat box y that has been reserved for encapsulation. Then Nat box y receives a packet with that destination address, in decapsulates the IP header and routes the packet internally.

3.5 Various Header Manipulations

In addition to modifying the IP address, Nat must modify the IP checksum, the TCP checksum, places in ICMP and Telnet where the IP address appears, and perhaps other places where the IP address appears[7].

The checksum modifications to IP and TCP are simple and efficient. Since both use a one's complement sum, it is sufficient to calculate the arithmetic difference between the before-translation and after-translation addresses and add this to the checksum. The only tricky part is determining whether the addition resulted in a wrap-around (in either the positive or negative direction) of the checksum. If so, 1

7. The author knows of no other such places off hand, but there are undoubtedly some. Hopefully, most such applications will be discovered during experimentation with Nat.

must be added or subtracted to satisfy the one's complement arithmetic. Sample code (in C) for this is as follows:

```
unsigned 16bitInt checksum, diff;
unsigned char add;
       checksum = ~checksum;
       if (add)
       {      checksum += diff;
              if (checksum < diff)
                     checksum++; /* to account for positive zero crossing */
       }
       else
       {      if (checksum < diff)
                     checksum--; /* to account for negative zero crossing */
              checksum -= diff;
       }
       checksum = ~checksum;
```

Where `add` and `diff` are pre-calculated for each assignment as:

```
unsigned 32bitInt newAddress, oldAddress;
unsigned 16bitInt newCheck, oldCheck;
       newCheck = ((newAddress >> 16) & 0xffff) + (newAddress & 0xffff);
       oldCheck = ((oldAddress >> 16) & 0xffff) + (oldAddress & 0xffff);
       if (newCheck > oldCheck)
       {      add = 1;
              diff = newCheck - oldCheck;
       }
       else
       {      add = 0;
              diff = oldCheck - newCheck;
       }
```

The File Transfer Protocol (FTP) has a function called the PORT command that allows a user to specify that a file transfer take place from a different port of that from which the FTP session was established. The arguments to the PORT command include an IP address. If the IP address in the PORT command is the same as that of the host sending the PORT command, then the (local) IP address in FTP can be substituted with the (global) assigned IP address. If the IP address in the PORT command is different from that of the host sending the PORT command, but the IP address is local to the stub domain, then Nat can create an assignment for the IP address and substitute that. Either way, the TCP checksum must be recalculated (again) as indicated above. If the IP address in the PORT command is not from the local stub, then it should not be modified. Of course, if the FTP session is encrypted, the PORT command will fail.

If an ICMP message is passed through Nat, it may require two address modifications and three checksum modifications. This is because most ICMP messages contain part of the original IP packet in the body.

Therefore, for Nat to be completely transparent to the host, the IP address of the IP header embedded in the data part of the ICMP packet must be modified, the checksum field of the same IP header must correspondingly be modified, and the ICMP header checksum must be modified to reflect the changes to the IP header and checksum in the ICMP body. Of course, the normal IP header must also be modified as already described.

It is not entirely clear that the IP header information in the ICMP part of the body really need be modified. This depends on whether or not there is really any host code that looks at this IP header information[8]. Indeed, it may be useful to provide the exact header seen by the router or host that issued the ICMP message to aid in debugging. In any event, no modifications are needed for the Echo and Timestamp messages, and Nat should never need to handle a Redirect message.

3.6 Other Aspects of Nat

Global Routing and Addressing Issues

Over the short term, Nat provides scaling benefits by allowing for subnetting of stubs by backbone networks. In doing this, we essentially add a level of hierarchy to IP routing. We also introduce the coupling of route to address that the OSI community is now having to face. Namely, if an IP address is handed out by a backbone, and that backbone advertises that address as reachable through it, then routes will naturally go through that backbone. If an alternate route through another backbone is desired (for instance, because the primary route failed), that route may not be available.

Viewed another way, however, this coupling of route to address can actually be a feature rather than a bug. If a host or user wishes to route through one backbone vs. another, it can manipulate the choice by choosing the appropriate address. This would work as follows. When the DNS server queries the Nat boxes for assignments, it may get back multiple answers, one from each Nat clique, and possibly multiple assignments from a single Nat clique. These multiple answers essentially reflect reachability of the stub through multiple backbones. When the DNS server then returns the queries, the source host can choose the appropriate one. Indeed, with a small modification to TCP, addresses could be changed *during* a TCP connection to dynamically respond to failures.

8. In the theoretical worst case, an ICMP message could be sent concerning an FTP packet that contained a PORT command. In this case, modifications would be required to the PORT command and the TCP checksum, in addition to the fields already mentioned. In practice, this seems unnecessary.

The use of multiple addresses as a means of policy routing and scaling are discussed extensively in [Ts1]. The main point here is that the extra layer(s) of IP address hierarchy resulting from Nat make it possible to take advantage of multiple addresses.

Alternative Address Formats

In addition to the extra level of hierarchy introduced by Nat and described above, it may be desirable to break away from the class-A/B/C address format all together, for instance to introduce yet another level of hierarchy or to utilize address space more efficiently (for instance, using kampai addressing [Ts2]). Since local addresses are completely partitioned from global addresses, modifying addresses in the backbones would require changes to backbone routers only.

Dynamic Allocation of Nat Pool

The size of the pool of addresses needed by a Nat box varies dynamically. At certain times more addresses are needed than at others. If the Nat pools can be dynamically assigned to Nat boxes from a larger pool, then the benefits of statistical sharing can be realized. This suggests the possibility that the Nat pool could be dynamically assigned. Each Nat box could keep a pool large enough to handle most of its needs, but the Nat box could dynamically request more addresses from its backbone when necessary.

This could be done using the two-level kampai algorithm described in [Ts3]. That algorithm is for the purpose of assigning subnet numbers. Its main advantage is that it allows subnet numbers to be assigned efficiently without requiring advance knowledge of the size (in terms of number of hosts) and number of subnets. It does this by removing a bit from the mask when more space is needed in a subnet. This doubles the subnet's space. Since this algorithm is easily automated, it may be possible for Nat boxes to request and return address space in increments of powers of two.

Applications with IP-address Content

Any application that carries (and uses) the IP address inside the application will not work through Nat unless Nat knows of such instances and does the appropriate translation. It is not possible or even necessarily desirable for Nat to know of all such applications. And, if encryption is used then it is impossible for Nat to make the translation.

It may be possible for such systems to avoid using Nat, if the hosts in which they run are assigned global addresses. Whether or not this can work depends on the capability of the intra-domain routing algorithm and the internal topology. This is because the global address must be advertised in the intra-domain routing algorithm. With a low-feature routing algorithm like RIP, the host may require its own class C address space, that must not only be advertised internally but externally as well (thus hurting global scaling)[9]. With a high-feature routing algorithm like OSPF, the host address can be passed around individually, and can come from the Nat pool.

Privacy, Security, and Debugging Considerations

Unfortunately, Nat reduces the number of options for providing security. With Nat, nothing that carries an IP address or information derived from an IP address (such as the TCP-header checksum) can be encrypted. While most application-level encryption should be ok, this prevents encryption of the TCP header.

On the other hand, Nat itself can be seen as providing a kind of privacy mechanism. This comes from the fact that machines on the backbone cannot monitor which hosts are sending and receiving traffic (assuming of course that the application data in encrypted).

The same characteristic that enhances privacy potentially makes debugging problems (including security violations) more difficult. If a host is abusing the Internet is some way (such as trying to attack another machine or even sending large amounts of junk mail or something) it is more difficult to pinpoint the source of the trouble because the IP address of the host is hidden.

4.0 Conclusions

Nat may be a good short term solution to the address depletion and scaling problems. This is because it requires changes only to DNS, and can be installed incrementally. Nat has several negative characteristics that make it inappropriate as a long term solution, and may make it inappropriate even as a short term solution. Only implementation and experimentation will determine its appropriateness.

The negative characteristics are:

9. I say “may” rather than “must” because I don’t know enough about implementations of RIP to be sure.

1. It requires a sparse end-to-end traffic matrix. Otherwise, the Nat pools will be large, thus using up many addresses. While the expectation is that end-to-end traffic matrices are indeed sparse, experience with Nat will determine whether or not they are. In any event, future applications may require a rich traffic matrix (for instance, distributed resource discovery), thus making long-term use of Nat unattractive.

2. It may significantly increase the load on DNS. This is because DNS server will not be able to cache responses for as long. Currently, DNS represents a significant proportion of Internet traffic. Since current caching efficiency for DNS is not currently known, the DNS traffic increase may be light, or it may be heavy.

3. It increases the probability of mis-addressing.

4. It breaks certain applications (or at least makes them more difficult to run).

5. I won’t work with hosts that don’t use DNS (except for permanent assignments).

6. It hides the identity of hosts. While this has the benefit of privacy, it is generally a negative effect.

REFERENCES

[Ch] Chiappa, N., IETF Internet Draft, draft-chiappa-ipaddressing-00.txt, March, 1991.

[Lo] Lottor, M., “Domain Administrators Operations Guide”, RFC-1033, USC/Information Sciences Institute, November 1987.

[Mo] Mockapetris, P.V., “Domain names - implementation and specification”, RFC-1035, USC/Information Sciences Institute, November 1987.

[Ts1] Tsuchiya, P.F., “Robust and Efficient Policy Routing using Multiple Hierarchical Addresses”, TM-ARH-018519, Bellcore, March, 1991; submitted to SIGCOMM ‘91, September 1991.

[Ts2] Tsuchiya, P.F., “Efficient and Flexible Hierarchical Address Assignment”, TM-ARH-018495, Bellcore, February, 1991.

[Ts3 Tsuchiya, P.F., “On the Assignment of Subnet Numbers”, submitted as Internet-Draft, January 1991, submitted as RFC, March 1991.