# NAT Check

## *Version 3, with UDP and TCP support*

## Check Your Network Address Translator for Compatibility with Peer-to-Peer Protocols

**NAT Check by Bryan Ford, web magic by Dave Andersen**

**Hosted by the MIDCOM-P2P project on SourceForge**

***This page, the NAT servers, and the results database are no longer being actively maintained.*** (Sorry, just no time; I have to focus on other projects now!) For a detailed overview of P2P hole punching techniques for both TCP and UDP, as well as some compatibility statistics generated using NAT Check, please see my USENIX '05 paper:

> **Peer-to-Peer Communication Across Network Address Translators**, Bryan Ford, Pyda Srisuresh, and Dan Kegel. USENIX, April 2005. PDF PS HTML. (BibTeX)

You can still find the NAT Check code below, but to use it you'll have to run your own set of servers and modify the code to point to those servers. Note that the "natbouncer" program the page incorrectly mentioned for a while is actually just part of the "natserver" program now: just run "natserver 1" and "natserver 2" on the primary servers, and run "natserver 3" on the third ("bouncer") server .

You may also wish to check out the following related papers by others:

- NATBLASTER: Establishing TCP Connections Between Hosts Behind NATs, Andrew Biggadike, Daniel Ferullo, Geoffrey Wilson, Adrian Perrig. ACM SIGCOMM ASIA Workshop, April 2005.
- Characterization and Measurement of TCP Traversal through NATs and Firewalls, Saikat Guha and Paul Francis. Interet Measurement Conference (IMC), Oct 2005.

---

*(old page starts here)*

# Introduction

If you are accessing the Internet from behind a Network Address Translator (NAT) of some kind, I would appreciate your help in surveying the behavior of different NATs, in terms of how and whether they support a certain technique for enabling peer-to-peer communication between NATted hosts (particularly when *both* endpoints are behind NATs). See below for a brief technical summary of the technique and the properties of a NAT tested by NAT Check. For an extended discussion of this technique and related issues, please read my Internet Draft on the topic. (Comments and feedback are greatly appreciated!) **Note:** this draft does NOT yet include a discussion of the new TCP-based NAT P2P hole punching technique that version 3 of NAT Check tests for. A new version of teh Internet-Draft should be available shortly.

To participate, please download, compile, and run this trivial C program on your system: natcheck.c. (If you

are on a 64-bit machine, then either add "-DHAVE_STDINT" to your compile command line or edit the #define's toward the beginning appropriately.) Alternatively, if you are the trusting type, the following pre-compiled binaries are available:

- Linux-x86, md5sum 5b4b7c1f5e7f37c66ff4500431bfffcd.
- FreeBSD-x86, md5sum 7daef2cd6ec398d3d025aa61e270a383.
- Win32-x86, md5sum 1cf715d6d8d4d4f168d34b9c4421c627. (Thanks to Philippe Verdy for this port!)

Still at version 2, needs updating:

- Mac OS X, md5sum 8dc2d9e814909e319be9d7e63000485b. (Thanks to Richard Elmore for this port!) GUI by Dustin Sallings

Then run the **natcheck** program and submit the results here. You can view the accumulated results at this page. *(Sorry, these no longer work - I'll try to recover and post a final snapshot of the database when I can.)*

Thank you very much for your help!

- Bryan

---

# Technical Explanation

This technique is fairly simple, has been pointed out before (e.g., here and here), and has been implemented in a variety of on-line games and other peer-to-peer applications. The purpose of this page is mainly to draw more attention to the technique and to discover how widely compatible the technique is with current NATs.

## The Technique

Suppose there are three communicating hosts: A, B, and C. Host A is a "well-known" Internet server with a permanent IP address, which acts as an "introducer" for the other two nodes. (For example, Host A might be a well-known ultrapeer or a game catalog server of some kind.) Host B, using Host A's "introduction" services, would like to establish a direct peer-to-peer connection with host C. Both B and C, however, are behind (probably different) network address/port translators, and neither of them has exclusive use of any public IP address.

To initiate a peer-to-peer connection with host C, host B first sends A a message requesting an "introduction" to host C. A sends B a reply message containing C's IP address and UDP port number *as reported by host C*, in addition to C's IP address and UDP port number *as observed by A*. (If C is behind a NAT, then these two address/port combinations will be different.) At the same time, host A sends host C a message containing B's IP address and UDP port numbers - again, both the ones reported by B and the ones observed by A, which will be different if B is behind a NAT.

Now B and C each know that they want to initiate a connection with each other, and they know each other's public (NATted) as well as original IP addresses and UDP port numbers. Both B and C now start attempting to send UDP messages directly to each other, at each of the available addresses. If B and C happen to be behind *the same* NAT, then they will be able to communicate with each other directly using their "originally reported" IP addresses and UDP port numbers.

In the more common case where B and C are behind *different* NATs, the "originally reported" addresses will

be useless because they will both be private IP addresses in different addressing domains. Instead, the IP address/UDP port combinations observed by A can be used in this case to establish direct communication. Although B's NAT will initially filter out any UDP packets arriving from C's public (NATted) UDP port directed at B's public port, the first UDP message B sends to C will cause B's NAT to open up a new UDP session keyed on C's public port, allowing future incoming traffic from C to pass through the NAT to B. Similarly, the first few messages from B to C may be filtered out by C's NAT, but will be able to start passing through the firewall as soon as C's first message to B causes C's NAT to open up a new session. In this way, each NAT is tricked into thinking that *its* respective internal host is the "initiator" of this new session, when in fact the session is fully symmetrical and was initiated (with A's help) simultaneously in each direction.

## Required NAT Behavior

There is one important requirement that the NATs must satisfy in order for this technique to work: the NATs must be designed so that they assign only one (public IP address, public UDP port) pair to each (internal IP address, internal UDP port) combination, rather than allocating and assigning a new public UDP port for each new UDP *session*. Recall that a "session" in Internet terminology is defined by the IP addresses and port numbers of *both* communicating endpoints, so host B's communication with host A is considered to be one session while host B's communication with host C is a different session. If B's NAT, for example, assigns one public UDP port for B's communication with A, and then assigns B a *different* public UDP port for the new session B tries to open up with C, then the above technique for peer-to-peer communication will not work because C's messages to B will be directed to the wrong UDP port.

RFC 3022 explicitly allows and suggests that NATs behave in the former, "desirable" fashion, by maintaining a single (public IP, public port) mapping for a given (internal IP, internal port) combination independent of the number of active sessions involving this mapping. This behavior is not only good for compatibility with UDP applications, but it also helps to conserve the NAT's scarce pool of public port numbers. Maintaining a consistent public port mapping does not adversely affect security in any way, either, because incoming traffic can still be *filtered* on a per-session basis regardless of how addresses are *translated*. There in fact appears to be no good reason *not* to implement the desirable behavior in a NAT, except perhaps for the implementation simplicity of naively allocating a new public port for every new session. Unfortunately, RFC 3022 does not *require* NATs to implement the desirable behavior, which has led me to wonder just how many real NATs actually do, and hence this page.

## What NAT Check Does

The program natcheck.c is basically just a program that "pings" a well-known UDP port at two different servers that are publically accessible on the Internet. Both of these servers run the program natserver.c, with the command-line arguments "1" and "2" respectively. In addition, there a third "conspiring" server runs natserver with the command-line argument "3". Whenever each of the first two servers receives a UDP request, it not only sends a reply directly to the sender of that request, but also sends a message to the third server, which in turn "bounces" the reply back to the original client. The effect is that the client will receive not only solicited "ping" replies from the server the request was directed to, but also "unsolicited" replies from the third server.

To determine if the network address translator in use is implementing the desirable behavior of maintaining a single (public IP address, public port) mapping for a given (client IP address, client port), the client program natcheck.c basically just initiates a sequence of simultaneous pings to the first two servers (in case some of the requests or replies are lost in transit) and checks that the client's address and UDP port as reported by both servers is the same. If the NAT naively allocates a new public port for each new session, then the source port as reported by the two servers will be different, and it's time to upgrade your NAT.

The replies echoed from the third server are used only to check whether the NAT properly filters out unsolicited incoming traffic on a per-session basis. Since the client never sends any messages to the third server, if the NAT is properly implementing firewall functionality, the client should never see the third server's echoed replies even after opening up active communication sessions with the first two servers.

## Changes in Version 2

Version 2 of NAT Check contains the following enhancements:

- The NAT Check client no longer attempts to guess whether you have Basic NAT or Network Address/Port Translation (NAPT). It turns to be quite difficult to test for this property reliably, because many NAPTs attempt to bind a private UDP port to a public port with the same port number if that port number is available, causing NAT Check to falsely report Basic NAT. The only way to test for this property reliably would be to run NAT Check on at least two client machines simultaneously, and since this property isn't terribly important to P2P apps it's just not worth the trouble.
- The NAT Check client now tests for one additional NAT feature, which I call *loopback translation*. If a NAT supports loopback translation, it means that a host on the private network behind the NAT can communicate with other hosts on the same private network *using public (translated) port bindings assigned by the NAT*. Most NATs probably do not support this feature yet, but it may become increasingly important in the future where P2P clients may be located behind a common ISP-deployed NAT as well as individual home NATs. More details on loopback translation will appear in the next version of my Internet-Draft, to be released soon.
- The NAT Check client program now has a command-line option, "-v", which turns on verbose messages during the test.

## Related Links

- USENIX '05 paper on P2P over NAT
- NUTSS project at Cornell
- Dan Kegel's NAT/P2P page
- IETF BEHAVE working group
- IETF MIDCOM working group
- nat-peer-games group on Yahoo!