Internet-Draft
Expires: 21 August 2002

K. Moore
University of Tennessee

21 February 2002

Recommendations for the Design and Implementation of NAT-Tolerant Applications

draft-moore-nat-tolerance-recommendations-00

Status of this Memo

This document is an Internet-Draft and is subject to all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Comments on this document should be sent to the author, whose address appears near the end of this document. Please include the document identifier draft-moore-nat-tolerance-recommendations-00.ps in any comments.

Abstract

This document describes a set of recommendations for the design and implementation of networked applications. These recommendations are intended to provide such applications with increased ability to operate in the presence of Network Address Translators (NATs).

This document is being provided only as information, with no intent to impose these recommendations on application developers. It is not possible to make all networked applications tolerant of NATs; and for those applications for which NAT-tolerance is possible, adherence to the criteria needed to maximize NAT-tolerance may result in unacceptable loss of performance, scalability, or manageability for that application.

1. Introduction

The widespread deployment of Network Address Translators (NATs) in the network presents several difficult problems for developers of many kinds of applications. This document presents a list of recommendations for design and implementation of networked applications, which are designed to provide an increased degree of NAT-tolerance for applications implemented according to these recommendations. It is not claimed that these recommendations are appropriate for every application, nor that it is feasible to make every networked application NAT-tolerant.

This document is written primarily for designers and/or developers of applications for which the designer or developer has no control over the environment in which the application will be deployed, but who wish the application to be usable in as many environments as possible. For this reason, this document glosses over the differences between various kinds of NATs (such as between ordinary NATs and NAPTs), assuming that developers who follow these recommendations wish their applications to be maximally tolerant of all forms of NAT. Definitions of several terms used in this document, and a description of various flavors of NAT, can be found in [1].

Similarly, even though many NATs can be configured to be less hostile to some applications (such as by "nailing up" an address and/or port mapping for a particular application), this document assumes that this kind of workaround may not always be feasible for some environments in which it is desirable to deploy an application. Similarly, RSIP [2] and solutions which might be developed by the MIDCOM working group [3] are also assumed to be either inapplicable or unavailable in some environments. However, if the characteristics of the deployment environments are well-known and stable, or if it is known to be feasible to explicitly configure the intended deployment environments to accommodate the new application, it may be possible to relax some of these recommendations.

These recommendations also assume that it is desirable to be able to run the same application code in a variety of environments (e.g. non-NATted networks, NATted networks with or without DNS ALG [4] support, large multi-site enterprise networks behind a NAT, small private networks behind a NAPT, and private enterprise networks that are interconnected to one another via NATs without necessarily having access to the global Internet). It is also assumed to be desirable for application components in dissimilar environments to be able to interoperate with one another. This limits the applicability of several fixes to NAT problems such as DNS ALG and RSIP. Fixes which work in one environment, or for one application, may not work for another environment or application. For instance, there are limitations in both the DNS protocols and the deployment of DNS in practice which preclude use of DNS as a general solution to NAT problems.

The intent of this document is similar to the stated intent of RFC 3235 [5], but it has a different perspective. This document is written from the perspective of a developer of distributed applications trying to make them work in the face of the damage caused by a variety of different kinds of NATs. RFC 3235 treats different kinds of NATs separately, but the application designer will rarely have knowledge of, or control over, the types of NATs in use where his applications are deployed. RFC 3235 also contains several misleading or incorrect statements regarding the impact of NATs on applications, and it overstates the applicability of DNS names as a solution to NAT damage to usability of IP addresses. This document therefore provides its own summary of the impact of NAT on applications, and makes recommendations which differ from those in RFC 3235.

Some of the techniques described in these recommendations may also allow an application greater ability to operate in the presence of firewalls. It is not the purpose of this document to provide the ability to circumvent a site's network security policy enforcement mechanisms. Because there are other reasons to deploy NATs than as a security mechanism, the presence of one or more NATs in a network that prohibit certain kinds of network traffic is not taken as evidence that such traffic is prohibited as a matter of policy.

1.1 Problems caused by NATs

Many of the problems caused by NATs have been discussed elsewhere, from various points-of-view (for example [6] and [7]), and it is not the purpose of this document to discuss those problems at length. However, an analogy, followed by a brief summary of the problems, may help establish the need for these recommendations and the rationale behind them.

Let's say you are operating a package delivery business. However, the area you are serving has several interesting characteristics:

- Every door that a delivery person has to pass through in order to pick up or deliver a package is potentially locked in such a way that it can only be opened from one side;
- If a delivery person finds a locked door in his path, he/she must wait until the door is opened for him or her specifically;
- A delivery person has no way of asking that a locked door be opened from the other side;
- Deliveries are made to named recipients, not to rooms or locations;
- Assignments of rooms to occupants change frequently, randomly, and without notice;
- There is no directory listing the locations of room occupants; and
- All room occupants keep their doors locked.

This is analogous to the difficulty of writing applications that are tolerant of NATs. The "doors" are analogous to NATs (often NAPTs) which only allow traffic in one direction; and the random changes in room assignments are analogous to NAT's changing mappings between internal and external addresses. While not all NATs impose all of these limitations, these are representative of limitations commonly imposed by NATs. Applications developers need to assume that such limitations are in place if they want their applications to work in the widest possible set of environments.

If it seems that NATs are more tolerant of some applications than this (such as web browsers or mail readers), it is because, for those specific uses of those applications, the "doors" happen to "open" in a favorable direction; and/or because the NATs have been specially adapted (with application-level gateways) to allow those specific applications to work. This is why NATs appear benign to some applications while imposing a considerable barrier to deployment of others. A wide variety of applications are hampered by NATs, including instant messaging, audio and video conferencing, Internet telephony, distributed multi-party simulations, multi-player games, distributed computation (including "Grid computing"), and peer-to-peer applications in general.

The limitations imposed by NATs on applications developers generally fall into the following broad categories:

- *Limitations on the use of IP addresses.* In a NATted environment, IP addresses are no longer globally unique, and an IP address that is valid to reach a host from one addressing realm many not be usable to reach that host from another addressing realm.
- *Limitations on the use of transport-layer protocols.* NATs (NAPTs in particular) cannot be expected to support all transport-layer protocols, because they examine or manipulate transport-layer addresses (e.g. ports), in addition to network-layer IP addresses, and each transport-layer protocol has its own format for such information. Also, for transport protocols such as UDP which do not involve explicit connection setup and teardown between parties, the duration of the address bindings will vary from one NAT to another.
- *Limitations on the direction in which a communication session may be initiated.* Many kinds of NATs (especially NAPTs) will not permit a communication session to be established from a host on the "outside" of the NAT to a host on the "inside" of the NAT.
- *Limitations on the lifetime of host-to-address bindings.* NATs often impose short lease times on addresses that they assign to hosts; thus, an IP address-to-host binding may not remain valid for long even within the same addressing realm.
- *Limitations on the scope of address bindings.* Some NATs establish address bindings on a per-external-host basis. Therefore an address which can be used to reach host A from host B (through such a NAT) may not be usable from host C, even if B and C are in the same addressing realm.
- *Limitations on the ability to use IP Security.* NATs are generally incompatible with end-to-end IPsec.
- *Increased difficulty of problem diagnosis.* Because of multiple address realms and changing address-to-host bindings, problems with applications (whether or not they are caused by NATs) can be difficult to diagnose and difficult to reproduce.
- *Degraded network reliability.* NATs generally introduce additional sources of failure in the network, both because stateful NATs fail independently from any connection endpoint, and because it is not generally possible for the network to route traffic around a failed NAT.

1.2 Problems with DNS

DNS names are often suggested as a replacement for IP addresses for use within applications. However, applications developers cannot depend on DNS names as a replacement for IP addresses, for the following reasons:

- Many hosts, especially those in private small office or home networks, are not assigned fully-qualified DNS names. Such hosts may be assigned names which are not meaningful outside the hosts' local network.

- Many hosts do not know their own fully-qualified DNS names. In addition, many hosts use protocols other than DNS (e.g. WINS, NIS, NIS+) to lookup DNS names. Even if an application component can obtain its host's IP address(es) and is willing to implement the DNS protocol directly (bypassing system libraries) to look up the DNS name associated with its address(es), the host may lack the necessary configuration information (root server and/or resolver addresses) to allow it to do this. In addition, if the host is behind a NAT, a DNS name returned from a PTR query of the host's IP address might not be globally usable, since the host's address is not in the global addressing realm.
- A host that obtains its IP address using DHCP may not have a DNS name that is stably bound to the host. Even if a PTR lookup yields a DNS name for the IP address that is currently assigned to that host, the DNS name might change over time.
- DNS lookups are often slow and/or unreliable, especially when it is necessary to "fail-over" a query from one server to another, or when all servers for a particular zone are unreachable.
- DNS lookups occasionally (some would say frequently) produce erroneous results, especially when resource records are changed without increasing the serial number, without regard to the TTLs of previous versions of those records which might be cached elsewhere, or without updating glue records stored on other servers.
- Many environments employ "two-faced" DNS servers, allowing lookup of certain DNS names only from within an enterprise network. While fully-qualified DNS names obtained from such servers are likely to be globally unique, they may not be globally usable.
- DNS names are frequently used as names of services that are implemented by multiple hosts, rather than names of specific hosts. Accordingly, a DNS lookup for address records that results in multiple IP addresses may refer to several different hosts rather than different interfaces at the same host. Since an application component may maintain critical state which is accessible only from the host on which that component resides, DNS names used as service names do not provide an effective means to reach a particular application component.

For these reasons, DNS names are often less reliable than IP addresses, even in a NATted environment. Some of these problems can be fixed by changes to the configuration of DNS servers; however, those servers may not be under the control of those desiring to use the application. Between the NAT problems and the DNS problems, a component of a portable application can depend on neither DNS names nor IP addresses as an exclusive means of reaching other components.

Some have claimed that networks which don't support globally-usable and stable DNS for every host are fundamentally broken, and that it is unreasonable to expect applications to run in such environments. It would be at least as reasonable to argue that NATted networks are broken and that it is unreasonable to expect applications to run in the presence of NATs. Regardless of which is "true", from the standpoint of an application developer, telling a potential customer that he/she must configure his network in a certain way (e.g. with or without NATs or with a particular DNS setup) has the effect of reducing the set of environments in which the application can operate, and therefore, the potential market and user base for that application.

2. Rules for applications in NATted environments

The following rules are directly implied by the characteristics of NATs.

(R1) *Do not assume that IP addresses are unique.*

Applications must not assume, for any purpose, that an IP address is uniquely bound to any host or component, even at a specific time. For example, an application component that uses its host’s IP address in an attempt to construct a unique filename may fail if another component with the same IP address (in a different addressing realm) uses the same algorithm for constructing a file name. For similar reasons, the IP address of a host or component must not be used as an index to application state information (say, the status of a remote process) that is associated with that host or component.

Note that the uniqueness of an IP address-to-host binding at some time is distinct from both the "stability" of that address-to-host binding, and whether messages sent to that IP address from some location will reach a host that is associated with the address.

(R2) *Do not use IP addresses to identify components or hosts.*

IP addresses must not be used as host identifiers, both because an IP address is not guaranteed to be unique, and because the binding between an IP address and a host is not guaranteed to be stable. For similar reasons, an (IP address, port) pair is not usable as a component identifier.

(R3) *Do not assume that IP address-to-host bindings are stable.*

Such bindings may change for a variety of reasons: because of changes in address mappings across NATs, because of short DHCP lease lifetimes, because of mobile hosts, or because of intermittent connectivity.

(R4) *Do not assume a completely connected network.*

In other words, do not assume (in the absence of explicit configuration information) that direct communication is possible from any connected host to any other connected host. This is such a fundamental difference from the traditional Internet programming model that it needs to be stated explicitly. It follows that application components must be prepared to employ intermediaries and perform their own message routing if they are to communicate in a NATted network.

(R5) *Do not assume that the ability of host A to establish a connection to host B implies that host B can establish a connection to host A.*

Many kinds of NATs only permit connections in one direction.

(R6) *Do not assume that the ability of host A to establish a connection to host B using protocol X implies that host A can establish a connection to host B using protocol Y.*

This is partially because a NAT may have different degrees of support for different transport-level protocols, partially because of the use of port numbers in NAPTs to route different traffic for different ports to different hosts, and partially because some protocols require ALG support in order to function over NATs.

(R7) *Do not assume that an IP address that works from one location in the network will work from another location (or from another component of the same application).*

This follows directly from the existence of multiple addressing realms.

(R8) *Do not assume that traffic sent to different ports at an IP address will be received by the same host.*

This follows directly from the existence of NAPTs.

(R9) *Do not authenticate, grant access permissions, implement policy, or account for usage, based on an IP address.*

This has long been a bad idea, but the lack of address stability in the presence of NATs make it even less advisable.

(R10) *Do not make assumptions about the relationships of IP addresses to network topology.*

NATs that separate large networks can hide the topology of those networks.

(R11) *Use TCP, rather than some other transport protocol, if it is at all suitable.*

Support for UDP varies from one NAT to another, and NATs that support UDP may revoke UDP address bindings at arbitrary times and without notice. Support for other transport protocols is even less likely.

(R12) *Do not use IPsec.*

(R13) *Do not use IP multicast.*

(R14) *Long-running applications must be able to recover from connection failures caused by NATs.*

Such failures will occur with more frequency in a NATted environment due to failure of the NATs themselves or the NAT's revoking of an address binding that is in use (especially with UDP). Such failures will occur independently of link failures, router failures, or end-system failures, and the nature of NATs makes it difficult to address such failures by providing greater redundancy in the network. They must therefore be addressed by end-system components.

The following rules are derived from observed characteristics of DNS.

(R15) *Do not assume that every host has a DNS name from which useful addresses can be obtained, or that a host can determine such its DNS name even if one exists.*

(R16) *Do not assume that a DNS name obtained via a PTR lookup of an IP address is stable.*

(R17) *Do not assume that a DNS name that works from one application component is usable from other application components.*

2.1 What assumptions can be made?

As illustrated above, NATs break many of the provisions of classic Internet service. Given that an application on an Internet host can no longer expect that the network will make a "best effort" attempt to send traffic to another host, what assumptions can a developer reasonably make to allow his or her application to operate as widely as possible?

The following assumptions appear to be reasonable:

- Any host can connect to a host that is located on the global Internet. (Connections may also be possible to a host that is behind a NAT which provide DNS ALG service).

  However, it is not reasonable to expect that every host that needs to accept incoming connections can be located in the global addressing realm. Nor is it reasonable to assume that such hosts have stable addresses.

- Hosts which have stable addresses in the global addressing realm, that are expected to accept unsolicited traffic, will also have stable and usable DNS names.

  If a host's global IP address is stable, the barrier to providing a stable DNS name for that host is sufficiently low that sites can be expected to provide one.

However, these assumptions are insufficient to permit many kinds of applications to operate without explicit configuration and/or support from the network. In particular, for an application component to send unsolicited traffic to another application component that is behind a NAT (especially a NAPT), it will often be necessary to employ some sort of proxy or traffic forwarder to circunvent the NAT's traffic restrictions and lack of address stability. Even then, the need to provide such proxies can impose a significant barrier to deployment.

3. Recommendations For the Construction of NAT-Tolerant Applications

The following recommendations were intended to allow networked applications to function (as well as is reasonably feasible) within the constraints imposed by NATs and by frequently-observed limitations of DNS.

These rules are based on the following model:

- A networked application consists of one or more, perhaps an arbitrarily large number, of "components" which interact over the network. The application continues to exist as long as any components of the application exist, perhaps indefinitely. The set of components of an application may change during the lifetime of the application. Each of these components may potentially be located on any host in the Internet. Multiple components may reside on a single host.

- Any component may potentially need to communicate with any other component. In general, it is necessary for a "connection" to exist between two components before they can communicate. However, it is not assumed that the component wishing to send data is necessarily the one that establishes the connection over which the data is sent.
- "Connections" between components fall into several categories, which sometimes need to be treated differently:
  A. Connections made by "bilateral agreeement", for which both components are aware in advance that they will need to communicate.
  B. Unsolicited connections to components on hosts that have at least one stable component address which is generally reachable from all other components.
  C. Unsolicited connections to other components.
- Each component has zero or more "component addresses" at which it may be able to accept connections. A "component address" can be an IPv4 address, an IPv6 address, or a DNS name. Other kinds of addresses might also be useful.

This model specifically does not assume a client-server relationship between components, nor does it assume that components with particular roles are placed in favorable locations (relative to NATs) in the network topology. Such a model is appropriate for a wide range of applications.

In order for components that are behind NATs to be able to accept incoming connections, such components may employ "proxies". Each unreachable component maintains an open connection to one or more proxies, which are (for the purposes of that application) able to receive and accept connections from any other component in the application. Such a proxy acts as one endpoint of a tunnel, with the other endpoint being the application component that is using the proxy. The proxy then forwards traffic between "inside" and "outside" components.

(1) *Provide each component with a unique identifier*

Each component of the application should have a unique component identifier for use within the application. Since usable DNS names are not always available and IP addresses are not always unique within the application, neither of these makes a satisfactory component identifier. If the application is intended to support a large number of components, the easiest way to ensure uniqueness of component identifiers within the application is to base the component identifier on some other globally-unique identifier, such as a hardware serial number or the MAC address of a network interface. Alternatively, a unique identifier may be obtained from a high-quality random number generator with sufficient bits to make collisions unlikely.

This component identifier is intended for use within the application, not for use by humans. It may be necessary to provide a separate human-readable identifier for use in messages sent to or received from humans. Such identifiers are best derived from the DNS name of the component's host (if one exists and is known), or some other human-readable name associated with that host.

Applications should not assume that such identifiers are globally-unique.

(2) *Each component must have authentication credentials*

Authentication credentials are needed to verify the identity of peers after connection establishment, and/or to authenticate to proxies. Typically a public and private key pair would be generated and the public key associated with the component identifier. In some cases the key pair would need to be generated in advance, whereas in other cases it could be generated upon component creation.

(3) *Components should support access via proxies*

Each component that wishes to accept unsolicited incoming messages should be capable of being configured to establish and maintain a connection to one or more proxies, and to advertise those proxies' address(es) as means of establishing connections with that component.

(4) *Advertise multiple methods by which a component may be reached*

Since neither an IP address nor a DNS name suffices in all cases to provide the means to reach a component from an arbitrary other component, it may be necessary to support other means by which components can advertise to one another a variety of means by which one or more particular components may be reached from other locations. Such methods may include IPv4 addresses (potentially from multiple addressing realms), IPv6 addresses, and DNS names.

If proxies are supported by the application, the addresses or DNS names advertised may be the addresses of either the host supporting the component, or a proxy which is willing to forward PDUs to the component.

(5) *On establishing a connection, components should mutually verify each others' identities*

This is necessary because a change in an address binding within a NAT, or a change in an address-to-host binding, may cause the same address to reach different components at different times. A mismatch should be treated as a failure to establish a connection to that particular address. If other addresses are available for that component, attempts should be made to establish a connection via those addresses.

In order to thwart potential attacks, designers are strongly recommended to use cryptographic authentication techniques as a means of verifying component identities.

(6) *When a bilateral agreement exists to establish a connection between two components, each component should attempt to establish the connection.*

This insulates the application from NATs which might block such connections from one direction or the other.

(7) *When attempting to establish a connection using a DNS name, perform the DNS lookup prior to each connection attempt.*

Most DNS lookup interfaces (e.g. gethostbyname()) do not preserve TTLs, so a name-to-address binding might have expired. Even if the TTL of a previous lookup has not expired, the name-to-address binding may have a different lifetime than that of the address-to-address binding internal to a NAT. Support for DNS ALG cannot be assumed; but there are hosts behind NATs which update DNS via other means.

(8) *It is necessary to detect and recover from failure of open connections.*

This attempts to compensate for temporary NAT failures or for changes in address bindings.

Because application components are generally unaware of the network topology and in particular the relative location of any NAT devices, it is generally necessary for both components which were participating in a connection to detect and attempt to recover from a broken connection. This can be accomplished by TCP keepalives (when these are supported by the underlying platform), or by explicitly exchanging messages between application components at periodic intervals.

Alternate addresses should be used if the attempt to re-establish a connection at the initial address fails.

(9) *Connections should be kept open until it is known that neither party needs to send any more messages to its peer.*

Alternatively, if it is known that component A may reliably estabish a connection to component B, A may arrange to poll B periodically in case B has messages for A. In any case, conditions for connection termination should be explicitly defined by the application protocol.

(10) *Messages (PDUs) sent between components must be self-delimiting.*

This is necessary to allow proxies to separate one message from another.

(11) *Messages (PDUs) sent between components must adhere to a uniform format, with destination component identifers in well-known locations.*

This is necessary to allow proxies to identify the destinations of messages and to route them to the appropriate components.

(12) *It is generally necessary to ensure that proxies restrict access to authorized users or components, so that they are not used by unauthorized parties as a means of attacking application components or hosts.*

This prevents proxies from being used to hide the source of attacks.

4. Limitations

These rules and mechanisms are not a general solution to the NAT problem. No set of rules or mechanisms which fail to restore a fully-connected network with a global address space can completely compensate for the introduction of NATs within the network.

While adoption of these rules and/or mechanisms may improve NAT-tolerance, they can also adversely affect the efficiency and scalability of an application, by consuming additional resources such as network bandwidth, per-connection state on hosts and address bindings on NATs.

5. Security Considerations

Marketing claims by various NAT vendors notwithstanding, the security benefits of NAT are largely illusory. In most cases the same security benefit could be obtained without using network address translation, and without impairing the flexibility of networks to support desirable applications as severely as NATs do. Nevertheless, some may feel that the use of the mechanisms described in this document subvert the technical measures they have chosen to enhance the security of their networks. Those relying on NAT to enforce security policy are urged to migrate to mechanisms which are more robust and/or which accommodate a wider variety of applications.

The NAT-imposed instability of address-to-host bindings has made the practice of trusting a host's (or component's) IP address even more dangerous than it traditionally was. Applications writers are (again) strongly urged to use suitable cryptographic authentication mechanisms to verify the identity of hosts and components and the integrity of data communications between those components. Similarly, when proxies are employed it is necessary to ensure that they cannot be used as a means of attacking the application, other applications, hosts, or networks.

6. Author's Address

Keith Moore
University of Tennessee CS Dept.
1122 Volunteer Blvd., Suite 203
Knoxville, TN 37996-3450
USA

email: moore@cs.utk.edu

7. References

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

[2] M. Borella, D. Grabelsky, J. Lo, K. Taniguchi. Realm Specific IP: Protocol Specification. RFC 3103, October 2001.

[3] IETF Middlebox Communication (MIDCOM) working group. Charter at http://www.ietf.org/html.charters/midcom-charter.html

[4] P. Srisuresh, G. Tsirtsis, P. Akkiraju, A. Heffernan. DNS extensions to Network Address Translators (DNS_ALG). RFC 2694, September 1999.

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

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

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