Intended Status: Best Current Practice
BEHAVE WG
Internet-Draft
Expires: September 5, 2007

B. Ford
M.I.T.
P. Srisuresh
Kazeon Systems
D. Kegel
kegel.com

March 5, 2007

Application Design Guidelines for Traversal through Network Address Translators <draft-ford-behave-app-05.txt>

Status of this Memo

Abstract

This document defines guidelines by which application designers can create applications that communicate reliably and efficiently in the presence of Network Address Translators (NATs), particularly when the application has a need for "peer-to-peer" (P2P) type of communication. The guidelines allow a P2P application to work reliably across a majority of existing NATs, as well as all future NATs that conform to the behave requirements specified in

companion documents. The NAT traversal techniques described in the document do not require the use of special proxy or relay protocols, do not require specific knowledge about the network topology or the number and type of NATs in the path, and do not require any modifications to IP or transport-layer protocols on the end hosts.

Table of Contents

1. Introduction and Scope

The present-day Internet has seen ubiquitous deployment of Network Address Translators (NATs), driven by a variety of practical challenges such as privacy and the ongoing depletion of the IPv4 address space. The asymmetric addressing and connectivity regimes established by NATs, however, cause problems for many applications such as teleconferencing ([SIP], [H.323]) and multiplayer on-line gaming systems. Such application protocols require "peer-to-peer" communication directly between arbitrary hosts, and not just traditional "client/server" communication between a "client" host and a "well-known" server with a global IP address and DNS name.

RFC 3235 [NAT-APPL] already proposes NAT friendly design guidelines for applications, but merely recommends against using peer-to-peer communication and does not provide a workable solution to this problem. This document acts as an adjunct to [NAT-APPL], with focus on Peer-to-peer application design guidelines. The guidelines

in the document apply to Traditional NATs as described in RFC 2663 [NAT-TERM]. As such, the term NAT used throughput the document refers to Traditional NAT.

Given the increasing demand for applications that require P2P communication, in conjunction with the ubiquity of NATs, applications are increasingly implementing and deploying various workarounds to this problem. Most of the workarounds take the form of a NAT traversal or "hole punching" algorithm, by which two "peers" lying behind one or more NATs cooperate with a well-known "rendezvous server" to set up a direct peer-to-peer communication path between them. As pointed out in [UNSAF], application endpoints are fixed uniquely in the public realm with the aid of the rendezvous server. The rendezvous server is crucial to the initial path setup but does not take part in the subsequent peer-to-peer data stream.

There are many different NAT traversal algorithms already in use and currently being explored. However, due to the lack of standardization for NAT behavior up to this point, none of these algorithms can be guaranteed to work reliably over all currently deployed NATs. Further, without standardization of NAT traversal algorithms there is a strong danger that the proliferation of traversal algorithms may further compound the reliability and predictability problems that NAT created in the first place.

This document focuses exclusively on NAT traversal techniques that do not require the application to communicate explicitly with the NATs in the path. Protocols that allow applications to obtain external communication endpoints through explicit interaction with NATs in the path are outside the scope of this document. Several such protocols exist and are documented elsewhere ([SOCKS], [RSIP], [MIDCOM], [UPNP]), but so far none of these protocols have become widely accepted.

This document defines a set of best current practices for implementing NAT traversal in applications. The specific

recommendations are described at length in the sections 2 through 5 and later summarized concisely in Section 6.

2. Terminology and Conventions Used

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, the IP addresses 192.0.2.1, 192.0.2.128, and 192.0.2.254 are used as example IP addresses [RFC3330]. Although

these addresses are all from the same /24 network, this is a limitation of the example addresses available in [RFC3330]. In practice, these addresses would be on different networks

3. BEHAVE-compliant versus Legacy NATs

BEHAVE-compliant NATs are those NAT devices that conform to the behavioral requirements set out in [BEH-TCP], [BEH-UDP], [BEH-ICMP], and other protocol specific behave document(s) in the future which define requirements for NATs when handling protocol specific traffic.

The NAT traversal techniques described in this document are known to work in practice with a variety of existing NATs in the most common interconnection scenarios.

To be considered "BEHAVE-compliant", an application MUST be designed to operate reliably when all NATs in its communication paths are BEHAVE-compliant. It is also RECOMMENDED that new applications assume that all NATs in the application path are BEHAVE-compliant, since non-BEHAVE-compliant NATs are expected to be deprecated quickly. Adding complexity to applications for the purpose of handling legacy NATs risks introducing additional unpredictability into the network.

This document does not specifically prohibit applications from implementing more elaborate NAT traversal algorithms that may function over a wider variety of non-BEHAVE-compliant, "legacy"

NATs. Some known techniques for operating over such poorly-behaved NATs are outlined briefly in [P2P-STATE], and are described more thoroughly in [NUTSS], [P2PNAT], [NATBLAST], and [NATTRAV]. Applications implementing fancier protocols such as these, however, must ensure that their traversal algorithms operate just as efficiently as the ones specified here over BEHAVE-compliant NATs, and do not create new security vulnerabilities or unnecessarily burden network components in the path.

REQ-1 Applications MUST be designed to operate reliably over BEHAVE-compliant NATs. New applications are RECOMMENDED to assume that all NATs in the path are BEHAVE-compliant.

4. General NAT Traversal Concepts

This section describes requirements and techniques for NAT traversal that are independent of transport protocol; subsequent sections will specifically address NAT traversal for the UDP and TCP transport protocols. For more detailed background information on current

practices in use by existing applications, please refer to the companion document [P2P-STATE].

4.1. NAT Functions Influencing Traversal Logic

Traditional NATs ([NAT-TRAD]) are the most commonly deployed NATs. These NATs integrate two logical functions, each of which interferes with peer-to-peer communication in a different way and thus requires NAT traversal support in applications.

Address Translation:

A NAT modifies the IP-level and often the transport-level header information in packets flowing across the boundary, in order to enable many "private" hosts behind the NAT to share the use of a smaller number of public IP addresses (often just one). Hosts behind the NAT usually have no unique, permanently usable address on the public Internet, and can only communicate through temporary public endpoints that the NAT assigns them dynamically as a result of communication attempts initiated by hosts on the private network.

When two hosts reside on two different private networks behind distinct NATs, neither of them has a permanent address that the other can reach at any time, so in order to establish peer-to-peer connections the hosts must rely on the temporary public endpoints their NATs assign them as a result of prior outgoing client/server style communication sessions. Discovering, exchanging, and using these temporary public endpoints generally requires that the two hosts first collaborate through a well-known server on the public Internet that both hosts can reach, as described below.

Filtering of Unsolicited Traffic:

The filtering function in a Traditional NAT device restricts communication between a private internal network and the public Internet by dropping incoming sessions that are deemed "unsolicited". All packets arriving from the public Internet are dropped unless they are part of an existing communication session that was previously initiated by a host on the private network.

When two hosts reside on two different private networks behind distinct NATs, an attempt by either host to initiate a peer-to-peer connection to the other will usually fail, even if the connection attempt is directed to the correct temporary public endpoint assigned by the opposite host's NAT, because the opposite

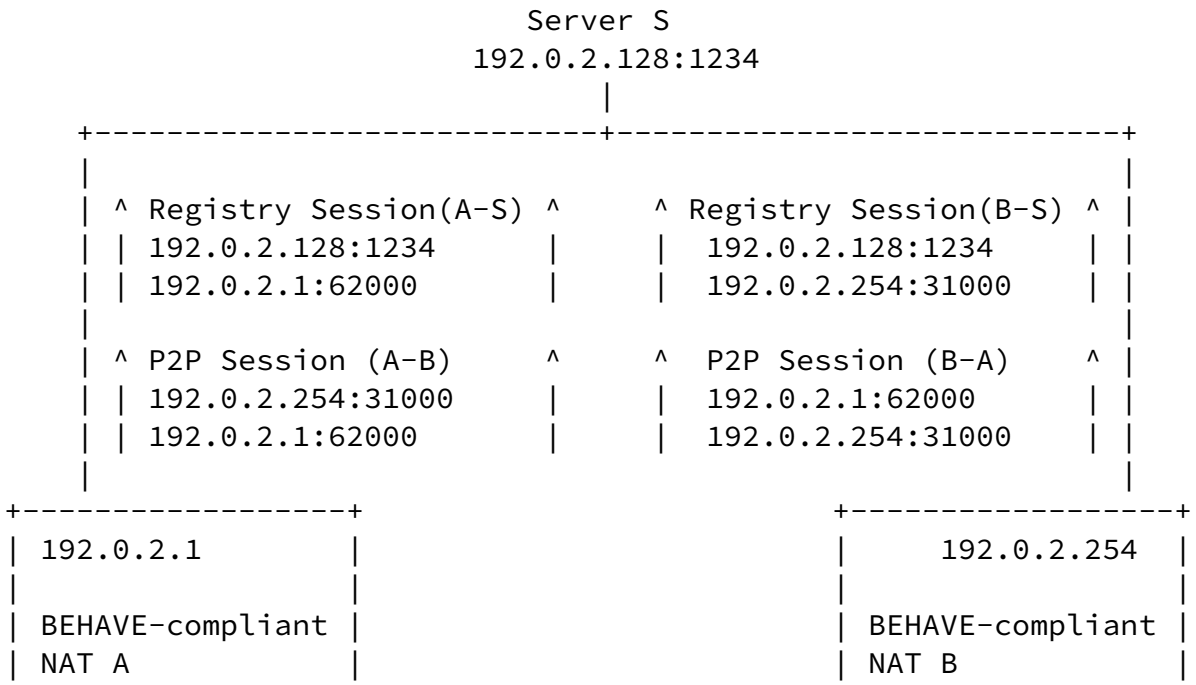
host's NATs will interpret this attempt as unsolicited incoming traffic and reject it. NAT traversal in this case requires the two hosts to cooperate, typically by communicating initially through a well-known server on the public Internet that they can both reach, so as to make their peer-to-peer connection appear to each host's NAT as if it was initiated from within that host's own private network.

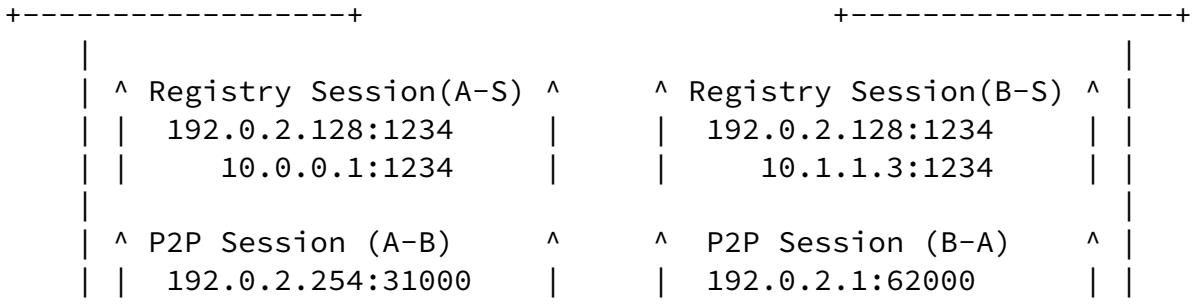
The cooperation of two hosts to create a peer-to-peer connection across NATs does not constitute a violation of the filtering policy imposed by the NAT. Network firewall functionality in general is outside the scope of this document, and this document does not condone any attempts by application developers to subvert security policies that may be imposed by NATs or firewalls.

4.2. Communication Between Peers Behind Distinct NATs

Although the details of NAT traversal vary from one transport protocol to another depending on how NATs recognize and handle sessions for that transport, the basic approach to NAT traversal is transport-independent. We merely assume for now that each transport uses session endpoints consisting of an (IP address, port number) pair to identify and differentiate communication sessions, and that each communication session is uniquely identified by its two endpoints. We focus here specifically on the one NAT traversal algorithm recommended here for new applications.

Suppose client hosts A and B both have private IP addresses and lie behind different NATs, as shown below.





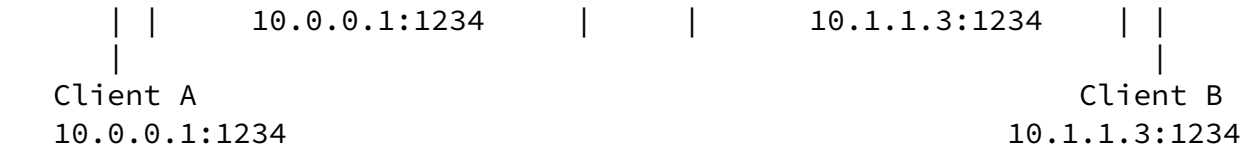


Figure 1: Simultaneous outgoing sessions to accomplish direct-P2P

A peer-to-peer application running on clients A and B, and also on a well-known rendezvous server S, each use port number 1234 at their own IP address to form their primary local communication endpoint. A and B have each initiated communication sessions from their local endpoint server S's endpoint. As a result of A's outgoing connection attempt to S, NAT A has dynamically assigned port 62000 at its own public IP address, 192.0.2.1, to A's session with S, so that S sees this session as having been initiated from the endpoint 192.0.2.1:62000, rather than from A's original private endpoint of 10.0.0.1:1234. Similarly, B's outgoing connection to S causes NAT B to assign port 31000 at its own IP address to B's session with S, thus forming B's public endpoint of 192.0.2.254:31000.

Now suppose that host A wants to establish a communication session directly with host B. If A just naively initiates a new communication session to the endpoint B believes itself to be using, namely 10.1.1.3:1234, then A's connection attempt will either reach the wrong host - a different host on A's own private network that happens to have the private IP address 10.1.1.3 - or it will reach no host at all, because B's private IP address 10.1.1.3 is not routable over the Internet.

Even if A learns B's temporary public endpoint, 192.0.2.254:31000, from server S, and attempts to initiate a communication session to that destination endpoint, NAT B may reject this attempt because the IP packet's source and destination endpoints do not match those of an existing session previously initiated from within the private network. The destination endpoint of A's connection attempt to B matches the source endpoint of B's existing session with S, but the source endpoint of A's connection attempt is of course different. Similarly, if B makes a unilateral connection attempt to A's public endpoint, NAT A may similarly reject B's attempt.

As it turns out, this difficulty could arise even if NAT-A and

NAT-B are replaced by firewalls with the standard filtering policy of rejecting unsolicited incoming communication attempts.

In order to operate reliably across NATs and firewalls that reject unsolicited incoming communication, the client hosts A and B collaborate with an external server S to learn each other's public AND private endpoints, and then each of the two client hosts initiate "approximately simultaneous" connection attempts from their existing primary local endpoints (the same local endpoints they used previously for the connection to S), and directed at all of the known endpoints (public and private) for the other host. In the scenario illustrated above, A's connection attempt to B's public endpoint is interpreted by NAT A as a legitimate, outgoing session whose private source endpoint (10.0.0.1:1234) is the same as that of A's existing session with S, but whose public destination endpoint (192.0.2.254:31000) is different. If NAT A is BEHAVE-compliant, it will translate A's private source endpoint for this new session in the same way that it did for A's existing session with S, so that the new session appears on the public Internet to be a session between A's public endpoint, 192.0.2.1:62000, and B's public endpoint, 192.0.2.254:31000.

In similar fashion, B's "approximately simultaneous" connection attempt from its private endpoint, 10.1.1.3:1234, to A's public endpoint, 192.0.2.1:62000, results in NAT B opening a new translation session that reuses the existing public endpoint for B, 192.0.2.254:31000, which NAT B previously assigned to B's session with S. NAT B is now set up to allow communication between A's public endpoint and B's private endpoint, and on the public Internet this session has the endpoints 192.0.2.1:62000 and 192.0.2.254:31000, the same as the endpoints of the session that A initiated above toward B's public endpoint. Both NATs are thus set up to permit communication between these two public endpoints, translating and forwarding the traffic comprising this session to the respective client hosts on the private networks as appropriate.

Applications wishing to establish peer-to-peer communication MUST support NAT traversal using the "approximate simultaneous" connection technique. The traversal technique relies on certain aspects of NAT behavior described fully in the companion documents [BEH-TCP], [BEH-UDP], and [BEH-ICMP]. The technique also relies on the transport protocol allowing a connection to be initiated actively by two endpoints, rather than asymmetrically in traditional client/server fashion. Fortunately both of the two ubiquitous transports, TCP and UDP, allow symmetric connection initiation in this way.

REQ-2 Applications wishing to establish peer-to-peer communication MUST support NAT traversal using the "approximate simultaneous" connection technique and using the help of a "rendezvous server" in the public network.

4.3. Short-Circuiting Sessions on Private Networks

Although the network topology illustrated in figure 1 is typical of the situation seen by P2P applications, it is by no means the only possible scenario. Only one of the client hosts may be behind, or one or more of the clients may be located behind two or more levels of NATs, any number of which may be shared between the two clients. The general NAT traversal algorithm described above will work reliably in all of the common topological scenarios provided that the NATs involved are BEHAVE-compliant. One other particularly common scenario is worth special consideration however. In the situation illustrated below the two clients (probably unknowingly) happen to reside behind the same NAT, and are therefore located in the same private IP address space.

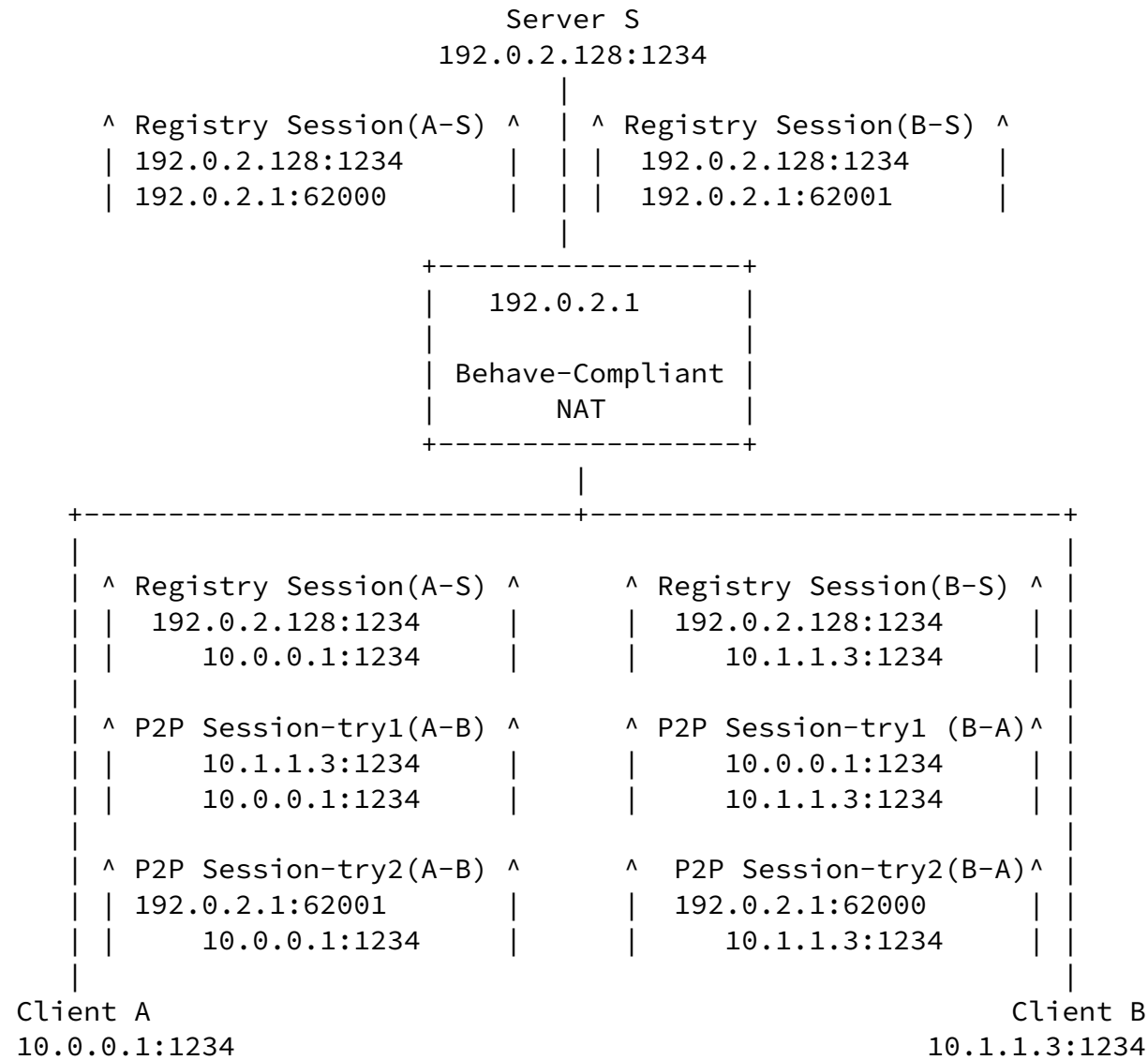


Figure 2: Register private identity & NAT identity with Relay server

In this scenario, client A has established a session with well-known server S as before, to which the common NAT has assigned public port number 62000. Client B has similarly established a session with S, to which the NAT has assigned public port number 62001. Suppose that

A and B use the NAT traversal technique outlined above to establish a communication channel using server S as an introducer. If A and B only attempt simultaneous connections to each other's public endpoints, 192.0.2.1:62001 and 192.0.2.1:62000 respectively, then their connection attempts will succeed only if the NAT supports hairpin translation, as described in [P2P-STATE] and [BEH-TOP]. Although hairpin translation is required for a NAT to be considered fully BEHAVE-compliant, this feature is not yet widely supported by commonly deployed NATs at the time of this writing.

Additionally, the resulting connection between A and B will be sub-optimal in this case because all traffic will unnecessarily pass through and be translated by the NAT, whereas the two endpoint hosts are perfectly capable of communicating directly on their common IP network without the NAT intervention.

To address this problem, P2P applications MUST exchange their local endpoints as known to themselves, in addition to the global endpoints they register with the rendezvous server. In case an application host has multiple IP addresses or is registered with multiple rendezvous servers, the P2P application SHOULD exchange all pertinent endpoints with its peers.

Further, P2P applications MUST be prepared to make "approximately simultaneous" connection attempts to all exchanged endpoints, including the private endpoints and the public endpoints of the desired peer. By doing this, a P2P application is able to use whichever connection succeeds first in establishing bi-directional communication between the correct peers. If the two end hosts happen to be located in the same private network, their connection attempt using each others' private endpoints is likely to succeed first because it follows a shorter network path not involving the NAT. If the NAT does not support hairpin translation, the connection attempts using the hosts' private endpoints will be the only one to succeed.

REQ-3 Applications implementing NAT traversal MUST exchange their local endpoints as known to themselves, in addition to the global endpoints they register with the rendezvous server. In case an application host has multiple IP addresses or is registered with multiple rendezvous servers, the P2P application SHOULD exchange all pertinent endpoints with

its peers. Further, peering applications MUST be prepared to make "approximately simultaneous" connection attempts to all exchanged endpoints of the desired peer.

4.4. Authenticating Peer-to-Peer Connections

It is extremely important not only for security but also for general robustness that applications implementing a NAT traversal protocol authenticate any peer-to-peer connections they establish, using some higher-level application-specific notion of host or user identity. To operate reliably and securely, applications must consider any IP addresses and port numbers they use for communication with other hosts to be merely "locators" for hosts, serving as hints indicating how the desired host might be reached, and not as a reliable "identifier" for the target host or user.

In particular, applications must not merely assume that the first communication attempt that establishes transport-level connectivity and elicits a response from a particular target endpoint (IP address and port number) necessarily represents a connection to the desired host. Consider the following topological scenario, for example, which is in fact extremely common in today's Internet.

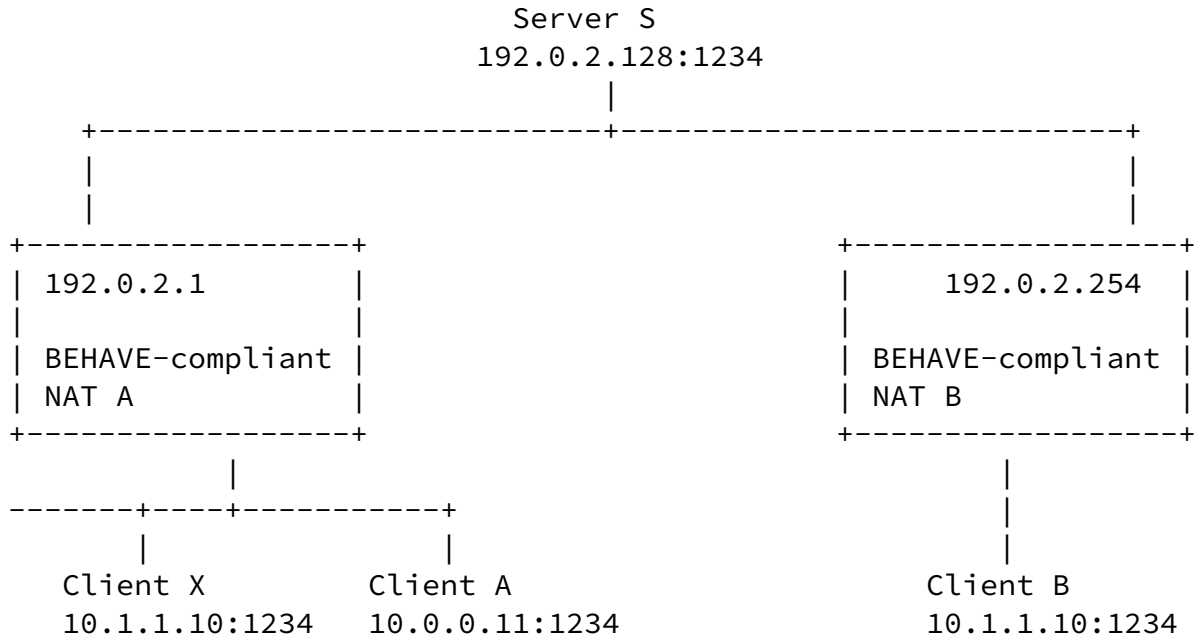


Figure 3: Clients behind different NATs can bear same local endpoint

In this scenario, suppose that NAT A and NAT B are both "off-the-shelf" consumer NAT routers from the same vendor, which the vendor has configured by default to act as DHCP servers that hand out private IP addresses starting at 10.1.1.10. (Most users of such devices know little or nothing about IP addresses, and therefore are very unlikely to reconfigure their NATs any more than is necessary to get them to connect to the Internet.) As before, Client A wishes to establish a peer-to-peer connection with Client B with the help of Server S. Client A happened to receive private IP address 10.1.1.11 on NAT A's private network, after Client X had already been assigned private IP address 10.1.1.10. Client B happens to be the only host on NAT B's private network, and thus received the first available private IP address, 10.1.1.10. Client X happens to be running the same P2P application as is running on clients A and B, and thus has port 1234 allocated and ready to initiate and accept peer-to-peer connections.

Suppose Client A follows the NAT traversal approach described above to establish a peer-to-peer session with Client B. As per the suggested protocol, A and B each make approximately simultaneous

connection attempts both to each other's public and private endpoints. B's connection attempt to A's private endpoint, 10.1.1.11:1234, will of course fail because there is no host 10.1.1.11 on NAT B's private network and that IP address is not globally routable. A's connection attempt to B's public endpoint and B's connection attempt to A's public endpoint will eventually succeed in establishing the desired peer-to-peer connection if the two NATs are BEHAVE-compliant. However, A's connection attempt to B's private endpoint, 10.1.1.10:1234, will succeed at the transport layer but connect to the wrong host: namely client X, the host on NAT A's private network that happens to have the same private IP address as B does on NAT B's network. Furthermore, this bogus connection to client X is likely to succeed much more quickly than the actually desired connection to client B, because X is on the same private network as A. If the application running on client A does not properly authenticate its peer-to-peer connections using some higher-level notion of identity that is independent of IP address, then client A is likely to assume that its transport-level connection to X

is the desired peer-to-peer connection, cancel its attempt to connect to B's public endpoint, and subsequently become very confused when the peer it connected to fails to behave like client B.

Given the prevalence of NAT routers that are pre-configured by their vendors to hand out private IP addresses via DHCP in more-or-less deterministic fashion from a standard private IP address block, different hosts on different private networks are very likely to have the same private IP addresses, making the above scenario extremely likely for P2P applications to encounter. P2P applications therefore MUST authenticate their transport-layer connections using a higher level application-specific notion of identity, before concluding they have successfully connected to the desired host. Strong cryptographic authentication using standard algorithms is of course preferred.

REQ-4 P2P applications MUST authenticate their transport-layer connections using a higher level application-specific notion of identity, before concluding they have successfully connected to the desired host.

4.5. NAT Behavior Detection

In many existing NAT traversal protocols for both TCP and UDP, each client attempts to determine experimentally certain properties of any NATs it is located behind before attempting to establish peer-to-peer connections with other clients. For example, even when a NAT does not re-use the same public endpoint for all sessions involving a given private endpoint as required for BEHAVE compliance, it is sometimes possible to predict which port the NAT will assign to a

new session.

Extensive testing of various existing NATs, however, has revealed that there is no truly robust way a client can predict how a legacy NAT will behave in the future based on such experimental tests. Some legacy NATs behave differently depending on the local port number the application is using on the client, and can even switch behaviors dynamically depending on unpredictable timing and network conditions. Therefore, applications SHOULD NOT attempt to predict the future behavior of NATs in the path through empirical tests. If they do use such experimental tests in an attempt to make peer-to-peer

connections work across a wider variety of legacy NATs, they MUST ensure that such methods do not delay or otherwise impede the the performance or reliability of the application over BEHAVE-compliant NATs.

REQ-5 Applications SHOULD NOT attempt to predict the future behavior of NATs in the path through empirical tests. If they do, applications MUST ensure that any such tests do not delay or otherwise impede the performance or reliability of NAT traversal over BEHAVE-compliant NATs.

5. NAT Traversal for UDP

NAT traversal for UDP, also commonly known as UDP "hole punching", was mentioned briefly in section 5.1 of RFC 3027 [NAT-PROT], and first publicly documented informally on the Internet [KEGEL]. Because of UDP's simplicity and its connectionless nature, NAT traversal for UDP is somewhat simpler, more widely understood, and hence more universally supported by NATs and applications than is NAT traversal for TCP, though the principles are the same for both transports. NAT traversal for UDP has been used in several recent experimental Internet protocols [TEREDO], [ICE] along with various proprietary or non-standardized protocols. The NAT traversal approach recommended in this document is also described informally in [P2PNAT], and other variations of hole punching are explored more thoroughly in other recent research papers [NUTSS], [NATBLAST], and [NATTRAV].

To set up a peer-to-peer UDP session between two clients A and B, we assume that the clients have each bound to a particular primary local UDP port, and that the clients have each initiated a UDP session from this primary local port to a well-known rendezvous server S, as described earlier. Each client then learns the other's public and private UDP endpoints from the server S, and simply begins sending UDP datagrams, from their respective primary local ports (the same ports they used to contact S), to all of the other client's known endpoints. If one or both of the clients is behind a BEHAVE-

compliant NAT, the outgoing datagrams from each client will "open a hole" through a firewall or establish a translation session through the NAT, causing the NAT to forward subsequent incoming datagrams from the opposite client as desired.

5.1. UDP Idle Timeouts

Because of its inherently connectionless nature, NATs have no fully reliable way to determine when a UDP communication session crossing the NAT has terminated, other than simply by assuming the session is over if it observes a sufficiently long idle period. Applications whose UDP communication sessions may experience long idle periods must therefore account for this idle timeout.

As specified in [BEH-UDP], any BEHAVE-compliant NAT is required to have an idle timeout of at least two minutes, but idle timeouts as small as 30 seconds have been observed in existing NATs. Additionally, BEHAVE-compliant NATs are only required to reset the idle timer on the observance of outgoing traffic leaving the private network; the NAT may ignore incoming traffic for this purpose, in order to prevent external hosts from being able to hold UDP sessions open unilaterally and thus consume NAT resources indefinitely. BEHAVE-compliant NATs are required to support Address and Port Dependent filtering Behavior, which essentially resets the idle timer for each session whenever outbound traffic is seen for that session. A NAT's UDP idle timeouts affects P2P applications implementing NAT traversal in two main ways:

Rendezvous Server Registration Sessions:

Client hosts implementing UDP hole punching typically register with one or more well-known rendezvous servers, S in the above scenarios, and expect to be notified by S when a second client wishes to open a peer-to-peer connection to the first. However, if a NAT's UDP idle timer times out while the first client is waiting for incoming connections, then the client will not receive the notification from S of the second client's desire to connect. The client therefore SHOULD send periodic outbound "keep-alive" packets to the rendezvous server(s) in order to ensure that the registration session remains open while the application is active. If a UDP application maintains active registration sessions with more than one well-known rendezvous server simultaneously, then the application SHOULD send outbound keep-alive packets periodically to each of the rendezvous servers it is registered with. The periodicity is at least once within the BEHAVE-compliant NAT UDP timeout [BEH-UDP].

If a UDP application merely desires to be compatible with BEHAVE-

compliant NATs, then its outbound keep-alive packets need not elicit a response from the server unless the application is concerned about detecting if the server disappears.

REQ-6 Applications wishing to accept connections from other peers after registering via UDP with one or more rendezvous servers SHOULD send periodic outgoing UDP "keep-alive" packets to each of the rendezvous servers, at least once within the BEHAVE-compliant NAT UDP timeout [BEH-UDP] in order to ensure that the registration session remains open while the application is active.

Peer-to-Peer Sessions:

Once two client hosts have used a rendezvous server to set up a peer-to-peer UDP communication session between them, this peer-to-peer session is similarly vulnerable to being closed by any of the NATs along the path if it goes idle for too long.

If an application has only a few peer-to-peer sessions active at once, then the application SHOULD use keep-alives for each of the active peering sessions to keep the sessions open. If an application has many idle peer-to-peer sessions at once, then the application SHOUL NOT use keep-alives on peer-to-peer sessions so the network is not flooded with keep-alives. Instead, the application SHOULD be prepared to re-establish peer-to-peer sessions as needed after an idle period, by simply re-running the NAT traversal protocol via the original rendezvous server.

REQ-7 An Application SHOULD use the following guidelines with regard to its UDP peer-to-peer sessions.
a) If the application has only a small number of peer-to-peer sessions active at once, then send periodic outgoing UDP "keep-alive" packets to each active peer at least once within the BEHAVE-compliant NAT UDP timeout [BEH-UDP].
b) If the application has many peer-to-peer sessions active at once, then do not send periodic "keep-alive" packets to peers so the network is not flooded with keep-alives.
c) If the application has a peer-to-peer UDP session that may go idle for more than the BEHAVE-compliant NAT UDP timeout at a time without a keep-alive, and the session connectivity is detected to have been lost, then be prepared to re-run the original NAT traversal protocol to re-establish the peer-to-peer session.

6. NAT Traversal for TCP

NAT traversal for TCP, or "TCP hole punching," is not yet as well-

understood or widely supported as is UDP hole punching. Nevertheless, the general technique described in section 2 above works for TCP as well as UDP, as long as all NATs in the path are well-behaved. The recommended NAT traversal algorithm for TCP, described here, makes use of the symmetric TCP connection initiation feature of TCP as specified in RFC 793 [TCP] and RFC 1122 [RFC1122]. This algorithm is guaranteed to work reliably as long as all NATs in the path are BEHAVE-compliant [BEH-TCP], and as long as the end-hosts correctly implement the TCP protocol.

Other more complex TCP hole punching algorithms have been developed and explored elsewhere in [NUTSS], [NATBLAST], and [NATTRAV]. These algorithms use various tricks to work around the nonstandard behaviors of many existing NATs, and/or to work around bugs in the TCP implementations of certain existing operating systems. Applications MAY implement more complex algorithms such as these in order to achieve broader compatibility with existing NATs and hosts, but applications MUST ensure that any such alternative algorithm still works reliably and efficiently over BEHAVE-compliant NATs without substantially burdening the network and any NATs on the path.

To prepare for TCP NAT traversal, a P2P client application first binds to an arbitrary local port, which becomes the application's primary local port. The Application SHOULD use the port to simultaneously listen for incoming peer-to-peer connections and to initiate outgoing connections to rendezvous servers and other peers. Because standard sockets APIs usually associate TCP sockets with individual TCP sessions rather than with a local port as with UDP, the application must typically open multiple TCP sockets - one listen socket and one or more connect-sockets - and explicitly bind them to the same local port, using a special socket option usually named SO_REUSEADDR or SO_REUSEPORT.

Once a TCP application has bound to its primary local port, started listening on it, and opened connections to one or more rendezvous servers, the application SHOULD use "approximately simultaneous" connection technique to initiate outgoing connections or to accept incoming connections. Each peer SHOULD use the "approximate simultaneous" connection technique to connect to all of the known

endpoints (including original and translated) of its peer. For example, say two clients, A and B, wish to establish a peer-to-peer connection with the help of a common rendezvous server S. They first exchange their public and private TCP endpoints through S as described in section 2. Each client then simultaneously attempts to initiate outgoing TCP connections from its primary local port to each of the opposite client's known TCP endpoints (public and private). As long as all NATs in the path are well-behaved, each

client's outgoing TCP connection attempt will open firewall and/or translation sessions through any NATs it is located behind, eventually resulting in a working bi-directional TCP connection through all intervening NATs on the path, in the same way as for UDP.

Because of timing dependencies and differences in TCP implementations, applications may observe slightly different (but functionally equivalent) results when a P2P connection is successfully established using this method. If client B is not actually located behind a firewall or NAT, for example, and client A's first attempt to connect directly to B reaches B before its peer-to-peer connection request relayed through S reaches B, then B will accept A's connection via its outstanding listen socket, in traditional client/server fashion. Even if A's connection request (SYN packet) to B crosses B's corresponding request to A, resulting in a TCP simultaneous open at the protocol level, some end-host operating systems may still "deliver" the resulting connection to the application via the application's outstanding listen socket for its primary local port, rather than via the socket by which the application explicitly initiated a connection to the opposite client. The application must be prepared to handle all such possible cases gracefully.

Applications MAY alternatively establish peer-to-peer TCP connections via other, asymmetric methods if one or both endpoint hosts do not correctly support simultaneous TCP open.

REQ-8 Applications implementing peer-to-peer communication via TCP SHOULD simultaneously listen for incoming peer-to-peer connections and open connections to rendezvous servers and other peers from the same endpoint.

REQ-9 Applications SHOULD establish peer-to-peer TCP connections by

making "approximately simultaneous" connection attempts from each peer to all of the known endpoints (including original and translated) for its peer.
Applications MAY alternatively establish peer-to-peer TCP connections via other, asymmetric methods if one or both endpoint hosts do not correctly support simultaneous TCP open.

6.1. Ensuring Robustness

Some existing NATs actively reject an apparently-unsolicited incoming TCP connection by sending back TCP RST or ICMP error packets to the connection initiator, rather than simply dropping the incoming SYN. This behavior can cause one of the clients to observe bogus timing-dependent connection failures. While this NAT behavior is

deprecated and not allowed for BEHAVE-compliant NATs, P2P applications can easily make themselves robust against this behavior. If a client's attempt to initiate a peer-to-peer connection fails with a "Connection Refused" or "Network Unreachable" or similar network-related error before some application-defined peer-to-peer connection timeout has expired, the application SHOULD simply retry the same outgoing connection attempt. However, the application MUST NOT retry more frequently than once per second. Doing so avoids accidental flooding of the network with SYNs if the cause of the error is close to the client and is thus reported very quickly after each attempt.

REQ-10 Applications SHOULD re-try peer-to-peer TCP connection attempts that fail due to network conditions other than timeout, but MUST NOT re-try connecting to a given peer more than once per second.

7. Summary of Requirements

An application that supports all of the mandatory requirements of this specification (the "MUST" requirements), is "compliant with this specification" or "BEHAVE-compliant". An application that supports all of the mandatory and optional recommendations of this specification (including the "SHOULD" or "RECOMMENDED" ones) is "fully compliant with all the mandatory and recommended

requirements of this specification."

REQ-1 Applications MUST be designed to operate reliably over BEHAVE-compliant NATs. New applications are RECOMMENDED to assume that all NATs in the path are BEHAVE-compliant.

REQ-2 Applications wishing to establish peer-to-peer communication MUST support NAT traversal using the "approximate simultaneous" connection technique and using the help of a "rendezvous server" in the public network.

REQ-3 Applications implementing NAT traversal MUST exchange their local endpoints as known to themselves, in addition to the global endpoints they register with the rendezvous server. In case an application host has multiple IP addresses or is registered with multiple rendezvous servers, the P2P application SHOULD exchange all pertinent endpoints with its peers. Further, peering applications MUST be prepared to make "approximately simultaneous" connection attempts to all exchanged endpoints of the desired peer.

REQ-4 P2P applications MUST authenticate their transport-layer

connections using a higher level application-specific notion of identity, before concluding they have successfully connected to the desired host.

REQ-5 Applications SHOULD NOT attempt to predict the future behavior of NATs in the path through empirical tests. If they do, applications MUST ensure that any such tests do not delay or otherwise impede the performance or reliability of NAT traversal over BEHAVE-compliant NATs.

REQ-6 Applications wishing to accept connections from other peers after registering via UDP with one or more rendezvous servers SHOULD send periodic outgoing UDP "keep-alive" packets to each of the rendezvous servers, at least once within the BEHAVE-compliant NAT UDP timeout [BEH-UDP] in order to ensure that the registration session remains open while the application is active.

REQ-7 An Application SHOULD use the following guidelines with regard

to its UDP peer-to-peer sessions.
a) If the application has only a small number of peer-to-peer sessions active at once, then send periodic outgoing UDP "keep-alive" packets to each active peer at least once within the BEHAVE-compliant NAT UDP timeout [BEH-UDP].
b) If the application has many peer-to-peer sessions active at once, then do not send periodic "keep-alive" packets to peers so the network is not flooded with keep-alives.
c) If the application has a peer-to-peer UDP session that may go idle for more than the BEHAVE-compliant NAT UDP timeout at a time without a keep-alive, and the session connectivity is detected to have been lost, then be prepared to re-run the original NAT traversal protocol to re-establish the peer-to-peer session.

REQ-8 Applications implementing peer-to-peer communication via TCP SHOULD simultaneously listen for incoming peer-to-peer connections and open connections to rendezvous servers and other peers from the same endpoint.

REQ-9 Applications SHOULD establish peer-to-peer TCP connections by making "approximately simultaneous" connection attempts from each peer to all of the known endpoints (including original and translated) for its peer.
Applications MAY alternatively establish peer-to-peer TCP connections via other, asymmetric methods if one or both endpoint hosts do not correctly support simultaneous TCP open.

REQ-10 Applications SHOULD re-try peer-to-peer TCP connection

attempts that fail due to network conditions other than timeout, but MUST NOT re-try connecting to a given peer more than once per second.

8. Security Considerations

This document does not inherently create new security issues. This section describes security risks the applications could inadvertently create in attempting to support P2P communication across NAT devices.

8.1. Denial-of-service attacks

P2P applications and the public registry servers that support them must protect themselves against denial-of-service attacks, and ensure that they cannot be used by an attacker to mount denial-of-service attacks against other targets. To protect themselves, P2P applications and registry servers must avoid taking any action requiring significant local processing or storage resources until authenticated two-way communication is established. To avoid being used as a tool for denial-of-service attacks, P2P applications and servers must minimize the amount and rate of traffic they send to any newly-discovered IP address until after authenticated two-way communication is established with the intended target.

For example, P2P applications that register with a public rendezvous server can claim to have any private IP address, or perhaps multiple IP addresses. A well-connected host or group of hosts that can collectively attract a substantial volume of P2P connection attempts (e.g., by offering to serve popular content) could mount a denial-of-service attack on a target host C simply by including C's IP address in their own list of IP addresses they register with the rendezvous server. There is no way the rendezvous server can verify the IP addresses, since they could well be legitimate private network addresses useful to other hosts for establishing network-local communication. The P2P application protocol must therefore be designed to size- and rate-limit traffic to unverified IP addresses in order to avoid the potential damage such a concentration effect could cause.

8.2. Man-in-the-middle attacks

Any network device on the path between a P2P client and a rendezvous server can mount a variety of man-in-the-middle attacks by pretending to be a NAT. For example, suppose host A attempts to register with rendezvous server S, but a network-snooping attacker is able to observe this registration

request. The attacker could then flood server S with requests that are identical to the client's original request except with a modified source IP address, such as the IP address of the attacker itself. If the attacker can convince the server to register the client using the attacker's IP address, then the

attacker can make itself an active component on the path of all future traffic from the server AND other P2P hosts to the original client, even if the attacker was originally only able to snoop the path from the client to the server.

The client cannot protect itself from this attack by authenticating its source IP address to the rendezvous server, because in order to be NAT-friendly the application must allow intervening NATs to change the source address silently. This appears to be an inherent security weakness of the NAT paradigm. The only defense against such an attack is for the client to authenticate and potentially encrypt the actual content of its communication using appropriate higher-level identities, so that the interposed attacker is not able to take advantage of its position. Even if all application-level communication is authenticated and encrypted, however, this attack could still be used as a traffic analysis tool for observing who the client is communicating with.

9. IANA Considerations

There are no IANA considerations.

10. Normative References

[BEH-ICMP] Srisuresh, P., Ford, B., Sivakumar, S., and Guha, S., "NAT Behavioral Requirements for ICMP protocol", draft-ietf-behave-nat-icmp-03.txt (Work In Progress), March 2007.

[BEH-TCP] Guha, S., Biswas, K., Ford, B., Francis, P., Sivakumar, S., and Srisuresh, P., "NAT Behavioral Requirements for Unicast TCP", draft-ietf-behave-tcp-00.txt (Work In Progress), February 2006.

[BEH-UDP] F. Audet and C. Jennings, "NAT Behavioral Requirements for Unicast UDP", RFC 4787, January 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

11. Informative References

[BEH-TOP] Srisuresh, P., and Ford, B., "Complications from Network Address Translator Deployment Topologies", draft-ford-behave-top-02.txt (Work In Progress), July 2006.

[H.323] "Packet-based Multimedia Communications Systems", ITU-T Recommendation H.323, July 2003.

[ICE] Rosenberg, J. "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", draft-ietf-mmusic-ice-09.txt (work in Progress), June 2006.

[KEGEL] Dan Kegel, "NAT and Peer-to-Peer Networking", July 1999. http://www.alumni.caltech.edu/~dank/peer-nat.html

[MIDCOM] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., and Rayhan, A., "Middlebox communication architecture and framework", RFC 3303, August 2002.

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

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

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

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

[NATBLAST] Biggadike, A., Ferullo, D., Wilson, G. and Perrig, A., "NATBLASTER: Establishing TCP Connections Between Hosts Behind NATs", ACM SIGCOMM Asia Workshop, April 2005.

[NUTSS] Guha, S., Takeday Y., and Francis, P., "NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity", SIGCOMM 2004 Workshops, August 2004.

[NATTRAV] Eppinger, J.L., "TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem", Carnegie

Mellon Tech Report CMU-ISRI-05-104, January 2005.

[P2PNAT] Ford, B., Srisuresh, P., and Kegel, D., "Peer-to-Peer Communication Across Network Address Translators", USENIX Annual Technical Conference, April 2005.

[P2P-STATE] Srisuresh, P., Ford, B., and Kegel, D., "State of Peer-to-Peer (P2P) communication across Network Address Translators (NATs)", draft-ietf-behave-p2p-state-02.txt (Work In Progress), February 2007.

[RFC1122] Braden, R., Editor, "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC3330] IANA, "Special-Use IPv4 Addresses", RFC 3330, September 2002.

[RSIP] Borella, M., Lo, J., Grabelsky, D., and Montenegro, G., "Realm Specific IP: Framework", RFC 3102, October 2001.

[SIP] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[SOCKS] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and Jones, L., "SOCKS Protocol Version 5", RFC 1928, March 1996.

[TCP] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[TEREDO] Huitema, C., "Teredo: Tunneling IPv6 over UDP through NATs", draft-ietf-ngtrans-shipworm-08.txt (Work In Progress), September 2002.

[UPNP] UPnP Forum, "Internet Gateway Device (IGD) Standardized Device Control Protocol V 1.0", November 2001. http://www.upnp.org/standardizeddcps/igd.asp

[UNSAF] Daigle, L., and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address

Translation", RFC 3424, November 2002.

Authors' Addresses:

Bryan Ford
Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology
77 Massachusetts Ave.
Cambridge, MA 02139
U.S.A.
Phone: (617) 253-5261
E-mail: baford@mit.edu
Web: http://www.brynosaurus.com/

Pyda Srisuresh
Kazeon Systems, Inc.
1161 San Antonio Rd.
Mountain View, CA 94043
U.S.A.
Phone: (408)836-4773
E-mail: srisuresh@yahoo.com

Dan Kegel
Kegel.com
901 S. Sycamore Ave.
Los Angeles, CA 90036
Phone: (323) 931-6717
E-mail: dank@kegel.com
Web: http://www.kegel.com/

Intellectual Property Statement

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the IETF Trust.