Network Address Translation and Peer-to-Peer Applications (NATP2P)

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

Distribution of this document is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

Abstract

This document describes and recommends methods by which peer-to-peer (P2P) applications can operate efficiently in the presence of Network Address Translation (NAT). This document also provides recommendations for the design of network address translators, in order for them to support P2P applications effectively without compromising security or performance. This memo focuses on the interaction of P2P with NAT in the absence of any special proxy, gateway, or relaying protocols. While not intending to preclude the use of such protocols, the goal of this memo is to enable P2P applications to function automatically without specific knowledge of the type, location, or configuration of the NAT.
1. Introduction

The ubiquitous deployment of Network Address Translation (NAT) on the Internet has provided an effective if hopefully temporary means of working around the ongoing depletion of the IP address space. At the same time, however, the asymmetric addressing and connectivity regimes established by NAT and firewall systems have created unique problems for peer-to-peer (P2P) applications and protocols, such as teleconferencing and multiplayer on-line gaming. This document discusses these issues and how they can be addressed. Familiarity is assumed with NAT terminology and conventions, as specified in [NAT-TRAD] and [NAT-TERM]. As used throughout this document, the term "NAT" refers to "Traditional NAT" in both of its standard variants: namely Basic NAT, in which only IP addresses are translated, as well as Network Address/Port Translation (NAPT), where both IP addresses and transport-level port numbers are translated. In general, this document always assumes NAPT as the standard "worst-case" scenario.

In the traditional client/server paradigm, for which current NAT and firewall mechanisms are primarily designed, network communication typically operates in terms of sessions that originate in a privately addressed network behind a network address translator, and are directed to a well-connected public server with a stable IP address and DNS mapping. The client, or originator, of these connections often does not have its own routable IP address on the public Internet, but instead must share a single public IP address with a number of other hosts on the same private network using the NAT as a multiplexor. The lack of a stable, dedicated public IP address is not a problem for most client software such as web browsers, because the client only needs to be addressable for the duration of a particular session.

In the peer-to-peer paradigm, however, Internet hosts that would normally be considered to be "clients" need to establish communication sessions directly with each other. In this case, both the initiator and responder in a communication session might lie behind a NAT/Firewall, without any permanent IP address or other form of public network presence. A common on-line gaming architecture, for example, is for the participating application hosts to contact a well-known server for initialization and administration purposes, but then to establish direct connections with each other for fast and efficient propagation of updates during gameplay. Similarly, a file sharing application might contact a well-known server for resource discovery or searching purposes, but establish direct connections with peer hosts for data transfer. NAT creates a problem for peer-to-peer connections because unless the NAT is specially configured, hosts behind the NAT have no consistent, permanently usable ports to which incoming TCP or UDP connections from "the Internet at large"
Firewall functionality, which is typically but not always bundled with NAT, creates a similar problem because firewalls are generally designed as one-way filters: sessions initiated inside the protected network are allowed, but attempts by external hosts on the Internet at large to initiate communication sessions with hosts inside the firewall are blocked. RFC 3235 [NAT-APPL] briefly addresses this issue, but does not offer any general solutions that do not compromise security; filling that gap is the purpose of this document.

2. Techniques for P2P Communication with NAT

This section describes in detail the currently known techniques for implementing peer-to-peer communication in the presence of NAT, from the perspective of the application or protocol designer.

2.1. Relaying

The most reliable, but least efficient, method of implementing peer-to-peer communication in the presence of NAT is to make the peer-to-peer communication look to the network like client/server communication. For example, suppose two client hosts, A and B, have each initiated TCP or UDP connections with a well-known server S having a permanent IP address. Clients A and B both reside on privately addressed networks behind network address translators, however, and neither of them have control over a public IP address or permanently stable TCP or UDP port to which incoming connections can be directed.

```
          Server S
                +----------------+------------------+
                |                |                  |
                +----------------+------------------+
                      NAT A         NAT B
              +------------------+
              |                  |
              +------------------+
                    Client A        Client B

Instead of attempting to establish a direct TCP or UDP connection between clients A and B, the two clients can simply use the server S to relay messages between them. For example, to send a message to client B, client A simply sends the message to server S along its already-established client/server connection, and server S then sends the message on to client B using its existing client/server connection with B. This method has the advantage that it will always work as long as both clients have connectivity to the server. Its obvious disadvantages are that it consumes the server’s processing...
power and network bandwidth unnecessarily, and communication latency between the two clients is likely to be increased even if the server is well-connected.

2.2. Connection Reversal

The second technique works if only one of the clients is behind a NAT. For example, suppose client A is behind a NAT but client B is not, as in the following diagram:

Server S
18.181.0.31:1235

----------------------------------------

NAT A
155.99.25.11:62000

Client A
10.0.0.1:1234

Client B
138.76.29.7:1234

Client A has private IP address 10.0.0.1, and the application is using TCP port 1234. This client has established a connection with server S at public IP address 18.181.0.31 and port 1235. NAT A has assigned TCP port 62000, at its own public IP address 155.99.25.11, to serve as the temporary public endpoint address for A's session with S: therefore, server S believes that client A is at IP address 155.99.25.11 using port 62000. Client B, however, has its own permanent IP address, 138.76.29.7, and the peer-to-peer application on B is accepting TCP connections at port 1234.

Now suppose client B would like to initiate a peer-to-peer communication session with client A. B might first attempt to contact client A either at the address client A believes itself to have, namely 10.0.0.1:1234, or at the address of A as observed by server S, namely 155.99.25.11:62000. In either case, however, the connection will fail. In the first case, traffic directed to IP address 10.0.0.1 will simply be dropped by the network because 10.0.0.1 is not a publicly routable IP address. In the second case, the TCP SYN request from B will arrive at NAT A directed to port 62000, but NAT A will typically reject the connection request with a RST packet because only outgoing connections are allowed.

After attempting and failing to establish a direct connection to A, client B can use server S to relay a request to client A to initiate a "reversed" connection to client B. Client A, upon receiving this
relayed request through S, opens a TCP connection to client B at B’s public IP address and port number. NAT A allows the connection to proceed because it is originating inside the firewall, and client B can receive the connection because it is not behind a NAT.

A variety of current peer-to-peer systems implement this technique. Its main limitation, of course, is that it only works as long as only one of the communicating peers is behind a NAT: if both peers are behind NATs, then the method fails. Because connection reversal is not a general solution to the problem, it is NOT recommended as a primary strategy. Applications may choose to attempt connection reversal, but should be able to fall back automatically on another mechanism such as relaying if neither a "forward" nor a "reversed" connection can be established.

2.3. UDP Hole Punching

The third technique, and the one of primary interest in this memo, is sometimes known as "UDP Hole Punching." UDP hole punching relies on well-established NAT conventions to allow appropriately designed peer-to-peer applications to "punch holes" through NATs and firewalls and establish direct connectivity with each other, even when both communicating hosts may lie behind a NAT. This technique was mentioned briefly in section 5.1 of RFC 3027 [NAT-PROT] and has been informally described elsewhere on the Internet [KEGEL]. As the name implies, unfortunately, this technique works reliably only with UDP.

We will consider two specific scenarios, and how applications can be designed to handle both of them gracefully. In the first situation, representing the common case, two clients desiring direct peer-to-peer communication reside behind different NATs. In the second, the two clients actually reside behind the same NAT, but do not necessarily know that they do.

2.3.1. Clients Behind Different NATs

Suppose clients A and B both have private IP addresses and lie behind different network address translators. The peer-to-peer application running on clients A and B and on server S each use UDP port 1234. A and B have each initiated UDP communication sessions with server S, causing NAT A to assign its own public UDP port 62000 for A’s session with S, and causing NAT B to assign its port 31000 to B’s session with S, respectively.
Now suppose that client A wants to establish a UDP communication session directly with client B. If A simply starts sending UDP requests to B’s public address, 138.76.29.7:31000, then NAT B will typically discard these incoming messages because the source address and port number does not match those of S, with which the original outgoing session was established. Similarly, if B simply starts sending UDP requests to A’s public address, then NAT A will discard these messages.

Suppose A starts sending UDP requests to B’s public address, however, and simultaneously relays a request through server S to B, asking B to start sending UDP requests to A’s public address. A’s outgoing messages directed to B’s public address (138.76.29.7:31000) will cause NAT A to open up a new communication session between A’s private address and B’s public address. At the same time, B’s messages to A’s public address (155.99.25.11:62000) will cause NAT B to open up a new communication session between B’s private address and A’s public address. Once the new UDP sessions have been opened up in each direction, client A and B can communicate with each other directly without further reference to or burden on the "introduction" server S.

The UDP hole punching technique has several useful properties. Once a direct peer-to-peer UDP connection has been established between two NATted clients, either party on that connection can in turn take over the role of "introducer" and help the other party establish peer-to-peer connections with additional peers, minimizing the load on the initial introduction server S. The application does not need to attempt to detect explicitly what kind of NAT it is behind, if any [STUN], since the procedure above will establish peer-to-peer communication channels equally well if either or both clients do not happen to be behind a NAT. The hole punching technique even works automatically under "Twice NAT", where one or both clients are removed from the public Internet via two or more levels of address translation.
2.3.2. Clients Behind the Same NAT

Now consider the scenario in which the two clients (probably unknowingly) happen to reside behind the same NAT, and are therefore located in the same private IP address space. Client A has established a UDP session with server S, 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.

```
Server S
  18.181.0.31:1234
    |
    NAT
    A-S 155.99.25.11:62000
    B-S 155.99.25.11:62001
    +----------------------+----------------------+
      |                                    |
      |  Client A                             |
      |  10.0.0.1:1234                         |
      |                                    |
      |                                    |
      |  Client B                             |
      |  10.1.1.3:1234                        |
```

Suppose that A and B use the UDP hole punching technique as outlined above to establish a communication channel using server S as an introducer. Then A and B will learn each other’s public IP addresses and port numbers as observed by server S, and start sending each other messages at those public addresses. The two clients will be able to communicate with each other this way as long as the NAT allows hosts on the internal network to open translated UDP sessions with other internal hosts and not just with external hosts. For example, when A sends a UDP packet to B’s public address, the packet initially has a source IP address and port number of 10.0.0.1:124 and a destination of 155.99.25.11:62001. The NAT receives this packet, translates it to have a source of 155.99.25.11:62000 (A’s public address) and a destination of 10.1.1.3:1234, and then forwards it on to B. Even if supported by the NAT, this translation and forwarding step is obviously unnecessary in this situation, and is likely to add latency to the dialog between A and B as well as burdening the NAT.

The solution to this problem is straightforward, however. When A and B initially exchange address information through server S, they should include their own IP addresses and port numbers as "observed" by themselves, as well as their addresses as observed by S. The clients then simultaneously start sending packets to each other at each of the alternative addresses they know about, and use the first address that leads to successful communication. If the two clients are behind the same NAT, then the packets directed to their private
addresses are likely to arrive first, resulting in a direct
communication channel not involving the NAT. If the two clients are
behind different NATs, then the packets directed to their private
addresses will fail to reach each other at all, but the clients will
hopefully establish connectivity using their respective public
addresses. It is important that these packets be authenticated in
some way, however, since in the case of different NATs it is entirely
possible for A’s messages to B’s private address to reach some other,
unrelated node on A’s private network, or vice versa.

2.3.3. Consistent Port Mappings

The hole punching technique has one main caveat: it works only if
each of the NATs maintains a single, consistent mapping from a given
(private IP address, private UDP port) pair to a translated (public
IP address, public UDP port) pair, for as long as that UDP port is in
use. For a given UDP port on the private network, the NAT must
ensure that the internal port is always mapped to the same public UDP
port on the globally addressable Internet, even if communication
occurs between that internal UDP port and multiple distinct external
destinations on the Internet. In particular, the NAT must not just
naively allocate and assign a new public UDP port for each new
session initiated from within the firewall boundary, where a
"session" is defined by the addresses and port numbers of both
communicating endpoints. Assigning a new public port for each new
session makes it impossible for a UDP application to reuse an
already-established (public IP address, public UDP port) mapping for
communication with different external destinations.

While RFC 3022 [NAT-TRAD] suggests and explicitly allows NATs to
maintain a single mapping from a (private IP address, private port)
pair to a (public IP address, public port) pair, it unfortunately
does not mandate this behavior. Therefore, while many NATs implement
the desirable behavior and therefore allow direct UDP-based P2P
connections using the hole punching technique, other NATs currently
do not support the technique. Since this is the most efficient known
method of establishing direct peer-to-peer communication between two
nodes that are both behind NATs, and it works with a wide variety of
existing NATs, it is recommended that applications use this technique
if efficient peer-to-peer communication is required, but be prepared
to fall back on simple relaying when direct communication cannot be
established.

2.4. UDP Port Number Prediction

A variant of the UDP hole punching technique discussed above exists
that allows peer-to-peer UDP sessions to be created in the presence
of some NATs that do not maintain a consistent mapping between
private and public UDP ports. This method, sometimes called the "N+1" technique [BIDIR], works if public port numbers selected by the NAT are not held constant across all sessions originating at a given private IP address and port, but are nevertheless predictable. Consider again the situation in which two clients, A and B, each behind a separate NAT, have each established UDP connections with a permanently addressable server S:

Server S
18.181.0.31:1234

+----------------------+----------------------+
|                                             |
| NAT A                                         NAT B |
| A-S 155.99.25.11:62000                        B-S 138.76.29.7:31000 |
|                                             |
| Client A                                      Client B |
| 10.0.0.1:1234                                 10.1.1.3:1234 |

NAT A has assigned its own UDP port 62000 to the communication session between A and S, and NAT B has assigned its port 31000 to the session between B and S. By communicating through server S, A and B learn each other’s public IP addresses and port numbers as observed by S. Client A now starts sending UDP messages to port 31001 at address 138.76.29.7 (note the port number increment), and client B simultaneously starts sending messages to port 62001 at address 155.99.25.11. If NATs A and B assign port numbers to new sessions sequentially, and if not much time has passed since the A-S and B-S sessions were initiated, then a working bidirectional communication channel between A and B should result. A’s messages to B cause NAT A to open up a new session, to which NAT A will (hopefully) assign public port number 62001, because 62001 is next in sequence after the port number 62000 it previously assigned to the session between A and S. Similarly, B’s messages to A will cause NAT B to open a new session, to which it will (hopefully) assign port number 31001. If both clients have correctly guessed the port numbers each NAT assigns to the new sessions, then a bidirectional UDP communication channel will have been established as shown below.
Obviously there are many things that can cause this trick to fail. If the predicted port number at either NAT already happens to be in use by an unrelated session, then the NAT will skip over that port number and the connection attempt will fail. If either NAT sometimes or always chooses port numbers non-sequentially, then the trick will fail. If a different client behind NAT A (or B respectively) opens up a new outgoing UDP connection to any external destination after A (B) establishes its connection with S but before sending its first message to B (A), then the unrelated client will inadvertently "steal" the desired port number. This trick is therefore much less likely to work when either NAT involved is under load. For all of these reasons, it is NOT recommended that new applications implement this trick; it is described here for purely for historical and informational purposes.

2.5. Simultaneous TCP Connection Initiation

There is a method that can be used in some cases to establish direct peer-to-peer TCP connections between a pair of nodes that are both behind NATs. Most TCP sessions start with one endpoint sending a SYN packet, to which the other party responds with a SYN-ACK packet. It is possible and legal, however, for two endpoints to start a TCP session by simultaneously sending each other SYN packets, to which each party subsequently responds with a separate ACK. This procedure is known as a "simultaneous open."

If a NAT receives a TCP SYN packet from outside the private network attempting to initiate an incoming TCP connection, the NAT will normally reject the connection attempt by sending back a TCP RST (connection reset) packet. If, however, the SYN packet arrives with source and destination addresses and port numbers that correspond to a TCP session that the NAT believes is already active, then the NAT will allow the packet to pass through. In particular, if the NAT has just recently seen and transmitted an outgoing SYN packet with the
same addresses and port numbers, then it will consider the session active and allow the incoming SYN through. If clients A and B can each correctly predict the public port number that its respective NAT will assign the next outgoing TCP connection, and if each client initiates an outgoing TCP connection with the other client timed so that each client’s outgoing SYN passes through its local NAT before either SYN reaches the opposite NAT, then a working peer-to-peer TCP connection will result.

Unfortunately, this trick is even more fragile and timing-sensitive than the UDP port number prediction trick described above. First, all the same things can go wrong with each side’s attempt to predict the public port numbers the NATs will assign to the new sessions. In addition, if either client’s SYN arrives at the opposite NAT too quickly, then the NAT will reject the SYN with a RST packet, causing the local NAT in turn to close the new session. Finally, even though support for simultaneous open is technically a mandatory part of the TCP specification [TCP], it is not implemented correctly or at all in many common operating systems. For this reason, this trick is likewise mentioned here only for historical interest; it is NOT recommended for use by applications. Applications that require efficient, direct peer-to-peer communication should use UDP.

3. NAT Design Guidelines

This section discusses considerations for the design of network address translators, as they affect peer-to-peer applications.

3.1. Maintaining Consistent Public/Private Mappings for UDP Ports

The primary and most important recommendation of this document for NAT designers is that the NAT maintain a consistent and stable mapping between a given (internal IP address, internal UDP port) pair and a corresponding (public IP address, public UDP port) pair for as long as any active sessions exist using that mapping. The NAT may filter incoming traffic on a per-session basis, by examining both the source and destination IP addresses and port numbers in each packet. When a node on the private network starts sending outgoing packets to a new external destination, however, while using the same source IP address and UDP port as an existing translated UDP session, the NAT should ensure that the new UDP session is given the same public IP address and UDP port numbers as the existing session.

One method of implementing this behavior, meant to be only illustrative and not prescriptive, is outlined here. The "critical-path" processing performed by a NAT on a packet flowing in either direction typically involves extracting the source and destination IP addresses from the IP header, and the source and destination TCP/UDP
port numbers from the TCP/UDP header, and using these four values to index into a table of active sessions. When a packet is received for which no entry is found in the session table, the NAT must decide whether or not and how to establish a new session, and this is where the typical "outgoing sessions only" firewall policy comes into effect.

If the unknown packet was received from the public network, then it is usually dropped (or perhaps rejected with a RST in the case of TCP). If a UDP packet for an unknown session arrives from the private network, however, and the NAT is configured in the typical way to allow outgoing UDP sessions, then NAT looks up the packet’s (source IP address, source UDP port) in a second internal table: a "mapping table." For a given (private IP address, private UDP port) on the internal network, the NAT stores in the mapping table the corresponding (public IP address, public UDP port) it has assigned to represent that private endpoint. If a mapping already exists, the NAT simply uses the existing mapping when constructing the new session. If no such entry is present in the mapping table, then the NAT allocates a new public UDP port from its pool and creates a new mapping table entry along with the new session. The NAT also maintains with each entry in the mapping table a list or count of the active sessions using that mapping, so that it can reassign the public UDP port to other purposes once all of the outstanding sessions for the mapping are deemed inactive.

3.1.1. Preserving Port Numbers

Some NATs, when establishing a new UDP session, attempt to assign the same public port number as the corresponding private port number, if that port number happens to be available. For example, if client A at address 10.0.0.1 initiates an outgoing UDP session with a datagram from port number 1234, and the NAT’s public port number 1234 happens to be available, then the NAT uses port number 1234 at the NAT’s public IP address as the translated endpoint address for the session. This behavior might be beneficial to some legacy UDP applications that expect to communicate only using specific UDP port numbers, but it is not recommended that applications depend on this behavior since it is only possible for a NAT to preserve the port number if at most one node on the internal network is using that port number.

In addition, a NAT should NOT try to preserve the port number in a new session if doing so would conflict with the goal of maintaining a consistent mapping between public and private endpoint addresses. For example, suppose client A at internal port 1234 has established a session with external server S, and NAT A has assigned public port 62000 to this session because port number 1234 on the NAT was not available at the time. Now suppose port number 1234 on the NAT
subsequently becomes available, and while the session between A and S is still active, client A initiates a new session from its same internal port (1234) to a different external node B. In this case, because an active mapping has already been established between client A’s port 1234 and the NAT’s public port 62000, this mapping should be maintained and the new session should also use port 62000 as the public port corresponding to client A’s port 1234. The NAT should NOT assign public port 1234 to this new session just because port 1234 has become available; that behavior would not be likely to benefit the application in any way since the application has already been operating with a translated port number, and it would break any attempts the application might make to establish peer-to-peer connections using the UDP hole punching technique.

3.2. Maintaining Consistent Public/Private Mappings for TCP Ports

For consistency with the behavior of UDP translation, it is suggested that NATs also maintain a consistent mapping between private and public (IP address, TCP port number) pairs for TCP connections, in the same way as described above for UDP. Maintaining consistent mappings for TCP ports facilitates the operation of the simultaneous TCP open technique, which although not recommended in general for peer-to-peer applications, may be useful in controlled situations when the two endpoints are sufficiently well synchronized. Maintaining TCP endpoint mappings consistently may also increase the NAT’s compatibility with other applications that initiate multiple TCP connections from the same source port.

3.3. Proxy Protocols

Besides adopting the above recommendations to make a NAT’s basic "transparent-mode" operation as peer-to-peer friendly as possible, it is helpful for NATs also to support proxy protocols that allow applications to request an explicitly managed presence on the public side of the NAT. Unfortunately, several alternative protocols have been proposed with varying characteristics [SOCKS, RSIP, MIDCOM, UPNP], and as of this writing none of them have achieved clear acceptance or dominance in the Internet community. Furthermore, it is not clear yet how well these protocols will work in the increasingly common "Twice NAT" situation where clients are located behind multiple levels of NAT, especially if the NATs are from different vendors, support different features and policies, and are under different administrative domains. (In the common case, one is owned and managed by the ISP and the other by the end user.) For these reasons, this document makes no attempt to explore this issue in detail or to recommend specific proxy protocols for NATs to implement.
4. Security Considerations

Using the UDP hole punching technique in peer-to-peer applications and supporting it in NATs should not create any new security issues. In particular, the technique does not require a NAT firewall to be "promiscuous" in any way about accepting incoming UDP traffic. As long as outgoing UDP sessions are enabled and the firewall maintains consistent mappings between internal and external UDP ports, the firewall can still filter out all incoming UDP packets except those with (source IP, source port, destination IP, destination port) tuples that exactly match those of active sessions initiated from within the enclave. Filtering incoming traffic aggressively while maintaining consistent mappings thus allows a firewall to be "peer-to-peer friendly" without compromising the standard firewall security principle of rejecting all unsolicited incoming traffic.

It might be argued that maintaining a consistent internal/external port mapping can "leak" some information to the outside about the applications on the internal network, particularly about the relationships between different UDP sessions that cross the firewall boundary. If the security requirements are so critical that such a subtle information channel is of concern, however, then the firewall almost certainly should not be configured to allow unrestricted outgoing UDP traffic in the first place. Controlling information flow to this degree generally requires that the firewall only allow communication only via tightly-controllable application-level gateways, in which case the firewall can either implement the proper peer-to-peer communication behavior itself or disallow it entirely.

References


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


http://www.upnp.org/standardizeddcps/igd.asp

Author’s Address

Bryan Ford  
Laboratory for Computer Science  
Massachusetts Institute of Technology  
77 Massachusetts Ave.  
Cambridge, MA 02139  

Phone: (617) 253-5261  
E-mail: baford@mit.edu  
Web: http://www.brynosaurus.com/

Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this
document itself may not be modified in any way, such as by removing
the copyright notice or references to the Internet Society or other
Internet organizations, except as needed for the purpose of
developing Internet standards in which case the procedures for
copyrights defined in the Internet Standards process must be
followed, or as required to translate it into languages other than
English.

The limited permissions granted above are perpetual and will not be
revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an
"AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING
TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.