                 The IP Network Address Translator (NAT)

Status of this Memo

   This memo provides information for the Internet community. This memo
   does not specify an Internet standard of any kind. Distribution of
   this memo is unlimited.

Abstract

   The two most compelling problems facing the IP Internet are IP
   address depletion and scaling in routing. Long-term and short-term
   solutions to these problems are being developed. The short-term
   solution is CIDR (Classless InterDomain Routing). The long-term
   solutions consist of various proposals for new internet protocols
   with larger addresses.

   It is possible that CIDR will not be adequate to maintain the IP
   Internet until the long-term solutions are in place. This memo
   proposes another short-term solution, address reuse, that complements
   CIDR or even makes it unnecessary. The address reuse solution is to
   place Network Address Translators (NAT) at the borders of stub
   domains. Each NAT box has a table consisting of pairs of local IP
   addresses and globally unique addresses. The IP addresses inside the
   stub domain are not globally unique. They are reused in other
   domains, thus solving the address depletion problem. The globally
   unique IP addresses are assigned according to current CIDR address
   allocation schemes. CIDR solves the scaling problem. The main
   advantage of NAT is that it can be installed without changes to
   routers or hosts. This memo presents a preliminary design for NAT,
   and discusses its pros and cons.

for a long time, we were the only ones who had had the idea).

1. Introduction

   The two most compelling problems facing the IP Internet are IP
   address depletion and scaling in routing. Long-term and short-term
   solutions to these problems are being developed. The short-term
   solution is CIDR (Classless InterDomain Routing) [2]. The long-term
   solutions consist of various proposals for new internet protocols
   with larger addresses.

   Until the long-term solutions are ready an easy way to hold down the
   demand for IP addresses is through address reuse. This solution takes
   advantage of the fact that a very small percentage of hosts in a stub
   domain are communicating outside of the domain at any given time. (A
   stub domain is a domain, such as a corporate network, that only
   handles traffic originated or destined to hosts in the domain).
   Indeed, many (if not most) hosts never communicate outside of their
   stub domain. Because of this, only a subset of the IP addresses
   inside a stub domain, need be translated into 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. (A few
   unusual applications may require changes). 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. Overview of NAT

   The design presented in this memo is called NAT, for Network Address
   Translator. NAT is a router function that can be configured as shown
   in figure 1. Only the stub border router requires modifications.

   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. If there
   is more than one exit point it is of great importance that each NAT
   has the same translation table.

```
     \ | /                        .                                    /
  +--------------+  WAN     .              +----------------+/
  |Regional Router|---------------------|Stub Router w/NAT|---
  +--------------+          .              +----------------+\
                            .                       |         \
                            .                       |  LAN
                            .              ---------------
                            .
                      Stub border
```
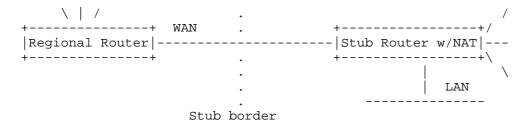
                   Figure 1: NAT Configuration

For instance, in the example of figure 2, both stubs A and B
internally use class A address 10.0.0.0. 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. The class C addresses are globally unique no
other NAT boxes can use them.

```
                              \ | /
                         +---------------+
                         |Regional Router|
                         +---------------+
                       WAN |        | WAN
                           |        |
          Stub A .............|....    ....|........... Stub B
                           |        |
          {s=198.76.29.7,^  |        |  v{s=198.76.29.7,
           d=198.76.28.4}^  |        |  v d=198.76.28.4}
              +----------------+      +----------------+
              |Stub Router w/NAT|      |Stub Router w/NAT|
              +----------------+      +----------------+
                      |                       |
                      | LAN              LAN  |
              -------------              -------------
                      |                       |
          {s=10.33.96.5, ^  |              |  v{s=198.76.29.7,
           d=198.76.28.4}^ +--+        +--+ v d=10.81.13.22}
                          |--|        |--|
                          /____\        /____\
                      10.33.96.5        10.81.13.22
```
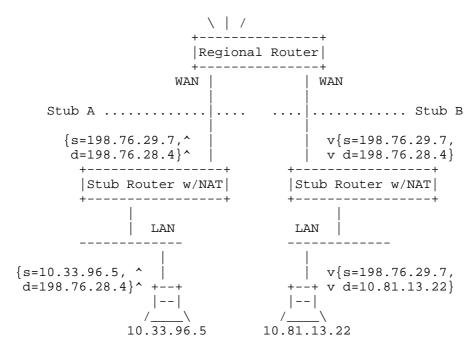
                   Figure 2: Basic NAT Operation

When stub A host 10.33.96.5 wishes to send a packet to stub B host
10.81.13.22, it uses the globally unique address 198.76.28.4 as
destination, and sends the packet to it's primary router. The stub
router has a static route for net 198.76.0.0 so the packet is
forwarded to the WAN-link. However, NAT translates the source address
10.33.96.5 of the IP header with the globally unique 198.76.29.7

before the package is forwarded. Likewise, IP packets on the return
path go through similar address translations.

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 the host in stub B. The address translations are
completely transparent.

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. 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

   A single class A address should be allocated for local networks. (See
   RFC 1597 [3].) This address could then be used for internets with no
   connection to the Internet. NAT then provides an easy way to change
   an experimental network to a "real" network by translating the
   experimental addresses to globally unique Internet addresses.

   Existing stubs which have unique addresses assigned internally, but
   are running out of them, can change addresses subnet by subnet to
   local addresses. The freed adresses can then be used by NAT for
   external communications.

3.2 Routing Across NAT

   The router running NAT should never advertise the local networks to
   the backbone. Only the networks with global addresses may be known
   outside the stub. However, global information that NAT receives from
   the stub border router can be advertised in the stub the usual way.

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 address tables, 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 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 destination address set
   to the global address of NAT box y that has been reserved for
   encapsulation. When NAT box y receives a packet with that destination
   address, it decapsulates the IP header and routes the packet
   internally.

3.3 Header Manipulations

   In addition to modifying the IP address, NAT must modify the IP
   checksum and the TCP checksum. Remember, TCP's checksum also covers a
   pseudo header which contains the source and destination address. NAT
   must also look out for ICMP and FTP and modify the places where the
   IP address appears. There are undoubtedly other places, where
   modifications must be done. Hopefully, most such applications will be
   discovered during experimentation with NAT.

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

```
void checksumadjust(unsigned char *chksum, unsigned char *optr,
int olen, unsigned char *nptr, int nlen)
/* assuming: unsigned char is 8 bits, long is 32 bits.
  - chksum points to the chksum in the packet
  - optr points to the old data in the packet
  - nptr points to the new data in the packet
*/
{
  long x, old, new;
  x=chksum[0]*256+chksum[1];
  x=~x;
  while (olen) {
    if (olen==1) {
      old=optr[0]*256+optr[1];
      x-=old & 0xff00;
      if (x<=0) { x--; x&=0xffff; }
      break;
    }
    else {
      old=optr[0]*256+optr[1]; optr+=2;
      x-=old & 0xffff;
      if (x<=0) { x--; x&=0xffff; }
      olen-=2;
    }
  }
  while (nlen) {
    if (nlen==1) {
      new=nptr[0]*256+nptr[1];
      x+=new & 0xff00;
      if (x & 0x10000) { x++; x&=0xffff; }
      break;
    }
    else {
      new=nptr[0]*256+nptr[1]; nptr+=2;
      x+=new & 0xffff;
      if (x & 0x10000) { x++; x&=0xffff; }
      nlen-=2;
    }
  }
  x=~x;
  chksum[0]=x/256; chksum[1]=x & 0xff;
}
```

The arguments to the File Transfer Protocol (FTP) PORT command
include an IP address (in ASCII!). If the IP address in the PORT
command is local to the stub domain, then NAT must substitute this.
Because the address is encoded in ASCII, this may result in a change
in the size of the packet (for instance 10.18.177.42 is 12 ASCII
characters, while 193.45.228.137 is 14 ASCII characters). If the new
size is the same as the previous, only the TCP checksum needs
adjustment (again). If the new size is less than the previous, ASCII
zeroes may be inserted, but this is not guaranteed to work. If the
new size is larger than the previous, TCP sequence numbers must be
changed too.

A special table is used to correct the TCP sequence and acknowledge
numbers with source port FTP or destination port FTP. The table
entries should have source, destination, source port, destination
port, initial sequence number, delta for sequence numbers and a
timestamp. New entries are created only when FTP PORT commands are
seen. The initial sequence numbers are used to find out if the
sequence number of a packet is before or after the last FTP PORT
command (delta may be increased for every FTP PORT command). Sequence
numbers are incremented and acknowledge numbers are decremented. If
the FIN bit is set in one of the packets, the associated entry may be
deleted soon after (1 minute should be safe). Entries that have not
been used for e.g. 24 hours should be safe to delete too.

The sequence number adjustment must be coded carefully, not to harm
performance for TCP in general. 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. Furthermore, the normal IP header must also be modified as
already described.

It is not entirely clear if the IP header information in the ICMP
part of the body really need to be modified. This depends on whether
or not any host code actually looks at this IP header information.
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.

   SNMP messages could be modified, but it is even more dubious than for
   ICMP messages that it will be necessary.

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). With
   a high-feature routing algorithm like OSPF, the host address can be
   passed around individually, and can come from the NAT table.

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 is 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. Conclusions

   NAT may be a good short term solution to the address depletion and
   scaling problems. This is because it requires very few changes 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:

1. It requires a sparse end-to-end traffic matrix. Otherwise, the NAT
   tables will be large, thus giving lower performance. 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 increases the probability of mis-addressing.

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

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

5. Problems with SNMP, DNS, ... you name it.

Current Implementations

   Paul and Tony implemented an experimental prototype of NAT on public
   domain KA9Q TCP/IP software [1]. This implementation manipulates
   addresses and IP checksums.

   Kjeld implemented NAT in a Cray Communications IP-router. The
   implementation was tested with Telnet and FTP. This implementation
   manipulates addresses, IP checksums, TCP sequence/acknowledge numbers
   and FTP PORT commands.

   The prototypes has demonstrated that IP addresses can be translated
   transparently to hosts within the limitations described in this
   paper.

REFERENCES

   [1] Karn, P., "KA9Q", anonymous FTP from ucsd.edu
       (hamradio/packet/ka9q/docs).

   [2] Fuller, V., Li, T., and J. Yu, "Classless Inter-Domain Routing
       (CIDR) an Address Assignment and Aggregation Strategy", RFC 1519,
       BARRNet, cisco, Merit, OARnet, September 1993.

   [3] Rekhter, Y., Moskowitz, B., Karrenberg, D., and G. de Groot,
       "Address Allocation for Private Internets", RFC 1597, T.J. Watson
       Research Center, IBM Corp., Chrysler Corp., RIPE NCC, March 1994.

Security Considerations

   Security issues are not discussed in this memo.

Authors' Addresses

   Kjeld Borch Egevang
   Cray Communications
   Smedeholm 12-14
   DK-2730 Herlev
   Denmark

   Phone: +45 44 53 01 00
   EMail: kbe@craycom.dk

   Paul Francis
   NTT Software Lab
   3-9-11 Midori-cho Musashino-shi
   Tokyo 180 Japan

   Phone: +81-422-59-3843
   Fax +81-422-59-3765
   EMail: francis@cactus.ntt.jp