# Anatomy: A Look Inside Network Address Translators - The Internet Protocol Journal - Volume 7, Number 3



***by Geoff Huston, APNIC***

Over the past decade numerous IP-related technologies have generated some level of technical controversy. One of these is the *Network Address Translator*, or NAT. This article describes the inner workings of NATs in some detail, and then looks at the issues that have accompanied the deployment of NATs in the Internet that appear to have fueled this technical controversy. NATs are a very widespread feature of today's Internet, and this article attempts to provide some insight as to how they operate, why there is such a level of technical controversy about NATs, and perhaps some pointers to what we have learned about technology and the process of standardization of technology along the way.

### NAT Motivation

The first RFC document describing NATs was by Kjeld Egevang and Paul Francis in 1994 [1]. The original motivation behind the NAT work was based on efforts in the early 1990s associated with a successor protocol to IPv4. The overall effort of a successor protocol to IPv4 was to devise a protocol that would directly address the issues of accelerating address consumption in IPv4 that appeared to be leading to the prospect of imminent address exhaustion. Although IPv4 was capable of uniquely addressing some 4.4 billion devices, it was evident by as early as 1992 that the world was heading down a path of very intensive deployment of devices that included communications capabilities, and that IPv4 was not going to be able to extend across the full range of future device deployment. The objective with NAT was to define a mechanism that allowed IP addresses to be shared across numerous devices. In addition, it was intended that NATs could be deployed in a piecemeal fashion within the Internet, without causing changes to hosts or other routers. Other forms of address-sharing technologies relied on intermittent connectivity, whereas NATs were intended to allow a collection of connected devices to share an address pool dynamically. The original RFC portrays this approach as being a measure that can "provide temporarily relief while other, more complex and far-reaching solutions are worked out."

So, as documented, the original intent of NATs was to be a possible short-term response to address exhaustion while longer-term solutions were being devised. NATs were also intended to be unmanaged devices that are transparent to end-to-end protocol interaction, requiring no specific interaction between the end systems and the NAT device.

A decade later NATs are attaining a status of near-ubiquitous deployment across the Internet, and although IPv6 has been defined and deployment is commencing, NATs appear to be a very well-entrenched part of the network landscape. And, for the most part, NATs continue to function as unmanaged devices.

They can be transparent to some forms of protocol interaction, but, as the voice-over-IP folks are finding out, they can be very obvious to the point of being highly disruptive to other forms of protocol operation.

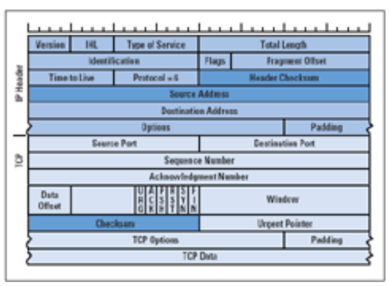
### NAT Operation

The operation of NATs is deceptively easy to describe in general terms. They are active units placed in the data path, usually as a functional component of a border router or site gateway. NATs intercept all IP packets, and may forward the packet onward with or without alteration to the contents of the packet, or may elect to discard the packet. The essential difference here from a conventional router or a firewall is the discretional ability of the NAT to alter the IP packet before forwarding it on. NATs are similar to firewalls, and different from routers, in that they are topologically sensitive. They have an "inside" and an "outside," and undertake different operations on intercepted packets depending on whether the packet is going from inside to outside, or in opposite direction.

NATs are IP header translators, and, in particular, NATs are IP *address translators* . The header of an IP packet contains the source and destination IP addresses. If the packet is being passed in the direction *from* the inside *to* the outside, a NAT rewrites the source address in the packet header to a different value, and alters the IP and TCP header checksums in the packet at the same time to reflect the change of the address field. When a packet is received *from* the outside destined *to* the inside, the destination address is rewritten to a different value, and again the IP and TCP header checksums are recalculated (Figure 1). The "inside" does not use globally unique addresses to number every device within the network served by the NAT. The inside (or "local") network may use addresses from private address blocks, implying that the uniqueness of the address holds only for the site. Let's look at this using an example.

Figure 1: TCP/IP Header Fields Altered by NATs (Outgoing Packet)

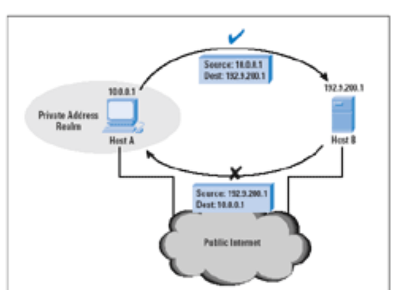
*Figure 1: TCP/IP Header Fields Altered by NATs (Outgoing Packet)*




As shown in Figure 2, how can local (private) host A initiate and maintain a TCP session with remote (public) host B? Host A first uses the *Domain Name System* (DNS) to find the public IP address for host B, and then creates an IP packet using host B's address as the destination address and host A's local address as the source, and passes the packet to the local network for delivery. If the packet was delivered to host B without any further alteration, then host B would be unable to respond. The public Internet does not (or should not at any rate!) carry private addresses, because they are not globally unique addresses.

Figure 2: Public/Private Communication

*Figure 2: Public/Private Communication*



With a NAT between hosts A and B, the NAT intercepts host A's outgoing packet and rewrites the source address with a public address. NATs are configured with a pool of public addresses, and when an "inside" host first sends an outbound packet, an address is drawn from this pool and mapped as a temporary alias to the inside host A's local address. This mapped address is used as the new source address for the outgoing packet, and a local session state is set up in the NAT unit for the mapping between the private and the public addresses.

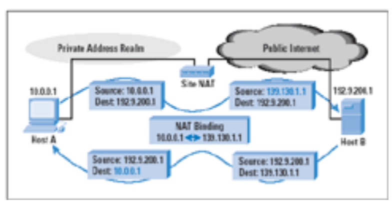
After this mapping is made, all subsequent packets within this application stream, from this internal address to the specified external address, will also have their source address mapped to the external address in the same fashion.

When an incoming packet arrives on the external interface, the destination address is checked. If it is one of the NAT pool addresses, the NAT box looks up its translation table. If it finds a corresponding table entry, the destination address is mapped to the local internal address, the packet checksums are recalculated, and the packet is forwarded. If there is no current mapping entry for the destination address, the packet is discarded.

The mode of operation of a NAT is shown in Figure 3. So, continuing our example, the local host at address A is directing packets to the external server host at address B. Because the NAT is in the path, the NAT has altered the packets so that address A is translated to address X. Host A is aware that it is communicating with host B, and from host A's perspective this is a normal session. Host B believes that it is communicating with a host at address X, and is entirely unaware of address A. From host B's perspective this is a normal session with a host at address X.

Figure 3: NAT Traversal

*Figure 3: NAT Traversal*



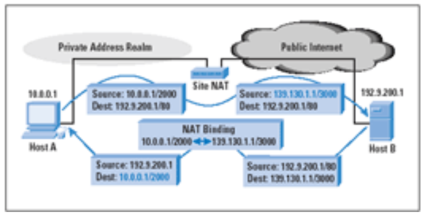

Dynamically created mapping entries (or "bindings") are typically maintained by the NAT with a *timer*. If no packets that use the mapping are received by the NAT within a certain time window, then the binding is removed from the NAT and the public address is returned to the NAT pool.

### NAPTs

A variant of the NAT is the *Port-Translating NAT*, or NAPT. This form of NAT is used in the context of TCP and *User Datagram Protocol* (UDP) sessions, where the NAT maps the local source address and source port number to a public source address and a public-side port number for outgoing packets. Incoming packets addressed to this public address and port pair are translated to the corresponding local address and port. Again, the binding is maintained by a NAT idle timer, and upon expiration of the timer the public address and port pair are returned to the NAT pool (Figure 4).

Figure 4: NAPT Traversal

Figure 4: NAPT Traversal




Again the NAPT is attempting to be transparent in terms of providing a consistent view of the session to each end, using a symmetric binding of a local address and port pair to an external address and port pair.

A reasonable question to ask is: Why should NAPTs bother with port translation? Are straight address translations not enough? Surprisingly, NATs can be relatively profligate with addresses. If each TCP session from the same local host is assigned a different and unique external pool address, then the peak address demands on the external address pool could readily match or exceed the number of local hosts, in which case the NAT could be consuming more public addresses than if there were no NAT at all! NAPTs allow concurrent outgoing sessions to be distinguished by the combination of the mapped address and mapped port value. In this way each unique external pool address may be used for up to 65,535 concurrent mapped sessions.

For a while the terminology distinction between NATs and NAPTs was considered important, but this has faded over time. For the remainder of this article we use current terminology, and look at NATs and NAPTs together and refer to them collectively as "NATs."

### NAT Behavior

The use of NATs involves two basic issues: One is that NATs make applications "brittle" in that NATs support a particular style of application operation, and if the application deviates in any way from this style then the application no longer works. The second is of much more concern, and that is that NATs differ from each other in quite fundamental ways. What works across one NAT may not work at all for another class of NAT. It has also been reported that NATs differ not only on a vendor-by-vendor basis, but even on a model-by-model basis within a single vendor's range of NAT units. The implication here is that such differences of behavior become a matter for discovery by applications rather than something applications can predict in advance. This section explores this behavioral aspects of NATs in further detail.

### Symmetry and Sessions

NATs can manage address mapping in numerous ways, and many implementations of NATs use a form of binding termed a "symmetric" binding.

A
*symmetric*
binding is where the mapping of a local address to a public address is exclusively tied to the destination address used in the initial trigger outgoing packet for the lifetime of the binding. Incoming external packets with the mapped public address as their destination are translated to the local address only if the source address of the incoming packet matches the destination address of the original mapping. Multiple sessions to different public hosts may use the same mapped public address, or may use different public addresses for each session. This mapping is "endpoint" sensitive. Symmetric NATs represent a restricted model of operation, where each NAT binding represents a window through the NAT that is visible only to the destination host (Figure 5).

By comparison, a
*full-cone*
NAT allows any external host to use this opened window, where all incoming packets addressed to the mapped external address are translated

to the mapped internal address and forwarded through the NAT. Symmetric NATs represent the most restrictive form of behavior, whereas full-cone NATs represent a far more permissive mode of operation.

In the context of NATs, this symmetric mode of operation refers to the session state 5-tuple, made up of Transport Protocol, the local IP address and port number, and the destination IP address and port number. When a session is opened from the local host to a remote service port on a remote host, then only that remote service can pass packets back through the NAT to the local host on that port. As with NATs, a full-cone NAT allows any remote service entity to direct packets back through the port window.

NATs can be further refined by having different behaviors for TCP and UDP transports. A NAT may behave in a symmetric manner for TCP sessions, and operate in a full-cone mode for UDP transactions. The variations in NAT behavior has led to an exercise in categorizing NAT behaviors and developing a discovery protocol whereby a pair of cooperating systems can discover if one or more NATs is on the network path between them, as well as attempting to establish the type of NAT.

### Discovering NAT Behaviors and STUN

NAT behavior has not been the topic of any industry standardization efforts, and it should not be surprising to learn that, given that a range of possible NAT behaviors exist under certain conditions, the market contains NAT offerings that cover the full spectrum of possibilities. In the absence of common specifications or standards, implementers have been placed in the position of having to make some creative guesses as to what the "right" behavior should be under such circumstances. This is a significant problem for the application designer, given the prospect that in today's Internet any popular application must have a means of being able to function correctly in the face of one or more NATs on the path between two hosts that are communicating using the application.

One of the more pressing problems here is that NATs commonly enforce an application model where the local "hidden" host must initiate a transaction in order to create a window in the NAT to allow the packets of the remote host back into the local network.

Some applications may wish to undertake "referral," where the correspondent host on the external side may want to pass the externally presented address and port details of the local host to a third party in order to commence a further part of the transaction. Other application transactions may simply want to be initiated from the external side. Although this may have been thought of as a relatively obscure condition, it was brought into the forefront of attention when various forms of voice-over-IP and peer-to-peer applications gained popularity. In particular, the question of "how can the external side initiate a packet flow in the presence of a NAT?" has become increasingly important.

Given that the application needs to perform some additional gymnastics in such a case, there is the additional question that the application must answer, namely: "How does the application learn that there are NATs in the path in the first place?"

At this point the application is placed in the role of performing a forensic exercise of establishing whether or not its packets are being altered by one of more NATs when it attempts to establish an end-to-end packet transaction. If so, what types of implementation decisions have been made by the NAT in terms of the way in which packets are being systematically modified? In others words, what is the anatomy of the particular NATs that have been discovered along the path? This anatomy exercise is further complicated by the observation that NATs are silent devices, so the application cannot directly interrogate the NAT to establish its behavior. All that is left is a somewhat unsatisfying guessing game for the application. It is forced to send particular types of test packets through the NAT to some pre-defined counterpart on the other side. The application must then compare the self-view of the IP address and port number of the local host to the remote view of its IP address and port number, and then attempt to guess the nature of the systematic transforms that the NAT is applying.

In the case of TCP it appears that the prevalent NAT behavior is that of a symmetric NAT based on address and port bindings. This implies that when the local host opens up a TCP session with a remote host, the NAT address and port bindings for the local host are coupled with the address and port of the destination host. Only packets with a source field of the destination host can pass packets back through the NAT to the TCP session of the local host. In other words, when a TCP session has been established within a NAT, only the two endpoints of the TCP session can access the NAT bindings, and attempts by others to direct packets to the external-side presented address and port meet with the NAT discard response. The fine-grained behavior of NATs with respect to TCP sessions can vary according to the amount of TCP state maintained by the NAT. At a basic level, the NAT can maintain a binding based on the local address and port and the remote address and port. The NAT also can keep the binding timer at a high value until a
`FIN`
exchange is observed, or until the session is reset through the
`RST`
flag being set, at which point the binding timer can be reduced to a very short interval. The NAT can also track the sequence number windows of the two sides and associated window sequence number scaling values and not adjust the binding timer of the session for TCP packets with sequence numbers outside the sequence number window with their
`FIN`
or
`RST`
flags set.

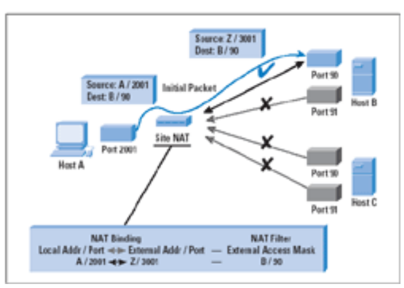
These NAT behaviors are based on the explicit signaling of changes in session state within the TCP packet exchange, and the consequent ability of the NAT to track the session state and adjust the associated binding timer in response to this state information. UDP is not so straightforward, because there is no explicit session state within a UDP packet exchange, and various NATs behave differently with respect to UDP-based bindings.

Various classes of NAT behavior relate to how UDP bindings are managed within a NAT. These have been classified into four types of behaviors [11]:

- *Symmetric:* We have already encountered the symmetric NAT, where the NAT mapping refers specifically to the connection between the local host address and port number and the destination address and port number and a binding of the local address and port to a *publicside* address and port. Any attempts to change any one of these fields requires a different NAT binding. This is the most restrictive form of NAT behavior under UDP, and it has been observed that this form of NAT behavior is becoming quite rare, because it prevents the operation of all forms of applications that undertake referral and handover.

  Figure 5: Symetric NAT

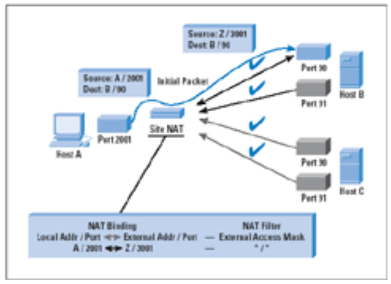
  *Figure 5: Symmetric NAT*




- *Full-cone:* A full-cone NAT is the least restrictive form of NAT behavior, where the binding of a local address and port to a public-side address and port, when established, can be used by any remote host on any remote port address. (Refer to Figure 6.)

  Figure 6: Full Cone NAT

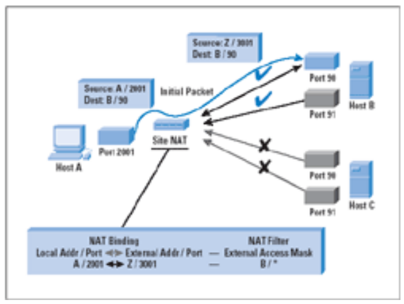
Figure 6: Full Cone NAT




*Restricted-cone:* A restricted-cone NAT is one where the NAT binding is accessible only by the destination host, although in this case the destination host can send packets from any port address after the binding is created. (Refer to Figure 7.)

Figure 7: Restricted-Cone NAT

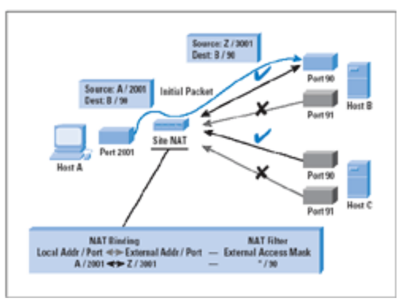
Figure 7: Restricted-Cone NAT




*Port-restricted-cone:* A port-restricted-cone NAT is one where the NAT binding is accessible by any remote host, although in this case the remote host must use the same source port address as the original port address that triggered the NAT binding. (Refer to Figure 8.)

Figure 8: Port-Restricted-Cone NAT

Figure 8: Port-Restricted-Cone NAT




So can an application tell if one or more NATs are in the path, and, if so, what form of behavior the NAT is using? For this purpose the *Simple Traversal of UDP through NATs* (STUN) protocol has been developed [11]. STUN is a probe system that examines the interchange between a STUN client that may lie behind a NAT and a STUN server that is positioned on the public side of the NAT. The STUN-server host must be configured with two IP addresses, and the STUN itself should respond to queries on two UDP port numbers. The protocol is a simple UDP request-response protocol that uses embedded addresses in the data payload, and compares these addresses with header values in order to determine the type of NAT that may lie in the path between client and server.

The basic operation of STUN is a request-response protocol, using a common request of the form: "Please tell me what public address and port values were used to send this query to you."

STUN can be used to discover if a NAT is on the path between a client and server, and attempt to discover the type of NAT by a structured sequence of requests and responses. The client sends an initial request to the STUN server. If the public address and port in the returned response are the same as the local address, then the client can conclude that there is no NAT in the path between the client and the server. If the values differ, the client can conclude that there is a NAT on the path. STUN then uses subsequent requests to determine the type of NAT. One critical additional item of information returned by the STUN server in the initial response is an alternate IP address and port number that can also reach the same STUN server.

The second STUN request is directed to the same address and port as the initial request, but this time the request includes a control flag that requests the STUN server to respond using its alternate source address and port values. If the STUN client receives this alternate-sourced response, then it can conclude that it is behind a full-cone NAT. This is because the initial NAT binding of the local host address to the external presentation address can evidently be accessed by third-party external hosts.

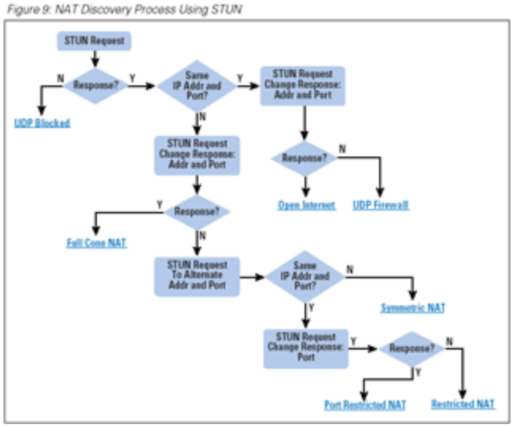
If no response is received to the second request, then the STUN client sends the original probe request, but this time the request is addressed to the alternate destination address and port pair for the STUN client. If the returned address and port values relating to the new NAT binding are different from those of the first request, then the client can conclude that it is behind a symmetric NAT.

If the values are unaltered, then a further request can be made to determine the form of restricted-cone behavior. This fourth request includes a control flag to direct the STUN server to respond using the same IP address, but with the alternate port value. A received response indicates the presence of a port-restricted cone, and the lack of a response indicates the presence of a restricted cone.

Periodic exchanges between the STUN client and server can also discover the timer used by the NAT to maintain address bindings. Additional components of STUN are intended to provide some reasonable level of integrity in the packet exchange. A flowchart of a STUN-based NAT discovery process is shown in Figure 9.

Figure 9: NAT Discovery Process Using STUN

*Figure 9: NAT Discovery Process Using STUN*




### Further Behaviors: Hairpins and Determinism

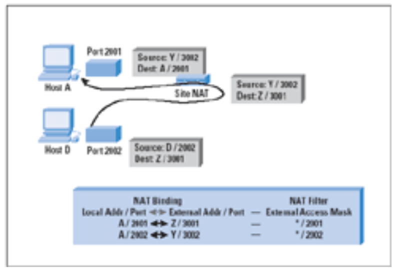
It would be good if NAT behavior remained that simple. However, it does not, and some further tests on NATs reveal further differences in various NAT implementations [16].

The first area of difference is whether the NAT supports the so-called *hairpin* operation, where a local host directs a packet to the public address and port of an already mapped local host, or even to its own mapped address and port. If successful, then the NAT supports hairpin operation, where the NAT bindings, when created, are available to either side of the NAT. (Refer to Figure 10.)

Furthermore, the NAT may generate a binding for this operation—or not—thereby presenting the hairpin packet with an external address and port, indicating that an outbound binding has been performed in conjunction with the inbound binding, or with an internal address and port, indicating that only an inbound binding is being performed.

Figure 10: Hairpin NAT Operation

*Figure 10: Harpin NAT Operation*




The second is in the general class of NAT determinism. Nondeterministic NATs change their binding behavior when a binding conflict of some sort occurs in the NAT. This is further based on the classification of whether "primary," "secondary," or even "tertiary" NAT behaviors differ. To explain primary, secondary, and tertiary behaviors, it is first noted that some NATs attempt to preserve the port address in the binding, so that the local source port and the externally bound port are the same whenever possible. This is the "primary" binding of the NAT. If another local host obtains a NAT binding using the same source port number, then the behavior of the NAT for this conflicting port binding may differ from that where the port number is preserved. The first conflict of port allocations in bindings is the "secondary" binding. In some cases the primary behavior is that of a full cone, or a restricted cone, while the NAT behaves in a symmetric fashion for the secondary instance where the port number has been mapped to a new value by the NAT.

A tertiary behavior occurs when a third binding is added to the NAT, because, again, the behavior of the NAT may be different for this binding.

It is also possible that the NAT may elect to preserve the binding in any case, and remove the current binding and replace it with a new binding that refers to the most recent packet that the NAT has processed.

All these behaviors can be classified as
*nondeterministic*
, in that the NAT behavior becomes one that is determined by the order of outbound traffic. The implication is that repetitions of the same STUN test at different times may produce different classifications of the type of NAT. The inference is that if an application uses STUN to determine the type of NAT in the path, and then selects a certain behavior based on this STUN-derived knowledge of the NAT type, nondeterministic NATs may behave differently between the STUN test and the application. The NAT response for a particular binding cannot be predicted in advance, and even when a binding state is established it may be disrupted or altered by subsequent traffic.

### Another Approach to Classifying NATs

Further tests on NATs reveal that the various behaviors are yet more complex, and that different sequences of tests across a NAT will lead the test routine to come to different conclusions as to the type of NAT [13]. The key observation here is that NATs are the conjunction of two distinct behavior sets:

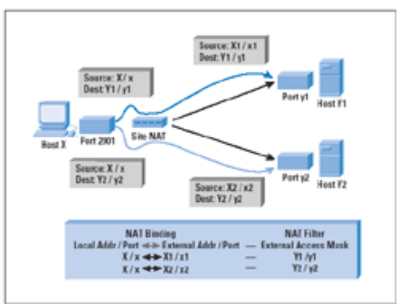
- *Binding*, or context-based packet translation: Detecting those packets that can be associated with a current binding and using that binding in a manner according to the logical direction of the packet to perform packet header transforms
- *Filtering*, or packet discard: Discarding those packets that cannot be associated with current bindings and discarding them

If a STUN-like test sequence was for a local host to send a packet to one destination and obtain a response of what NAT binding was used, and then to send a packet to a second destination and compare the results, the observation of the NAT using a different binding for each request may lead the tester to conclude that the NAT is a fully symmetric NAT. If the test sequence is for the NAT to send one packet to a destination and have the destination respond using a different source address, then the observation that the response packet is successfully delivered through the NAT back to the originating local host may lead the tester to the conclusion that the same tested NAT is some form of cone NAT.

The STUN approach classifies NAT behaviors on the basis of a single binding being established by the local host when contacting an external host, and then considers what constraints are placed on third-party external hosts as they attempt to access this initial binding. An adjunct to this approach is based on the local host establishing two bindings to two distinct external hosts, and looking for any relationship between these two bindings. (See Figure 11).

Figure 11: Outboung Connections from a Common Source

*Figure 11: Outbound Connections from a Common Source*




The behaviors of NATs under this condition can be classified under numerous behavioral aspects.

### Binding

Binding behavior can be seen as the amalgam of three somewhat distinct design decisions, namely the manner in which a binding is generated, the behavior of the NAT in managing external ports used in bindings, and the manner in which expiration timers that govern the continued existence of the binding are refreshed.

### NAT Binding Behavior:

- *Endpoint independent:* The NAT reuses the port binding for subsequent sessions initiated from the same internal IP address and port to any external IP address and port. This is analogous to a full-cone NAT.
- *Endpoint address dependent*: The NAT reuses the port binding for subsequent sessions initiated from the same internal IP address and port only for sessions to the same external IP address, regardless of the external port. This is a looser form of symmetric NAT, where the binding is created on the basis of the external address, rather than the external address and port.
- *Endpoint address and port dependent*: The NAT reuses the port binding for subsequent sessions initiated from the same internal IP address and port only for sessions to the same external IP address

and port. This is a more precise form of UDP symmetry where the binding is available only to a single session, where a session is the 5-tuple of protocol, source address, source port, destination address, and destination port.

### Port Binding Behavior:

- *Port preservation:* In addition to the differences in the binding between the two cases, the NAT may attempt to preserve the local port number, if possible. The terminology proposed here is port preservation to describe this NAT action.
- *Port overloading:* Some NATs attempt to undertake port preservation at all times, so that when a different local host establishes a binding using a port that is already being preserved, the new binding will usurp the existing binding. This behavior is proposed to be termed port overloading.
- *Port multiplexing:* The alternative to port overloading is use of the external entity to perform the demultiplexing of the port. In this case if two local systems use the same source port to send packets to two different external hosts, the NAT preserves the source port in the two bindings. If the NAT is using a single external address, the external view is two packets with the same source address and source port, sent to two different external addresses. The reverse packets have the same destination address and port, and the NAT determines the appropriate binding based on the source address and port in the reserve packets. This requires an endpoint address and port-dependant binding behavior. If two internal hosts are directing packets to the same external endpoint using the same source port addresses, then it is necessary for one of the sessions to use a binding with an altered port number. This could be considered as nondeterministic behavior.

### Binding Timer Refresh:

- *Bidirectional:* The NAT does not keep the binding active indefinitely, and normally removes the binding if there are no further packets that use the binding within a certain time period. However, there are variations in the classification of packets that the NAT considers as packets that reset the timer. In the case of bidirectional binding timer refresh, packets from either the local hosts or an external host that uses the NAT binding cause the NAT binding expiration time to be reset.
- *Outbound:* An outbound binding timer refresh NAT resets the expiration timer only when packets pass from the local host to the external host within the context of the binding. The implication is that a local host may have to use some form of keepalive operation to maintain a NAT binding in the face of an inbound UDP unidirectional traffic flow. Additionally, the expiration timer may be on a *persession* basis, or may be on a per-binding basis if multiple sessions are associated to a single binding in the NAT.
- *Inbound:* As the name suggests, this is the opposite of the previous case, where only inbound packets cause the expiration timer of the binding to be refreshed.
- *Transport Protocol state:* Although these forms are useful in the case of UDP-based sessions, when the binding is based on a transport session (such as TCP), the NAT can base its binding timer refresh on the transport session state. For TCP this would infer a binding refresh time that is refreshed by any session packet in either direction (bidirectional), with the exception of packets with the TCP `RST` or `FIN` flags set. Although it would be an option to drop the NAT binding state when such packets are seen, this makes the NAT vulnerable to denial-of-service attacks by third-party injection of TCP `RST` packets, so there is some merit in using the binding timer for TCP sessions.

### Filtering

The second phase of the test has two external hosts directing a probe to the same binding address, and classifying the behaviors based on what packets are filtered and discarded by the NAT (Figure 12).

### External Filtering:

- *Endpoint independent:* The NAT does not filter and discard packets that are addressed to the external part of the binding, irrespective of the source values in the packet. This is analogous to a full-cone NAT.

- *Endpoint address dependent*: The NAT filters and discards packets that are addressed to the external part of the binding, unless the source address of the packet matches the destination address used in the binding. This is analogous to a restricted-cone NAT.
- *Endpoint address and port dependent*: The NAT filters and discards packets that are addressed to the external part of the binding, unless the source address and port number of the packet matches the destination address used in the binding. This is analogous to a port-restricted-cone NAT or a symmetric NAT.

**External Filtering Timer Refresh:**

As with binding timers, these timers can be refreshed bidirectionally, inbound or outbound.

### NAT Behaviors

The approach of carefully identifying the areas where NAT behaviors differ and classifying these behavioral differences in a methodical manner is one that has the potential to at least allow us to use the same sets of words when we talk about NAT behaviors, and hopefully also refer to the same set of actual behaviors when we use the same descriptions. The original approach with the STUN work used the terms *symmetric*, *full-cone*, and forms of *restricted-cone* to describe variations of NAT behaviors. Experience with this form of classification has exposed further variations in NAT behaviors, and this has led to a form of NAT classification that first uses a delineation of binding and filtering behaviors, and then classifies the various ways in which these bindings and filters are maintained within the NAT. Additional classification attributes include whether the NAT supports hairpin connections or not and whether it operates in a deterministic or nondeterministic manner.

This exercise is not another study in comparative taxonomies. A NAT has no standard way in which to advertise its presence, nor does it have any standard way in which to advise protocols or applications of the particular behaviors it applies to packets being passed through the NAT. In the absence of such explicit advertisements of the presence of a NAT, it is left to the application to make the necessary adjustments that allow it to function in the presence of NATs. The aim of behavioral classification is to associate test sequences that expose the presence of a NAT, and to determine its behavior. This allows applications to invoke a test procedure that exposes a particular choice of behaviors of a NAT implementation, and then allows the application to invoke a mode of operation that can operate across the particular NAT.

The choices available to application environments include the use of *agents* as session initiation intermediaries, where the endpoints make initial contact through agents, who then assist in passing binding information to the endpoints, allowing them to directly communicate. Other forms of application behavior need to be invoked when the NAT is endpoint address and port dependant for both binding and filtering. Different application responses are applicable when one endpoint is behind a NAT and when both endpoints are behind NATs. A typical application response in this latter case where both endpoints are behind highly restrictive NATs is for the endpoints to use agents as session intermediaries, so that the application payload is then passed through the intermediaries because an end-to-end pair of NAT bindings cannot be established.

### Living in a NAT World

It would be a reasonable conclusion to draw from the previous sections that we are left in the somewhat unsatisfying position of observing that there is near-universal deployment in today's Internet of NAT devices that do not conform to any particular well-defined behavior set. NAT behavior varies across implementations, and NATs have no ability to disclose their particular behaviors to applications that are attempting to compensate for their presence in the path. It is extremely challenging for applications to reliably predict the behavior of the NATs that lie in the path, and more so in the face of multiparty applications, such as interactive game environments, where the application is attempting to understand the level to which this silent intermediary is capable of supporting a relatively promiscuous NAT binding state in terms of external entities that wish to send packets to the local host, and communicate between themselves about the local host as a single entity.

### NATs, Client-Server, Peer-to-Peer, and Multiparty Applications

NATs, as a class of devices, have strong associations with a client-server model of communications. As long as all the servers have a consistent external visibility, with stable addresses in terms of an IP address and port number, and as long as clients initiate connections with servers in a fixed two-party communications model using TCP as a transport protocol, and refrain from turning on
*IP Security*
(IPSec), then NATs generally behave in a relatively stable and unobtrusive manner. Applications that operate conservatively in this limited mode can be unaware of the presence of NATs in their path. The relatively widespread deployment of NATs and the continued use of client-server-based applications on the Internet attests to the capability of the NAT to perform transparently and effectively within the strict confines of this particular mode of communication.

However, peer-to-peer applications are more problematic for NATs, because they have extended the model of a NAT beyond its original realm of capability. If the desire is to continue to support the NAT dynamic binding, but also allow external parties to initiate a communication to a local host, then the NAT ceases to be transparent and unobtrusive, and in this extended environment the NAT transforms itself into an application-visible network element. It is overly presumptuous to claim that NATs have led to the increasing deployment of multiparty applications on the Internet, but certainly multiparty applications have been seen to be useful in circumventing some of the more aggravating shortcomings of NATs in various peer-to-peer realms.

In this latter context, the local party is forced to advertise its willingness to participate in a peer-to-peer realm by communicating with an external agent. The local agent performs a NAT discovery test, and then selects a mode of operation that is consistent with the discovered behaviors of a NAT that may be on the path between the client and the agent. The agent then advertises itself as the local party's intermediary to other peers within the application realm. Attempts to initiate a connection with the local party are directed to the external agent, who then undertakes to perform a rendezvous function in order to establish a session.

Depending on the NATs that may exist between the two parties, the rendezvous function may need to perform a convoluted handshake process, or, in some instances, may not be able to set up a peer-to-peer session at all. This topic of establishing connectivity in the face of NATs in the path is sufficiently complex to warrant a separate examination, and the various techniques and approaches are not examined in this article other than providing some suggestions for further reading.

The salient general observation is that NATs have fueled a new generation of applications that use intermediaries and rendezvous protocols. This shift in application behavior has implied greater attention to security frameworks for applications, because intermediaries represent an additional active element in the trust model. This, in turn, has implied that the application level has to turn to other chains of derivation of trust, because the basic Internet model of some form of persistent identity as being an attribute of an IP address is no longer a workable proposition in the face of NATs. The position we are reaching here is that identity and trust need to be derived from other attributes of the end host and the application that it has invoked.

### ICMP

If an

*Internet Control Message Protocol*
(ICMP) message is passed through NAT, there is not only the outer IP header to consider, but also the ICMP payload. Most ICMP messages contain part of the original IP packet in the body of the message, so for the NAT to behave as transparently as possible, the IP address of the IP header contained in the data part of the ICMP packet should be modified according to the NAT binding state, as well as the IP header Checksum field of this inner packet header.

### NATs and IP Fragmentation

NATs that use bindings that include both address and port values do not have a clear and uniform response to fragments of an IP packet. The TCP or UDP header is resident only in the initial IP fragment, and subsequent IP packet fragments do not contain a copy of the transport layer packet header.

Some NATs attempt packet reassembly as if they were the end host, and they perform the NAT translation only when the original IP packet has been reassembled. Of course the reassembled packet may be too large to be forwarded onward, and the NAT may be forced to further fragment the packet. The interplay between this behavior and various forms of path
*Maximum Transmission Unit*
(MTU) discovery become a source of frustration.

Other NAT packet fragmentation behaviors do not attempt packet reassembly, but rely on a stored packet fragment translation state that directs the translation to be performed on subsequent packet fragments after the initial packet header translation has been performed on the initial IP packet fragment.

This form of behavior has weaknesses in terms of out-of-order fragments, when following fragments are received by the NAT prior to the initial IP packet fragment, and in such cases the NAT often has little choice but to silently discard the out-of-order fragment as untranslatable.

### NATs and Application Level Gateways

This brings up one of the more vexing questions regarding NAT behavior, namely, should the NAT include knowledge of the payload of certain applications? Numerous applications, including FTP and the DNS resolution protocol, include IP addresses within the payload of the application. In an effort to achieve complete transparency of operation, some NATs have included
*Application Level Gateway*
(ALG) functionality for certain applications so that this use of IP addresses in the payload can be detected and altered according to the current NAT translation bindings.

The case of ICMP represents one of the simpler forms of gateway functionality, because it can be performed in the same manner as the basic NAT transform, on a per-packet basis while attempting to maintain retained session state. Payload transformations in the case of a TCP-based application have implications in terms of requiring subsequent alteration of TCP sequence numbers, length fields, and even the repacketization of the payload data stream, given that the data transform required by the address change may imply a change of payload length.

Some units attempt to combine the functionality of a NAT with that of an ALG, such that the NAT is an active intermediary in the transport session. This allows the NAT/ALG to perform "deep" inspection of the packets, and use both application protocol knowledge and per-application-session retained state in order to apply the NAT binding transforms to the application payload as well as to the outer IP packet header.

The most widely deployed application that can use IP addresses in the payload is FTP, where IP addresses are passed in the payload of the control channel in order to allow data sessions to be initiated on distinct transport sessions. The variability and reliability of FTP ALG support in

NATs has led to the widespread use of the passive mode of FTP operation, where the data flow is passed within the control session.

A related question is that of the use of IPSec and NATs. IPSec with *Authenticated Header* protection attempts to protect what it believes is the fixed part of the IP packet header, including the source and destination addresses. The NAT changes to the IP packet invalidate the Authentication Header integrity check. Also the NAT changes the IP and UDP or TCP checksums, and this disrupts the *Encapsulating Security Payload* (ESP) function of IPSec. The implication is that IPSec needs to operate upon a TCP or UDP payload, as in the IPSec operating tunnel model, or IPSec carried as a payload within other types of tunnel operation.

It is also the case that NATs today are heavily enmeshed with the UDP and TCP transport protocols. Other transport protocols exist, including the *Streams Control Transport Protocol* (SCTP) and the *Datagram Congestion Control Protocol* (DCCP), and doubtless more transport protocol offerings will follow over time. In each case it is a matter of individual choice how NAT implementations define NAT responses to such additional transport protocols. Although it is tempting to propose that NATs should fall back to an address-only form of binding that was not address-and-port based, this does not appear to be practical guidance. Another aspect of today's NAT deployment is that the most common scenario appears to be that of a single external address and mapping each locally initiated session into a binding that uses this common external IP address and a variable external port number. This means that NATs need to be able to identify and transform port addresses from the Transport Protocol section of the IP header.

Another salient factor here is the common association of NATs and firewalls into a single unit, and the coupling of address utilization compression properties of the NAT with its associated packet-filtering actions. Deploying a NAT at the external interface of a site does lead to more restrictive site filtering outcomes and a more restrictive model of application interaction, where the model attempts to impose the constraint that applications are initiated from within the site, and that unknown or unidentifiable external traffic is considered hostile and should be subject to firewall-based inspection and filtering. From this perspective there is little desire to make more permissive NATs as an isolated exercise, and there is instead a codependence between NAT behaviors and popularly used applications. Applications that work across today's NATs appear to enjoy popular uptake, and applications that enjoy popular uptake appear to determine what forms of traffic pass across NATs.

Popular or not, there are a class of applications that simply cannot work in a "native mode" across NATs, nor can ALGs assist here. These are applications that attempt to impose some level of end-to-end protection on the IP header fields, or use the IP address of the endpoint in a context of some form of persistant identity token. When the NAT alters the IP address, an application that uses strong forms of header validation rejects such packets as corrupted. Within this class of applications and tools, one of the more commonly referenced tools is that of IPSec with Authentication Header. There is a certain sense of irony in the observation that NATs are often seen as part of an overall approach to site security, yet cannot support a "native mode" operation of some of the basic tools that applications could use to support secure end-to-end data transfer.

### Views on NATs

It is certainly the case that NATs are very common in todays Internet, and it is worth understanding why NATs have enjoyed such widespread deployment while other technologies appear to be meeting some considerable resistance to widespread deployment. As the original NAT document points out:

> "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."
>
> —*Egevang and Fancis, "Network Address Translator,"* *RFC 1631*

More generally, the positive attributes of NATs include the following considerations:

- End hosts and local routers do not change. Whether there is a NAT in place between the local network and the Internet or not, local devices can use the same software and support the same applications. NATs do not require customized versions of operating systems or router images.
- As long as you accept the limitation that sessions must be initiated from the "inside," NATs can work in an entirely transparent fashion for a set of client-server classes of applications.
- If you accept the perspective that services and usage scenarios that are not supported by NATs are "unwelcome" or "unsafe," then NATs can be placed into a role as a component of a site's security architecture, providing protection from attacks launched from the outside toward the inside network.
- NAT conserves its use of public address space.
- NAT allows previously disconnected privately addressed networks to connect to the global Internet without any form of renumbering or host changes—and renumbering networks can be a very time-consuming, disruptive, and expensive operation, or, in other words, renumbering is difficult.
- NAT address space is an effective, provider-independent addressing solution with multihoming capabilities. NAT allows for rapid switching to a different upstream provider, by renumbering the NAT address pool to the new provider's address space. In essence, NATs provide the local network manager with the flexibility of using provider-independent space without having to meet certain size and use requirements that would normally be required for an allocation of public, provider-independent address space.
- NAT allows the network administrator to exercise some control over the form of network transactions that can occur between local hosts and the public network.
- NATs require no local device or application changes. This is perhaps one of the major "features" of NATs, in that the local network requires no changes in configuration to operate behind a NAT.
- NATs do not require a coordinated deployment. There is no transition, and no "flag day" across the Internet. Each local network manager can make an independent decision whether or not to use a NAT. This allows for incremental deployment without mutual dependencies.
- These days the common theme of the public address assignment policy stresses conservative use of address space with minimum waste. The standard benchmark is to be able to show that a target of 80 percent of assigned address space is assigned to a number of connected devices. Achieving such a very high usage rate is a challenging task in many network scenarios, and NATs represent an alternative approach where the local network can be configured using private addresses without reference to the use of public addresses.
- NATs are very widely available and bundled into a large variety of gateway and firewall units. In many units NATs are not an optional extra—they are configured in as a basic item of product functionality.

The market has taken NATs and embraced them wholeheartedly. And in a market-oriented business environment, what is wrong with that?

Unfortunately NATs represent a set of design compromises, and no delving into the world of NATs would be complete without exploring some of their shortcomings. So, after enumerating what are commonly seen as their benefits, it is now necessary to enumerate some of the broken aspects of the world of NATs.

"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."

*—Egevang and Francis,*
*"Network Address Translator,"* [RFC 1631]

"An opposing view of NAT is that of a malicious technology, a weed which is destined to choke out continued Internet development. While recognizing there are perceived address shortages, the opponents of NAT view it as operationally inadequate at best, bordering on a sham as an Internet access solution. Reality lies somewhere in between these extreme viewpoints."

*—Tony Hain,*
*"Architectural Implications of NAT,"* [RFC 2993]

- First, NATs cannot support applications where the initiator lies on the "outside." The external device has no idea of the address of the local internal device, and, therefore, cannot direct any packets to that device in order to initiate a session. This implies that peer-to-peer services, such as voice, cannot work unaltered in a NAT environment.
- The workaround to this form of shortcoming is to force an altered deployment architecture, where service platforms used by external entities are placed "beside" the NAT, allowing command and control from the interior of the local network, and having a permanent (non-NAT) interface to the external network. Obviously this implies some further centralization of IT services within the NATed site.
- Even this approach does not work well for applications such as voiceover-IP, where the "server" now needs to operate as some form of proxy agent. The generic approach here for applications to traverse NATs in the "wrong" direction is for the inside device to forge a UDP connection to the outside agent, and for the inside device to then establish what NAT translated address has been used, and the nature of the NAT in the path, and then republish this address as the local entity's published service rendezvous point. Sounds fragile? Unfortunately, it is. The other approach is to shift the application to use a set of endpoint identifiers that are distinct from IP addresses, and use a distributed set of "agents" and "helpers" to dynamically translate the application level identifiers into transport IP addresses as required. This tends to create added complexity in application deployment, and also embarks on a path of interdependency that is less than desirable. In summary, workarounds to reestablish a peer-topeer networking model with NATs tend to be limited, complex, and often fragile.
- The behavior of NATs varies dramatically from one implementation to another. Consequently, it is very difficult for applications to predict or expose the precise behavior of one or more NATs that may exist on the application data path.
- Robust security in IP environments typically operates on an end-toend model, where both ends include additional information in the packet that can detect attempts to alter the packet in various ways. In IPSec the header part of the packet is protected by the Authentication Header, where an encrypted signature of certain packet header fields is included in the IPSec packet. If the packet header is changed in transit in unexpected ways, the signature check will fail. Obviously IPSec attempts to protect the packet address fields—the very same fields that NATs alter! This leads to the observation that robust security measures and NATs do not mix very well. NATs inhibit implementation of security at the IP level.
- NATs have no inherent failover. NATs are an active in-band mechanism that cannot fail into a safe operating fallback mode. When a NAT goes offline, all traffic through the NAT stops. NATs create a single point where fates are shared in the NAT device maintaining connection state and dynamic mapping information.
- NATs sit on the data path and attempt to process every packet. Obviously bandwidth scaling requires NAT scaling.
- NATs are not backed up by industry-standardized behavior. Although certain NAT-traversal applications make assumptions about the way NATs behave, it is not the case that all NATs necessarily behave in precisely the same way. Applications that work in one context may not necessarily operate in others.
- Multiple NATs can get very confusing with "inside" and "outside" concepts when NATs are configured in arbitrary ways. NATs are best deployed in a strict deployment model of an "inside" being a stub

private network and an "outside" of the public Internet. Forms of multiple interconnects, potential loops, and other forms of network transit with intervening NATs lead to very strange failure modes that are at best highly frustrating.

- With NATs there is no clear, coherent, and stable concept of network identity. From the outside these NAT-filtered interior devices are visible only as transient entities.
- Policy-based mechanisms that are based on network identity (for example, *Policy Quality of Service* [QoS]) cannot work through NATs.
- Normal forms of IP mobility are broken when any element behind the NAT attempts to roam beyond its local private domain. Solutions are possible, generally involving specific NAT-related alterations to the behavior of the Home Agent and the mobile device.
- Applications that work with identified devices, or that actually identify devices (such as the *Simple Network Management Protocol* [SNMP] and DNS) require very careful configuration when operating an a NAT environment.
- NATs may drop IP packet fragments in either direction: without complete TCP/UDP headers, the NAT may not have sufficient stored state to undertake the correct header translation.
- NATs often contain ALGs that attempt to be context-sensitive, depending on the source or destination port number. The behavior of the ALGs can be difficult to anticipate, and these behaviors have not always been documented.
- Most NAT implementations with ALGs that attempt to translate TCP application protocols do not perform their functions correctly when the substrings they must translate span across multiple TCP segments; some of them are also known to fail on flows that use TCP option headers, for example timestamps.

From this perspective, NATs are a short-term expediency that is currently turning into a longer-term set of overriding constraints placed on the further evolution of the Internet. Not only do new applications need to include considerations of NAT traversal, but we appear to be entering into a situation where if an application cannot work across NATs, then the application itself fails to gain acceptance. We seem to be locking into a world that is almost the antithesis of the Internet concept. In this NAT-based world, servers reside within the network and are operated as part of the service provider's role, whereas end devices are seen as "dumb" clients, who can establish connections to servers but cannot establish connections between each other. The widespread use of NATs appears to be reinforcing a reemergence of the model of "smart network, dumb clients," whereas others would argue that the network is getting no smarter, it is just that the number of obstacles and amount of network debris is increasing while clients are getting worse at maintaining coherent end-to-end state in the face of such changes.

However, despite their shortcomings, despite the problems NATs create for numerous applications and their users, and despite the continued grappling over a common language to understand how NATs behave, numerous NATs are deployed, and, at least in the IPv4 realm, NATs appear to be a firmly fixed part of the future of the Internet. NATs continue to proliferate in today's Internet.

### Moving on with NATs

One commonly held belief is that deployment of IPv6 will eliminate the problem of NATs within the Internet. Certainly it is reasonable to observe that if achieving high address utilization densities is no longer the objective, then there will be plentiful public IPv6 address space and that particular reason to deploy NATs is significantly discounted in an IPv6 realm.

That does not say that IPv6 NATs will not be implemented, nor used. Indeed IPv6 NATs are already available, and they are being used, albeit to some small extent. NATs are, rightly or wrongly, considered to be part of a security solution for a site because of their filtering properties that prevent incoming packets from entering the site unless the NAT already has a permitting binding initiated from the inside. In addition, NATs allow a site to use an internally persistent naming and addressing scheme based on some form of deployment of IPv6 unique

*site local*
address, and deploy NATs at the edge to create an external view of the site that fits within a provider-based address aggregated view of the IPv6 Internet.

So it would perhaps be too enthusiastic a level of conjecture to suppose that IPv6 will drive away all forms of NAT use in IPv6. It is reasonable to predict that some use of NAT will be seen in IPv6, although many would be highly disappointed if the level of IPv6 NAT use rose to anywhere approaching that of NAT in IPv4.

However, the Internet is still largely a network that uses IPv4 and NATs, and efforts continue along the lines of reducing the amount of friction and frustration in a world in which NATs are prolific. One of the ways to progress here is to treat NAT boxes as yet another instance of Internet middleware, and attempt to apply the same sets of processes to NATs that appear in other instances of middleware. The work of the IETF in the
*Middlebox Communication Working Group*
uses a model that attempts to expose NATs, as well as firewalls, performance-enhancing proxies, application proxies, and relay agents, to the application, and allows the application to specify the policy that the middlebox should apply. In the case of NATs, this could allow an application to communicate to a NAT that it does not require any form of third-party access, and that a fully symmetric behavior could be applied to the binding without any loss in application functionality. Equally, an application could indicate to the NAT that it expects third parties to be able to use the NAT binding, and that the binding that the NAT will set up for the application should be managed as a port-restricted cone. There is much that could be achieved here that would allow applications to function with some level of determinism, rather than attempting to equip an application with a large and complex toolset of all the relevant techniques of NAT traversal that may be required by the application when confronted by various NAT behaviors.

In the meantime the NAT-behavior guessing game continues. The generic class of techniques that support this function is termed
*Unilateral Self-Address Fixing*
(UNSAF). This is a process whereby the local entity attempts to determine the address and port by which the entity is known externally, and to determine the characteristics of this association to understand in what contexts the external address may be used as a service rendezvous point for externally initiated communication. Work in this area [10] has exposed many relevant considerations, including a set of deficiencies noted in the previous section.

So, what would a NAT implementation look like if there were standards relating to NAT behaviors and the implementation were to comply with these standards? Numerous efforts have been made to document various forms of network- and application-friendly ways in which NATs could behave, but it would appear that such an effort will require the imprimatur of a standard in order to attain a level of general acceptance from NAT implementations. However, it is possible to predict that any such effort at a "standardized" form of NAT behavior will include the following considerations. The following set of behaviors is based on that enumerated in [13]:

- NATs must show endpoint-independent behavior for UDP-based bindings. This is to ensure that the NAT can support application rendezvous without the need for various multiparty relays and agents.
- NAT should not use port preservation nor port overloading, and should operate in a deterministic manner. Port preservation exposes the NATs to nonstandard behaviors when port preservation cannot be enforced. In addition, NATs must have deterministic behavior.
- A dynamic NAT UDP binding timer should be 5 minutes, and should avoid expiration timers of 2 minutes or less. This is to ensure that the timeout is long enough to avoid excessively frequent timer refresh packets.
- The NAT UDP timeout binding must use a timer refresh based on outbound traffic, and all sessions that use a particular binding should use a common refresh timer. This requirement is a security consideration, in that letting inbound traffic refresh the timer allows an external party to keep a port open on the NAT.

- The NAT filtering function should be address dependant. This represents a balance between security and utility.
- The timeout behavior of the NAT UDP filter must be the same as that of the NAT UDP binding timeout. This is intended to reduce the complexity of applications that are reliant on long-held NAT state.
- The NAT should support hairpin connections, using the external address and port.
- If the NAT includes ALG support, the ALGs should be configurable in terms of being able to turn off the ALG function on a per-application basis.
- NATs should support fragmentation and forwarding of packet fragments.
- NATs must support ICMP *Destination Unreachable* messages, and the ICMP timeout should be greater than 2 seconds.

### Learning from NATs

At this stage we can observe a few relevant lessons about NATs:

The first is that we need standards and we rely on standards. For many years the IETF has viewed standardization of NATs and their behavior as being an action that would encourage further deployment of a technology that was apparently considered undesirable. The result has been that NATs have been deployed for reasons entirely unconnected with the IETF and standardization, but because the original specification of NAT behavior was at such a general level each NAT implementor has been forced into making local decisions as to how the NAT should behave under specific circumstances. We now enjoy a network with widespread deployment of an active device that does not have consistent implementations and, in the worst cases, exhibits nondeterministic behaviors. This has made the task of deployment of certain applications on the Internet, including voice-based applications, incredibly difficult.

Whether NATs are good or bad, they would be less of a collective headache today if they shared a common standard core behavior. NATs for IPv6 may be considered to be unnecessary today, and it can be argued they represent no real value to an IPv6 site. But a collection of IPv6 NAT implantations with no common core behavior would constitute a far worse problem to application users. Standardization of technology at least eliminates some of the worst aspects of application level guesswork out of technology deployment.

Secondly, a little bit of security is often far worse than no security. NATs are very poor security devices, and in terms of their behavior with UDP, NATs afford only minor levels of protection. The task of securing a site from various forms of attack and disruption remains one of a careful exercise of assessment of acceptable risk coupled with detailed consideration of site-management functions. NATs are not a quick way out of this effort.

In considering NATs it seems that we are back to the very basics of networking. The basic requirements of any network are "who," "where," and "how," or "identity," "location," and "forwarding." In the case of IP, all these elements were included in the semantics of an IP address, and when addresses get translated dynamically we lose track of IP-level identity across the network. Maybe, just maybe, as we look at the longer-term developments of IP technology, one potential refinement may be the separation of endpoint identity to that of location, and as a potential outcome, NATs could readily manipulate location-based addresses while applications could look to a different token set as a means of establishing exactly who is the other party to the communications.

Of course, if we ever venture down such a path, I trust that such a move toward the use of explicit identities does not generate a complementary deployment of *Network Identity Translators*, or NITs, as an adjunct to the current set of NATs. Too many NITs and NATs will definitely send us all NUTs!

### Further Reading

There is no shortage of material on NATs from a wide variety of sources. The following is a list of IETF-related documents, encompassing both published *Request for Comments* (RFCs) and works in progress, that have been circulated as *Internet Drafts*.

**RFCs:**


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

[2] Srisuresh, P., and D. Gan, "Load Sharing Using IP Network Address Translation (LSNAT)," RFC 2391, August 1998.

[3] Srisuresh, P., and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663, August 1999.

[4] Tsirtsis, G., and P. Srisuresh, "Network Address Translation—Protocol Translation (NAT-PT)," RFC 2776, February 2000.

[5] Hain, T., "Architectural Implications of NAT," RFC 2993, November 2000.

[6] Srisuresh, P., and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, January 2001.

[7] Holdrege, M., and P. Srisuresh, "Protocol Complications with the IP Network Address Translator," RFC 3027, January 2001.

[8] D. Senie, "Network Address Translator (NAT)-Friendly Application Design Guidelines," RFC 3235, January 2002.

[9] Srisuresh, P., J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan, "Middlebox Communication Architecture and Framework," RFC 3303, August 2002.

[10] Daigle, L., and IAB, "IAB Considerations for Unilateral Self-Address Fixing (UNSAF) Across Network Address Translation," RFC 3424, November 2002.

[11] Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN—Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489, March 2003.

[12] Aboba, B., and W. Dixon, "IPsec—Network Address Translation (NAT) Compatibility Requirements," RFC 3715, March 2004.

**Internet Drafts:**

Internet Drafts enjoy a fleeting existence, and the following documents may not be available when you read this article. In such cases it is often the case that a decent Internet search will locate the document, or its successor.

[13] Audet, F., and C. Jennings, "NAT/Firewall Behavioral Requirements," work in progress, `draft-audet-nat-behave`, July 2004.

[14] Ford, B., P. Srisuresh, and D. Kegel, "Peer-to-Peer(P2P) Communication across Network Address Translators (NATs)," work in progress, `draft-ford-midcom-p2p`, June 2004.

[15] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for the Session Initiation Protocol (SIP)," work in progress, `draft-ietf-mmusic-ice`, July 2004.

[16] Jennings, C., "NAT Classification Results Using STUN," work in progress, `draft-jennings-midcom-stun-results`, July 2004.

[17] J. Rosenberg, J. Weinberger, R. Mahy, and C. Huitema, "Traversal Using Relay NAT (TURN)," work in progress, `draft-rosenberg-midcom-turn-01`, July 2004.

**Other Resources:**

NAT Check: Ford, B. and D. Andersen, Nat Check Website:

http://midcom-p2p.sourceforge.net

STUN Client and Server:

http://sourceforge.net/projects/stun

Phifer, Lisa, "The Trouble with NAT," *The Internet Protocol Journal*, Volume 3, No. 4, December 2000.

GEOFF HUSTON holds a B.Sc. and a M.Sc. from the Australian National University. He has been closely involved with the development of the Internet for the past decade, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector, and has served his time with Telstra, where he was the Chief Scientist in their Internet area. Geoff is currently the Internet Research Scientist at the *Asia Pacific Network Information Centre* (APNIC). He is also the Executive Director of the Internet Architecture Board, and is a member of the Board of the Public Interest Registry. He is

author of
*The ISP Survival Guide*
, ISBN 0-471-31499-4,
*Internet Performance Survival Guide: QoS Strategies for Multiservice Networks*
, ISBN 0471-378089, and co-author of
*Quality of Service: Delivering QoS on the Internet and in Corporate Networks*
, ISBN 0-471-24358-2, a collaboration with Paul Ferguson. All three books are published by John Wiley & Sons. E-mail:
gih@apnic.net