Anatomy

- A Look Inside Network Address Translators

Geoff Huston
August 2004

Over the past decade there have been a number IP-related technologies that have generated some level of technical controversy. One of these has been 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 fuelled 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've learned about technology and the process of standardization of technology along the way.

1. NAT Motivation

The first RFC document describing NATs was by Kjeld Egevang and Paul Francis in 1994 [RFC1631]. The original motivation behind the NAT work was based around efforts in the early 1990's 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. While 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 a number of 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, while 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 were 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 while 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 folk are finding out, they can be very obvious to the point of being highly disruptive to other forms of protocol operation.

2. 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 packet's contents, 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 the 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 will rewrite the source address in the packet header to a different value, and alter 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, which implies that the uniqueness of the address holds only for the site. Lets look at this using an example.

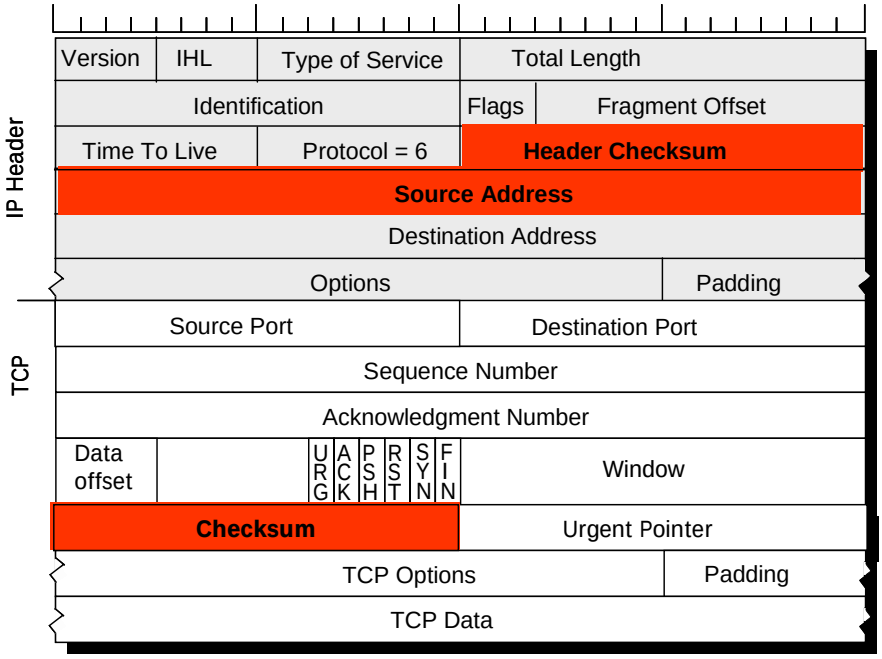

Figure 1 – TCP/IP header fields altered by NATs (Outgoing Packet)

As shown in Figure 2, how can local (private) host A initiate and maintain a TCP session with remote (public) host B? Host A will first look up the DNS to find the public IP address for B, and then create an IP packet using Host B's address as the destination address and Host A's local address as the source, and pass 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, as they are not globally unique addresses.

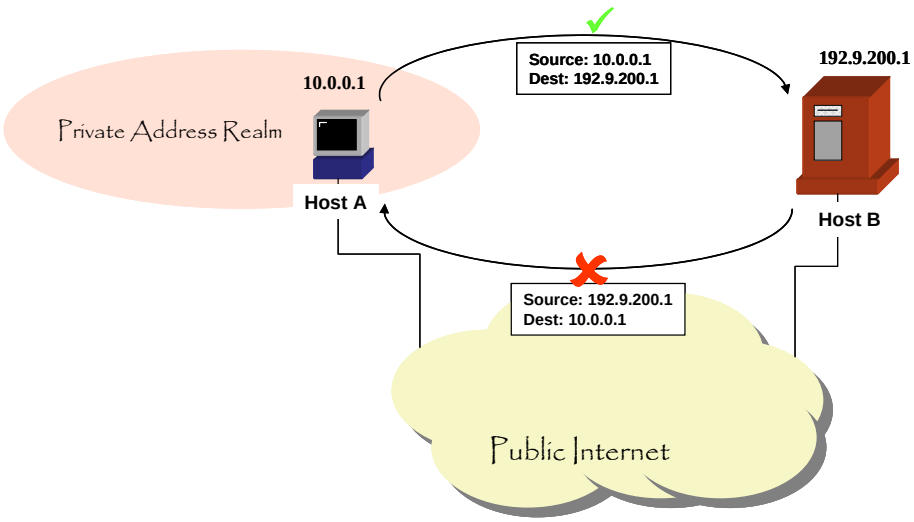

Figure 2 – Public/ Private Communication

With a NAT between Hosts A and B, the NAT will intercept Host A's outgoing packet and rewrite 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.

Once 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 its 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 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 B's perspective this is a normal session with a host at address X.

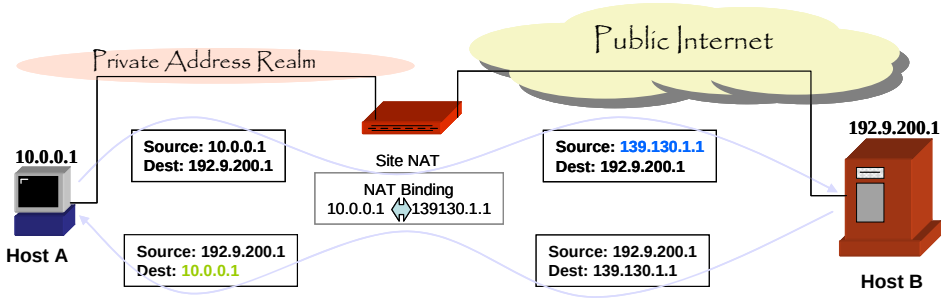

Figure 3 – NAT Traversal

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.

2.1 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 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's pool (Figure 4).

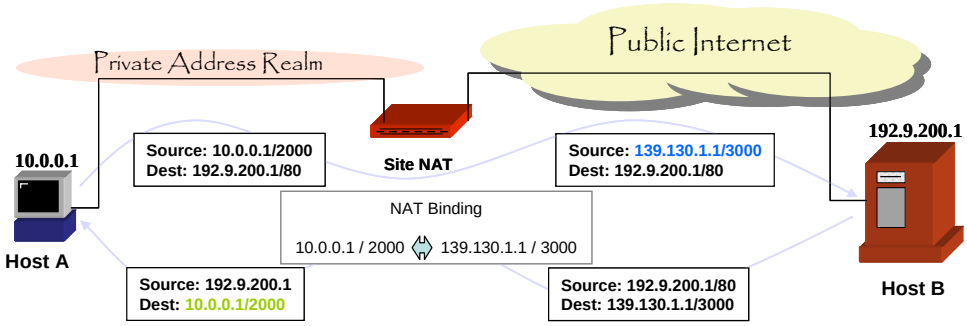

Figure 4 – NAPT Traversal

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 as local address and port pair to an external address and port pair.

Its a reasonable question to ask as to why should NAPTs bother with port translation ? Aren't straight address translations 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 65535 concurrent mapped sessions.

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

3. NAT Behaviour

There are two quite basic issues about the use of NATs. One is that NATs make applications “brittle” in that NATs support a particular style of application operation, and if the application deviates in any from this style then the application no longer works. The second is much more of a 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 behaviour become a matter for discovery by applications rather than something applications can predict in advance. This section explores this behavioural aspects of NATs in further detail.

3.1 Symmetry and Sessions

There are a number of ways that NATs can manage address mapping, 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 only translated to the local address 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 "end-point" sensitive. Symmetric NATs represent a restricted model of operation, where each NAT binding represents a window through the NAT that is only visible 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 will be translated to the mapped internal address and forwarded through the NAT. Symmetric NATs represent the most restrictive form of behaviour, while 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 will any remote service entity to direct packets back through the port window.

NATs can be further refined by having different behaviours 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 behaviour has lead to an exercise in categorizing NAT behaviours and developing a discovery protocol whereby a pair of cooperating systems can discover if there is one or more NATs on the network path between them, as well attempting to establish the type of NAT.

3.2 Discovering NAT Behaviours and STUN

NAT behaviour has not been the topic of any industry standardization efforts, and it should not be surprising to learn that, given that there are a range of possible NAT behaviours 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” behaviour 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 remote host's packets 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. While 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 an increasingly important question. 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?"

So 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, and, 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 behaviour. All that is left is a somewhat unsatisfying guessing game, where the application is forced to send particular types of packets through the NAT to its counterpart on the other side, and by comparing the original packet with its address and port transformed the two ends of the application attempt to guess the nature of the systematic transforms of the NAT.

In the case of TCP it appears that the prevalent NAT behaviour 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's address and port bindings for the local host are coupled with the address and port of the destination host, and only packets with a source field of the destination host can pass packets back through the NAT to the local host's TCP session. In other words, once a TCP session has been established within a NAT only the two end points 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's discard response. The fine-grained behaviour 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 can also 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 two sides’ sequence number windows and associated window sequence number scaling values and not adjust the session’s binding timer for TCP packets with sequence numbers outside the sequence number window with their FIN or RST flags set.

These NAT behaviours are based on the explicit signalling 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, as there is no explicit session state within a UDP packet exchange, and various NATs behave differently with respect to UDP-based bindings. There are various classes of NAT behaviour that relate to how UDP bindings are managed within a NAT. These have been classified into four types of behaviours [RFC3489]:

Symmetric

We've 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 public-side 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 behaviour under UDP, and it has been observed that this form of NAT behaviour is becoming quite rare, as it prevents the operation of all forms of applications that undertake referral and handover.

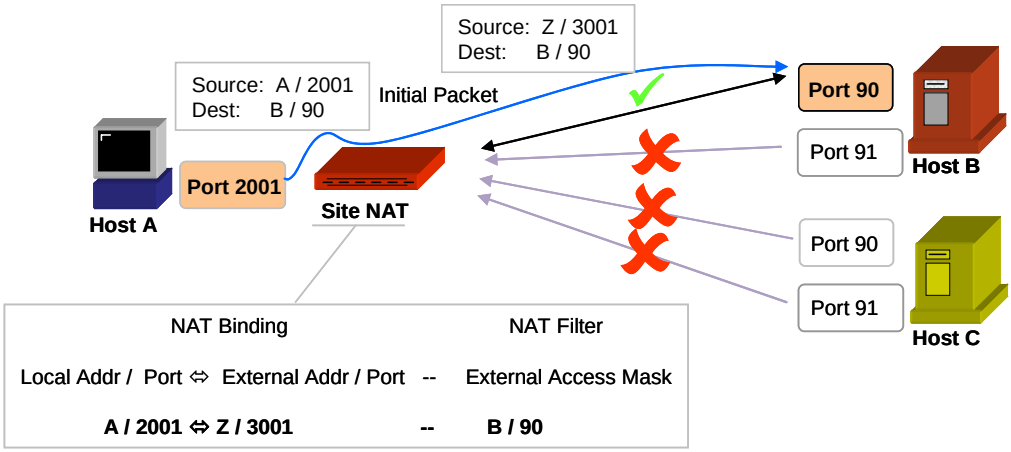

Figure 5 – Symmetric NAT

Full Cone

A full cone NAT is the least restrictive form of NAT behaviour, where the binding of a local address and port to a public-side address and port, once established, can be used by any remote host on any remote port address.

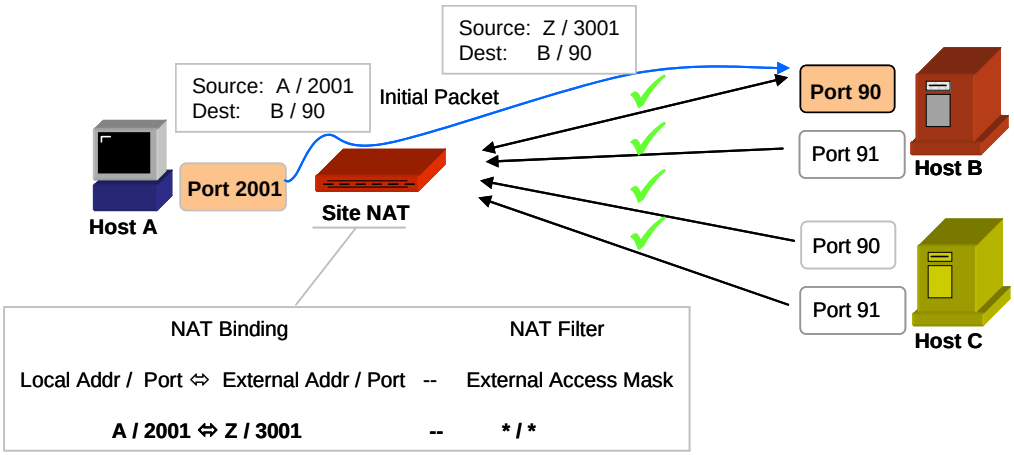

Figure 6 – Full Cone NAT

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 once the binding is created.

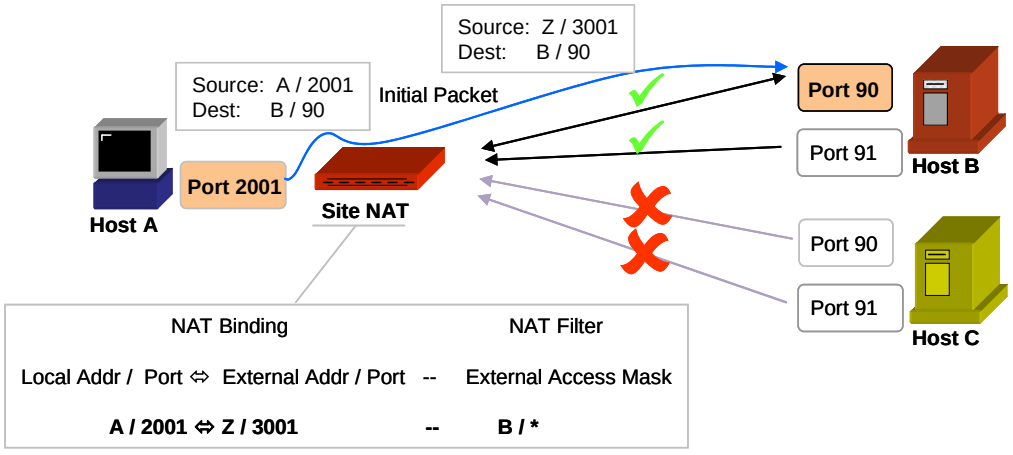

Figure 7 – Restricted Cone NAT

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.

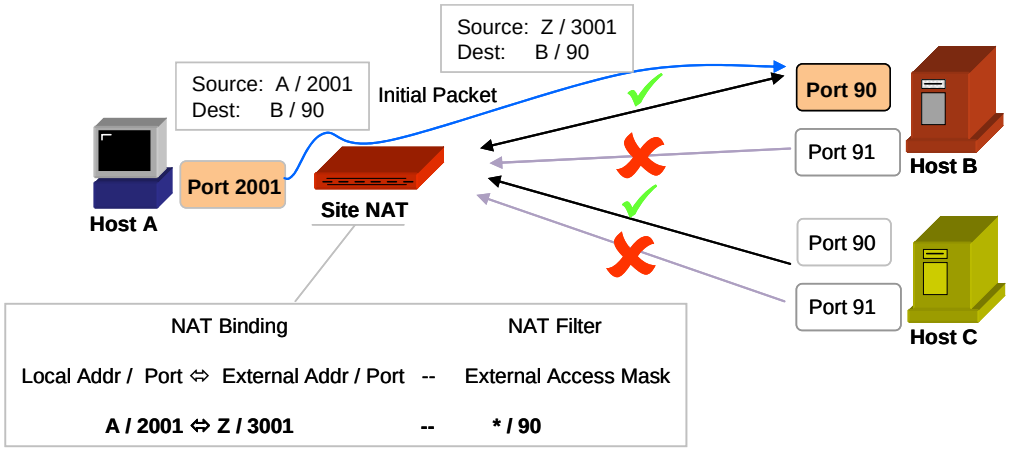

Figure 8 – Port Restricted Cone NAT

So can an application tell if there are one or more NATs in the path, and, if so, what form of behaviour the NAT is using? For this purpose the STUN (Simple Traversal of UDP through NATs) protocol has been developed [RFC 3489]. STUN is 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 the 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 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, as 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 to 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 behaviour. 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 restricted cone, and the lack of a response indicates the presence of a port 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.

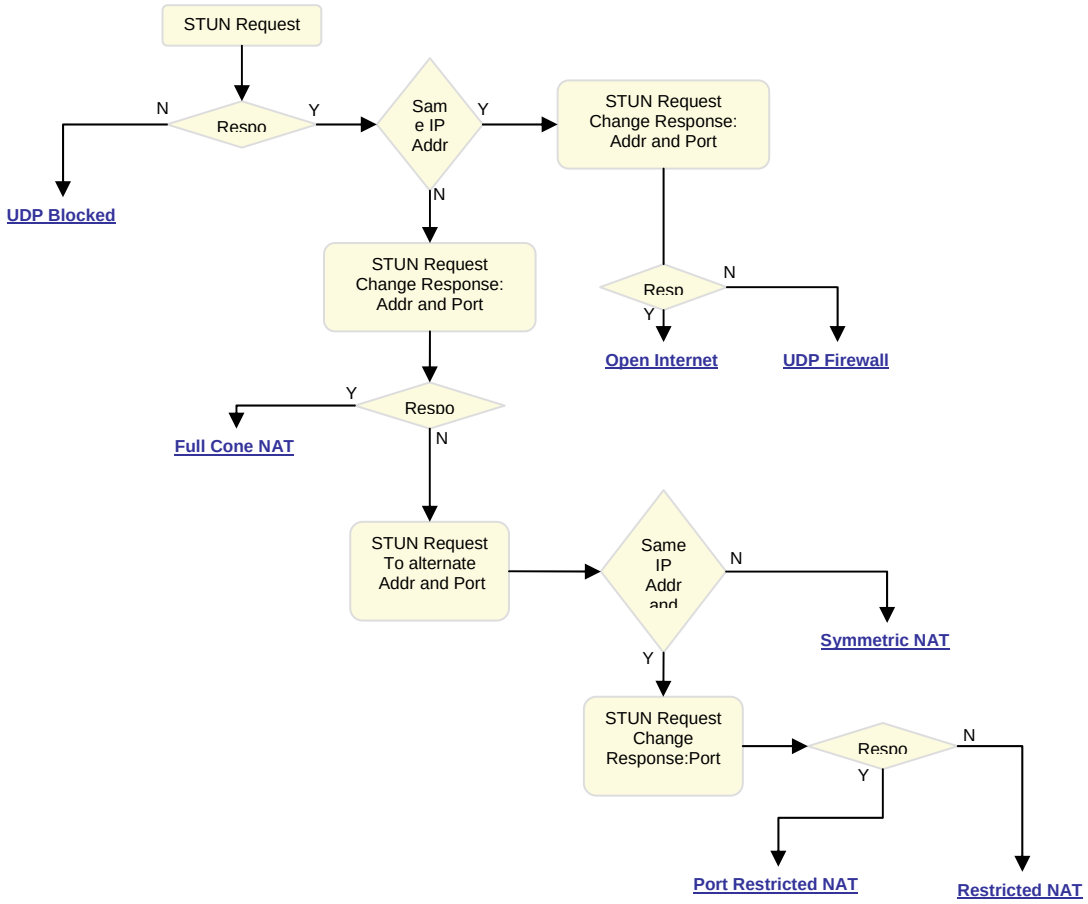

Figure 9 – NAT Discovery Process using STUN

3.3 Further Behaviours: Hairpins and Determinism

It would be good that NAT behaviour remained that simple. However, this is not the case, as some further tests on NATs reveal further differences in various NAT implementations [draft-jennings-midcom-stun-results].

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, once created, are available to either side of the NAT.

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.

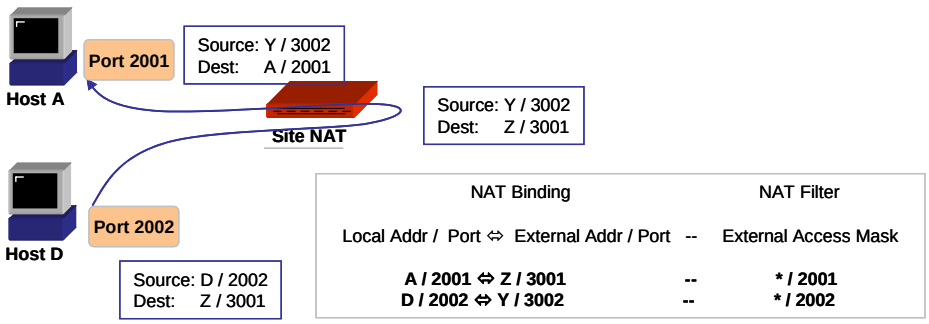

Figure 10 – Hairpin NAT Operation

The second is in the general class of NAT determinism. Non-deterministic NATs change their binding behaviour 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 behaviours differ. To explain primary, secondary and tertiary behaviours, 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 behaviour 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 behaviour 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 behaviour occurs when a third binding is added to the NAT, as, again, the behaviour 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 behaviours can all be classified as non-deterministic, in that the NAT behaviour 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 behaviours based on this STUN-derived knowledge of the NAT type, non-deterministic 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 once a binding state is established it may be disrupted or altered by subsequent traffic.

3.4 Another approach to classifying NATs

Further tests on NATs reveal that the various behaviours 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 [draft-audet-nat-behave]. The key observation here is that NATs are the conjunction of two distinct behaviours 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 packet's logical direction 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 compares 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 behaviours on the basis of a single biding 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.

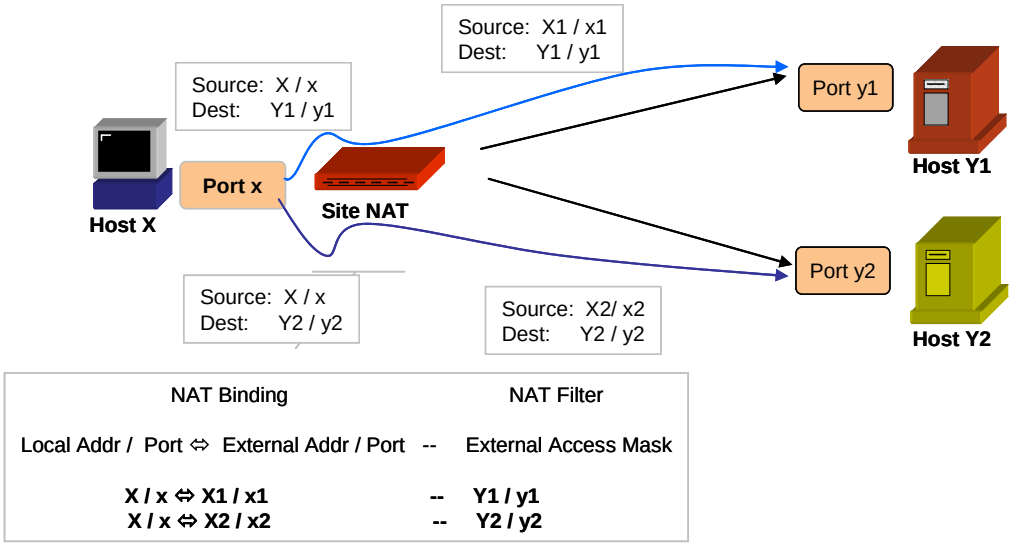

Figure 11 – Outbound connections from a common source

The behaviours of NATs under this condition can be classified under a number of behavioural aspects.

3.4.1 Binding

Binding behaviour can be seen as the amalgam of three somewhat distinct design decisions, namely the manner in which a binding is generated, the behaviour 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 Behaviour:

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 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, destination port)

Port Binding Behaviour:

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 behaviour 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 will preserve 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 will have the same destination address and port, The NAT will determine the appropriate binding based on the source address and port in the reserve packets. This requires an Endpoint Address and Port dependant binding behaviour. In the case where 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 non-deterministic behaviour.

Binding Timer Refresh:

Bidirectional
The NAT will not keep the binding active indefinitely, and will normally remove 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 will consider as packets that reset the timer. In a bidirectional binding timer refresh packets from either the local hosts or an external host that uses the NAT binding case the NAT binding expiration time to be reset.

Outbound
An outbound binding timer refresh NAT will only reset the expiration timer 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 per session basis, or may be on a per binding basis in the case that 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 case the binding's expiration timer to be refreshed.

Transport Protocol State
While the above 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. While it would be an option to drop the NAT's 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.

3.4.2 Filtering

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

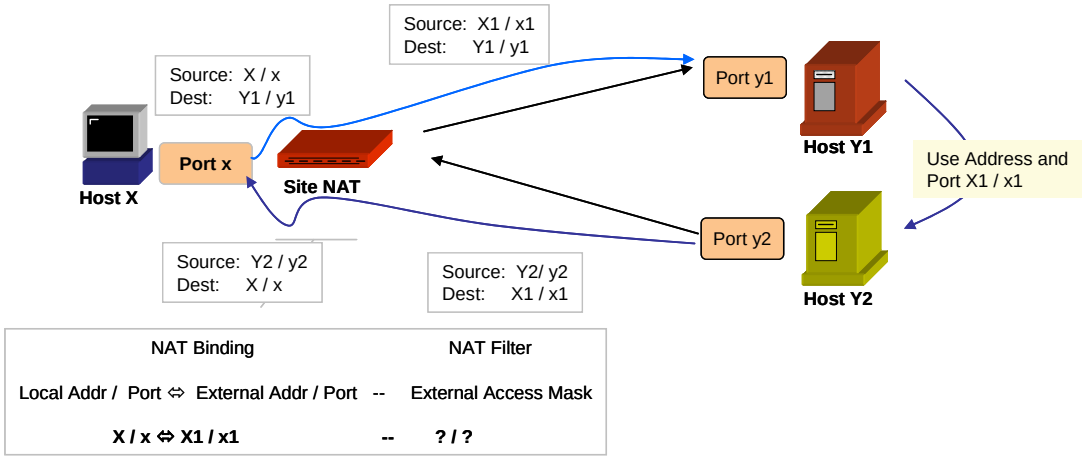

Figure 12 – Inbound test

External Filtering:

Endpoint Independent
The NAT will 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 will filter and discard 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 will filter and discard 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**.

3.5 NAT Behaviours

The approach of carefully identifying the areas where NAT behaviours differ, and classifying these behavioural 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 behaviours, and hopefully also refer to the same set of actual behaviours 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 behaviours. Experience with this form of classification has exposed further variations in NAT behaviours, and this has lead to a form of NAT classification that firstly uses a delineation of **binding** and **filtering** behaviours, and then classifying 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 **non-deterministic** manner.

This exercise is not another study in comparative taxonomies. A NAT has no standard way in which to advertise its presence, or any standard way in which to advise protocols or applications of the particular behaviours it will apply 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 behavioural classification is to associate test sequences that expose the presence of a NAT, and to determine its behaviour. This allows applications to invoke a test procedure that exposes a NAT implementation's particular choice of behaviours, 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 behaviour 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 where 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 as an end-to-end pair of NAT bindings cannot be established.

4. 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 don't conform to any particular well-defined behaviour set. NAT behaviour varies across implementations, and NATs have no ability to disclose their particular behaviours to applications that are attempting to compensate for their presence in the path. It is extremely challenging for applications to reliably predict the behaviour of the NATs that lie in the path, and more so in the face of multi-party 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.