## Teredo: Tunneling IPv6 over UDP
## through Network Address Translations (NATs)

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements.  Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol.  Distribution of this memo is unlimited.

Copyright Notice

Abstract

   We propose here a service that enables nodes located behind one or
   more IPv4 Network Address Translations (NATs) to obtain IPv6
   connectivity by tunneling packets over UDP; we call this the Teredo
   service.  Running the service requires the help of "Teredo servers"
   and "Teredo relays".  The Teredo servers are stateless, and only have
   to manage a small fraction of the traffic between Teredo clients; the
   Teredo relays act as IPv6 routers between the Teredo service and the
   "native" IPv6 Internet.  The relays can also provide interoperability
   with hosts using other transition mechanisms such as "6to4".

Table of Contents

1.  **Introduction**

   Classic tunneling methods envisaged for IPv6 transition operate by
   sending IPv6 packets as payload of IPv4 packets; the 6to4 proposal
   [RFC3056] proposes automatic discovery in this context.  A problem
   with these methods is that they don't work when the IPv6 candidate
   node is isolated behind a Network Address Translator (NAT) device:
   NATs are typically not programmed to allow the transmission of
   arbitrary payload types; even when they are, the local address cannot
   be used in a 6to4 scheme. 6to4 will work with a NAT if the NAT and
   6to4 router functions are in the same box; we want to cover the
   relatively frequent case when the NAT cannot be readily upgraded to
   provide a 6to4 router function.

   A possible way to solve the problem is to rely on a set of "tunnel
   brokers".  However, there are limits to any solution that is based on
   such brokers: the quality of service may be limited, since the
   traffic follows a dogleg route from the source to the broker and then
   the destination; the broker has to provide sufficient transmission
   capacity to relay all packets and thus suffers a high cost.  For
   these two reasons, it may be desirable to have solutions that allow
   for "automatic tunneling", i.e., let the packets follow a direct path
   to the destination.

   The automatic tunneling requirement is indeed at odds with some of
   the specificities of NATs.  Establishing a direct path supposes that
   the IPv6 candidate node can retrieve a "globally routable" address
   that results from the translation of its local address by one or more
   NATs; it also supposes that we can find a way to bypass the various
   "per destination protections" that many NATs implement.  In this
   memo, we will explain how IPv6 candidates located behind NATs use
   "Teredo servers" to learn their "global address" and to obtain
   connectivity, how they exchange packets with native IPv6 hosts
   through "Teredo relays", and how clients, servers, and relays can be
   organized in Teredo networks.

   The specification is organized as follows.  Section 2 contains the
   definition of the terms used in the memo.  Section 3 presents the
   hypotheses on NAT behavior used in the design, as well as the
   operational requirements that the design should meet.  Section 4
   presents the IPv6 address format used by Teredo.  Section 5 contains
   the format of the messages and the specification of the protocol.
   Section 6 presents guidelines for further work on configured tunnels
   that would be complementary to the current approach.  Section 7
   contains a security discussion, section 8 contains a discussion of
   the Unilateral Self Address Fixing (UNSAF) issues, and section 9
   contains IANA considerations.

## 2.  Definitions

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 [RFC2119].

This specification uses the following definitions:

### 2.1.  Teredo Service

The transmission of IPv6 packets over UDP, as defined in this memo.

### 2.2.  Teredo Client

A node that has some access to the IPv4 Internet and wants to gain
access to the IPv6 Internet.

### 2.3.  Teredo Server

A node that has access to the IPv4 Internet through a globally
routable address, and is used as a helper to provide IPv6
connectivity to Teredo clients.

### 2.4.  Teredo Relay

An IPv6 router that can receive traffic destined to Teredo clients
and forward it using the Teredo service.

### 2.5.  Teredo IPv6 Service Prefix

An IPv6 addressing prefix that is used to construct the IPv6 address
of Teredo clients.

### 2.6.  Global Teredo IPv6 Service Prefix

An IPv6 addressing prefix whose value is 2001:0000:/32.

### 2.7.  Teredo UDP Port

The UDP port number at which Teredo servers are waiting for packets.
The value of this port is 3544.

### 2.8.  Teredo Bubble

A Teredo bubble is a minimal IPv6 packet, made of an IPv6 header and
a null payload.  The payload type is set to 59, No Next Header, as
per [RFC2460].  The Teredo clients and relays may send bubbles in
order to create a mapping in a NAT.

### 2.9.  Teredo Service Port

The port from which the Teredo client sends Teredo packets.  This
port is attached to one of the client's IPv4 addresses.  The IPv4
address may or may not be globally routable, as the client may be
located behind one or more NAT.

### 2.10.  Teredo Server Address

The IPv4 address of the Teredo server selected by a particular
client.

### 2.11.  Teredo Mapped Address and Teredo Mapped Port

A global IPv4 address and a UDP port that results from the
translation of the IPv4 address and UDP port of a client's Teredo
service port by one or more NATs.  The client learns these values
through the Teredo protocol described in this memo.

### 2.12.  Teredo IPv6 Client Prefix

A global scope IPv6 prefix composed of the Teredo IPv6 service prefix
and the Teredo server address.

### 2.13.  Teredo Node Identifier

A 64-bit identifier that contains the UDP port and IPv4 address at
which a client can be reached through the Teredo service, as well as
a flag indicating the type of NAT through which the client accesses
the IPv4 Internet.

### 2.14.  Teredo IPv6 Address

A Teredo IPv6 address obtained by combining a Teredo IPv6 client
prefix and a Teredo node identifier.

### 2.15.  Teredo Refresh Interval

The interval during which a Teredo IPv6 address is expected to remain
valid in the absence of "refresh" traffic.  For a client located
behind a NAT, the interval depends on configuration parameters of the
local NAT, or the combination of NATs in the path to the Teredo
server.  By default, clients assume an interval value of 30 seconds;
a longer value may be determined by local tests, as described in
section 5.

## 2.16.  Teredo Secondary Port

   A UDP port used to send or receive packets in order to determine the
   appropriate value of the refresh interval, but not used to carry any
   Teredo traffic.

## 2.17.  Teredo IPv4 Discovery Address

   An IPv4 multicast address used to discover other Teredo clients on
   the same IPv4 subnet.  The value of this address is 224.0.0.253.

## 3.  Design Goals, Requirements, and Model of Operation

   The proposed solution transports IPv6 packets as the payload of UDP
   packets.  This is based on the observation that TCP and UDP are the
   only protocols guaranteed to cross the majority of NAT devices.
   Tunneling packets over TCP would be possible, but would result in a
   poor quality of service; encapsulation over UDP is a better choice.

   The design of our solution is based on a set of hypotheses and
   observations on the behavior of NATs, our desire to provide an "IPv6
   provider of last resort", and a list of operational requirements.  It
   results in a model of operation in which the Teredo service is
   enabled by a set of servers and relays.

## 3.1.  Hypotheses about NAT Behavior

   NAT devices typically incorporate some support for UDP, in order to
   enable users in the natted domain to use UDP-based applications.  The
   NAT will typically allocate a "mapping" when it sees a UDP packet
   coming through for which there is not yet an existing mapping.  The
   handling of UDP "sessions" by NAT devices differs by two important
   parameters, the type and the duration of the mappings.

   The type of mappings is analyzed in [RFC3489], which distinguishes
   between "cone NAT", "restricted cone NAT", "port restricted cone NAT"
   and "symmetric NAT".  The Teredo solution ensures connectivity for
   clients located behind cone NATs, restricted cone NATs, or port-
   restricted cone NATs.

   Transmission of regular IPv6 packets only takes place after an
   exchange of "bubbles" between the parties.  This exchange would often
   fail for clients behind symmetric NAT, because their peer cannot
   predict the UDP port number that the NAT expects.

   Clients located behind a symmetric NAT will only be able to use
   Teredo if they can somehow program the NAT and reserve a Teredo
   service port for each client, for example, using the DMZ functions of

the NAT.  This is obviously an onerous requirement, at odds with the
design goal of an automatic solution.  However, measurement campaigns
and studies of documentations have shown that, at least in simple
"unmanaged" networks, symmetric NATs are a small minority; moreover,
it seems that new NAT models or firmware upgrades avoid the
"symmetric" design.

Investigations on the performance of [RFC3489] have shown the
relative frequency of a particular NAT design, which we might call
"port conserving".  In this design, the NAT tries to keep the same
port number inside and outside, unless the "outside" port number is
already in use for another mapping with the same host.  Port
conserving NAT appear as "cone" or "restricted cone NAT" most of the
time, but they will behave as "symmetric NAT" when multiple internal
hosts use the same port number to communicate to the same server.

The Teredo design minimizes the risk of encountering the "symmetric"
behavior by asking multiple hosts located behind the same NAT to use
different Teredo service ports.

Other investigation in the behavior of NAT also outlined the
"probabilistic rewrite" behavior.  Some brands of NAT will examine
all packets for "embedded addresses", IP addresses, and port numbers
present in application payloads.  They will systematically replace
32-bit values that match a local address by the corresponding mapped
address.  The Teredo specification includes an "obfuscation"
procedure in order to avoid this behavior.

Regardless of their types, UDP mappings are not kept forever.  The
typical algorithm is to remove the mapping if no traffic is observed
on the specified port for a "lifetime" period.  The Teredo client
that wants to maintain a mapping open in the NAT will have to send
some "keep alive" traffic before the lifetime expires.  For that, it
needs an estimate of the "lifetime" parameter used in the NAT.  We
observed that the implementation of lifetime control can vary in
several ways.

Most NATs implement a "minimum lifetime", which is set as a parameter
of the implementation.  Our observations of various boxes showed that
this parameter can vary between about 45 seconds and several minutes.

In many NATs, mappings can be kept for a duration that exceeds this
minimum, even in the absence of traffic.  We suspect that many
implementation perform "garbage collection" of unused mappings on
special events, e.g., when the overall number of mappings exceeds
some limit.

In some cases, e.g., NATs that manage Integrated Services Digital
Network (ISDN) or dial-up connections, the mappings will be released
when the connection is released, i.e., when no traffic is observed on
the connection for a period of a few minutes.

Any algorithm used to estimate the lifetime of mapping will have to
be robust against these variations.

In some cases, clients are located behind multiple NAT.  The Teredo
procedures will ensure communications between clients between
multiple NATs and clients "on the other side" of these NATs.  They
will also ensure communication when clients are located in a single
subnet behind the same NAT.

The procedures do not make any hypothesis about the type of IPv4
address used behind a NAT, and in particular do not assume that these
are private addresses defined in [RFC1918].

## 3.2.  IPv6 Provider of Last Resort

Teredo is designed to provide an "IPv6 access of last resort" to
nodes that need IPv6 connectivity but cannot use any of the other
IPv6 transition schemes.  This design objective has several
consequences on when to use Teredo, how to program clients, and what
to expect of servers.  Another consequence is that we expect to see a
point in time at which the Teredo technology ceases to be used.

### 3.2.1.  When to Use Teredo

Teredo is designed to robustly enable IPv6 traffic through NATs, and
the price of robustness is a reasonable amount of overhead, due to
UDP encapsulation and transmission of bubbles.  Nodes that want to
connect to the IPv6 Internet SHOULD only use the Teredo service as a
"last resort" option: they SHOULD prefer using direct IPv6
connectivity if it is locally available, if it is provided by a 6to4
router co-located with the local NAT, or if it is provided by a
configured tunnel service; and they SHOULD prefer using the less
onerous 6to4 encapsulation if they can use a global IPv4 address.

### 3.2.2.  Autonomous Deployment

In an IPv6-enabled network, the IPv6 service is configured
automatically, by using mechanisms such as IPv6 Stateless Address
Autoconfiguration [RFC2462] and Neighbor Discovery [RFC2461].  A
design objective is to configure the Teredo service as automatically
as possible.  In practice, however, it is required that the client
learn the IPv4 address of a server that is willing to serve the
client; some servers may also require some form of access control.

### 3.2.3.  Minimal Load on Servers

During the peak of the transition, there will be a requirement to
deploy Teredo servers supporting a large number of Teredo clients.
Minimizing the load on the server is a good way to facilitate this
deployment.  To achieve this goal, servers should be as stateless as
possible, and they should also not be required to carry any more
traffic than necessary.  To achieve this objective, we require only
that servers enable the packet exchange between clients, but we don't
require servers to carry the actual data packets: these packets will
have to be exchanged directly between the Teredo clients, or through
a destination-selected relay for exchanges between Teredo clients and
other IPv6 clients.

### 3.2.4.  Automatic Sunset

Teredo is meant as a short-term solution to the specific problem of
providing IPv6 service to nodes located behind a NAT.  The problem is
expected to be resolved over time by transforming the "IPv4 NAT" into
an "IPv6 router".  This can be done in one of two ways:  upgrading
the NAT to provide 6to4 functions or upgrading the Internet
connection used by the NAT to a native IPv6 service, and then adding
IPv6 router functionality in the NAT.  In either case, the former NAT
can present itself as an IPv6 router to the systems behind it.  These
systems will start receiving the "router advertisements"; they will
notice that they have IPv6 connectivity and will stop using Teredo.

### 3.3.  Operational Requirements

### 3.3.1.  Robustness Requirement

The Teredo service is designed primarily for robustness: packets are
carried over UDP in order to cross as many NAT implementations as
possible.  The servers are designed to be stateless, which means that
they can easily be replicated.  We expect indeed to find many such
servers replicated at multiple Internet locations.

### 3.3.2.  Minimal Support Cost

The service requires the support of Teredo servers and Teredo relays.
In order to facilitate the deployment of these servers and relays,
the Teredo procedures are designed to minimize the amount of
coordination required between servers and relays.

Meeting this objective implies that the Teredo addresses will
incorporate the IPv4 address and UDP port through which a Teredo
client can be reached.  This creates an implicit limit on the

stability of the Teredo addresses, which can only remain valid as
long as the underlying IPv4 address and UDP port remain valid.

### 3.3.3.  Protection against Denial of Service Attacks

The Teredo clients obtain mapped addresses and ports from the Teredo
servers.  The service must be protected against denial of service
attacks in which a third party spoofs a Teredo server and sends
improper information to the client.

### 3.3.4.  Protection against Distributed Denial of Service Attacks

Teredo relays will act as a relay for IPv6 packets.  Improperly
designed packet relays can be used by denial of service attackers to
hide their address, making the attack untraceable.  The Teredo
service must include adequate protection against such misuse.

### 3.3.5.  Compatibility with Ingress Filtering

Routers may perform ingress filtering by checking that the source
address of the packets received on a given interface is "legitimate",
i.e., belongs to network prefixes from which traffic is expected at a
network interface.  Ingress filtering is a recommended practice, as
it thwarts the use of forged source IP addresses by malfeasant
hackers, notably to cover their tracks during denial of service
attacks.  The Teredo specification must not force networks to disable
ingress filtering.

### 3.4.  Model of Operation

The operation of Teredo involves four types of nodes: Teredo clients,
Teredo servers, Teredo relays, and "plain" IPv6 nodes.

Teredo clients start operation by interacting with a Teredo server,
performing a "qualification procedure".  During this procedure, the
client will discover whether it is behind a cone, restricted cone, or
symmetric NAT.  If the client is not located behind a symmetric NAT,
the procedure will be successful and the client will configure a
"Teredo address".

The Teredo IPv6 address embeds the "mapped address and port" through
which the client can receive IPv4/UDP packets encapsulating IPv6
packets.  If the client is not located behind a cone NAT,
transmission of regular IPv6 packets must be preceded by an exchange
of "bubbles" that will install a mapping in the NAT.  This document
specifies how the bubbles can be exchanged between Teredo clients in
order to enable transmission along a direct path.

Teredo clients can exchange IPv6 packets with plain IPv6 nodes (e.g.,
native nodes or 6to4 nodes) through Teredo relays.  Teredo relays
advertise reachability of the Teredo prefix to a certain subset of
the IPv6 Internet: a relay set up by an ISP will typically serve only
the IPv6 customers of this ISP; a relay set-up for a site will only
serve the IPv6 hosts of this site.  Dual-stack hosts may implement a
"local relay", allowing them to communicate directly with Teredo
hosts by sending IPv6 packets over UDP and IPv4 without having to
advertise a Teredo IPv6 address.

Teredo clients have to discover the relay that is closest to each
native IPv6 or 6to4 peer.  They have to perform this discovery for
each native IPv6 or 6to4 peer with which they communicate.  In order
to prevent spoofing, the Teredo clients perform a relay discovery
procedure by sending an ICMP echo request to the native host.  This
message is a regularly formatted IPv6 ICMP packet, which is
encapsulated in UDP and sent by the client to its Teredo server; the
server decapsulates the IPv6 message and forwards it to the intended
IPv6 destination.  The payload of the echo request contains a large
random number.  The echo reply is sent by the peer to the IPv6
address of the client, and is forwarded through standard IPv6 routing
mechanisms.  It will naturally reach the Teredo relay closest to the
native or 6to4 peer, and will be forwarded by this relay using the
Teredo mechanisms.  The Teredo client will discover the IPv4 address
and UDP port used by the relay to send the echo reply, and will send
further IPv6 packets to the peer by encapsulating them in UDP packets
sent to this IPv4 address and port.  In order to prevent spoofing,
the Teredo client verifies that the payload of the echo reply
contains the proper random number.

The procedures are designed so that the Teredo server only
participates in the qualification procedure and in the exchange of
bubbles and ICMP echo requests.  The Teredo server never carries
actual data traffic.  There are two rationales for this design:
reduce the load on the server in order to enable scaling, and avoid
privacy issues that could occur if a Teredo server kept copies of the
client's data packets.

## 4.  Teredo Addresses

The Teredo addresses are composed of 5 components:

```
+-------------+-------------+-------+------+-------------+
| Prefix      | Server IPv4 | Flags | Port | Client IPv4 |
+-------------+-------------+-------+------+-------------+
```

- Prefix: the 32-bit Teredo service prefix.
- Server IPv4: the IPv4 address of a Teredo server.

   - Flags: a set of 16 bits that document type of address and NAT.
   - Port: the obfuscated "mapped UDP port" of the Teredo service at
     the client.
   - Client IPv4: the obfuscated "mapped IPv4 address" of the client.

   In this format, both the "mapped UDP port" and "mapped IPv4 address"
   of the client are obfuscated.  Each bit in the address and port
   number is reversed; this can be done by an exclusive OR of the 16-bit
   port number with the hexadecimal value 0xFFFF, and an exclusive OR of
   the 32-bit address with the hexadecimal value 0xFFFFFFFF.

   The IPv6 addressing rules specify that "for all unicast addresses,
   except those that start with binary value 000, Interface IDs are
   required to be 64 bits long and to be constructed in Modified EUI-64
   format".  This dictates the encoding of the flags, 16 intermediate
   bits that should correspond to valid values of the most significant
   16 bits of a Modified EUI-64 ID:

```
      0        0 0       1
      |0       7 8       5
      +----+----+----+----+
      |Czzz|zzUG|zzzz|zzzz|
      +----+----+----+----+
```

   In this format:

   -  The bits "UG" should be set to the value "00", indicating a non-
      global unicast identifier;
   -  The bit "C" (cone) should be set to 1 if the client believes it is
      behind a cone NAT, to 0 otherwise; these values determine
      different server behavior during the qualification procedure, as
      specified in [Section 5.2.1](#), as well as different bubble processing
      by clients and relays.
   -  The bits indicated with "z" must be set to zero and ignored on
      receipt.

   Thus, there are two currently specified values of the Flags field:
   "0x0000" (all null) if the cone bit is set to 0, and "0x8000" if the
   cone bit is set to 1.  (Further versions of this specification may
   assign new values to the reserved bits.)

   In some cases, Teredo nodes use link-local addresses.  These
   addresses contain a link-local prefix (FE80::/64) and a 64-bit
   identifier, constructed using the same format as presented above.  A
   difference between link-local addresses and global addresses is that
   the identifiers used in global addresses MUST include a global scope
   unicast IPv4 address, while the identifiers used in link-local
   addresses MAY include a private IPv4 address.

5.  Specification of Clients, Servers, and Relays

   The Teredo service is realized by having clients interact with Teredo
   servers through the Teredo service protocol.  The clients will also
   receive IPv6 packets through Teredo relays.  The client behavior is
   specified in Section 5.2.

   The Teredo server is designed to be stateless.  It waits for Teredo
   requests and for IPv6 packets on the Teredo UDP port; it processes
   the requests by sending a response to the appropriate address and
   port; it forwards some Teredo IPv6 packets to the appropriate IPv4
   address and UDP port, or to native IPv6 peers of Teredo clients.  The
   precise behavior of the server is specified in Section 5.3.

   The Teredo relay advertises reachability of the Teredo service prefix
   over IPv6.  The scope of advertisement may be the entire Internet or
   a smaller subset such as an ISP network or an IPv6 site; it may even
   be as small as a single host in the case of "local relays".  The
   relay forwards Teredo IPv6 packets to the appropriate IPv4 address
   and UDP port.  The relay behavior is specified in Section 5.4.

   Teredo clients, servers, and relays must implement the sunset
   procedure defined in Section 5.5.

5.1.  Message Formats

5.1.1.  Teredo IPv6 Packet Encapsulation

   Teredo IPv6 packets are transmitted as UDP packets [RFC768] within
   IPv4 [RFC791].  The source and destination IP addresses and UDP ports
   take values that are specified in this section.  Packets can come in
   one of two formats, simple encapsulation and encapsulation with
   origin indication.

   When simple encapsulation is used, the packet will have a simple
   format, in which the IPv6 packet is carried as the payload of a UDP
   datagram:

   +------+-----+-------------+
   | IPv4 | UDP | IPv6 packet |
   +------+-----+-------------+

   When relaying some packets received from third parties, the server
   may insert an origin indication in the first bytes of the UDP
   payload:

```
+------+-----+------------------+-------------+
| IPv4 | UDP | Origin indication | IPv6 packet |
+------+-----+------------------+-------------+
```

The origin indication encapsulation is an 8-octet element, with the
following content:

```
+--------+--------+-----------------+
|  0x00  | 0x00   | Origin port #   |
+--------+--------+-----------------+
|  Origin IPv4 address              |
+-----------------------------------+
```

The first two octets of the origin indication are set to a null
value; this is used to discriminate between the simple encapsulation,
in which the first 4 bits of the packet contain the indication of the
IPv6 protocol, and the origin indication.

The following 16 bits contain the obfuscated value of the port number
from which the packet was received, in network byte order.  The next
32 bits contain the obfuscated IPv4 address from which the packet was
received, in network byte order.  In this format, both the original
"IPv4 address" and "UDP port" of the client are obfuscated.  Each bit
in the address and port number is reversed; this can be done by an
exclusive OR of the 16-bit port number with the hexadecimal value
0xFFFF, and an exclusive OR of the 32-bit address with the
hexadecimal value 0xFFFFFFFF.

For example, if the original UDP port number was 337 (hexadecimal
0151) and original IPv4 address was 1.2.3.4 (hexadecimal 01020304),
the origin indication would contain the value "0000FEAEFEFDFCFB".

When exchanging Router Solicitation (RS) and Router Advertisement
(RA) messages between a client and its server, the packets may
include an authentication parameter:

```
+------+-----+----------------+-------------+
| IPv4 | UDP | Authentication | IPv6 packet |
+------+-----+----------------+-------------+
```

The authentication encapsulation is a variable-length element,
containing a client identifier, an authentication value, a nonce
value, and a confirmation byte.

```
+--------+--------+--------+--------+
|  0x00  | 0x01   | ID-len | AU-len |
+--------+--------+--------+--------+
|  Client identifier (ID-len       |
+----------------+-----------------+
|  octets)       |  Authentication |
+----------------+--------+--------+
| value (AU-len octets)   | Nonce  |
+-------------------------+--------+
| value (8 octets)                 |
+-------------------------+--------+
|                         | Conf.  |
+-------------------------+--------+
```

The first octet of the authentication encapsulation is set to a null
value, and the second octet is set to the value 1; this enables
differentiation from IPv6 packets and from origin information
indication encapsulation.  The third octet indicates the length in
bytes of the client identifier; the fourth octet indicates the length
in bytes of the authentication value.  The computation of the
authentication value is specified in [Section 5.2.2](#). The
authentication value is followed by an 8-octet nonce, and by a
confirmation byte.

Both ID-len and AU-len can be set to null values if the server does
not require an explicit authentication of the client.

Authentication and origin indication encapsulations may sometimes be
combined, for example, in the RA responses sent by the server.  In
this case, the authentication encapsulation MUST be the first element
in the UDP payload:

```
+------+-----+----------------+--------+-------------+
| IPv4 | UDP | Authentication | Origin | IPv6 packet |
+------+-----+----------------+--------+-------------+
```

### [5.1.2](#).  Maximum Transmission Unit

Since Teredo uses UDP as an underlying transport, a Teredo Maximum
Transmission Unit (MTU) could potentially be as large as the payload
of the largest valid UDP datagram (65507 bytes).  However, since
Teredo packets can travel on unpredictable paths over the Internet,
it is best to contain this MTU to a small size, in order to minimize
the effect of IPv4 packet fragmentation and reassembly.  The default
link MTU assumed by a host, and the link MTU supplied by a Teredo
server during router advertisement SHOULD normally be set to the
minimum IPv6 MTU size of 1280 bytes [[RFC2460](#)].

Teredo implementations SHOULD NOT set the Don't Fragment (DF) bit of
the encapsulating IPv4 header.

## 5.2.  Teredo Client Specification

Before using the Teredo service, the client must be configured with:

- the IPv4 address of a server.
- a secondary IPv4 address of that server.

If secure discovery is required, the client must also be configured
with:

- a client identifier,
- a secret value, shared with the server,
- an authentication algorithm, shared with the server.

A Teredo client expects to exchange IPv6 packets through a UDP port,
the Teredo service port.  To avoid problems when operating behind a
"port conserving" NAT, different clients operating behind the same
NAT should use different service port numbers.  This can be achieved
through explicit configuration or, in the absence of configuration,
by picking the service port number at random.

The client will maintain the following variables that reflect the
state of the Teredo service:

- Teredo connectivity status,
- Mapped address and port number associated with the Teredo service
  port,
- Teredo IPv6 prefix associated with the Teredo service port,
- Teredo IPv6 address or addresses derived from the prefix,
- Link local address,
- Date and time of the last interaction with the Teredo server,
- Teredo Refresh Interval,
- Randomized Refresh Interval,
- List of recent Teredo peers.

Before sending any packets, the client must perform the Teredo
qualification procedure, which determines the Teredo connectivity
status, the mapped address and port number, and the Teredo IPv6
prefix.  It should then perform the cone NAT determination procedure,
which determines the cone NAT status and may alter the value of the
prefix.  If the qualification is successful, the client may use the
Teredo service port to transmit and receive IPv6 packets, according
to the transmission and reception procedures.  These procedures use
the "list of recent peers".  For each peer, the list contains:

- The IPv6 address of the peer,
- The mapped IPv4 address and mapped UDP port of the peer,
- The status of the mapped address, i.e., trusted or not,
- The value of the last nonce sent to the peer,
- The date and time of the last reception from the peer,
- The date and time of the last transmission to the peer,
- The number of bubbles transmitted to the peer.

The list of peers is used to enable the transmission of IPv6 packets
by using a "direct path" for the IPv6 packets.  The list of peers
could grow over time.  Clients should implement a list management
strategy, for example, deleting the least recently used entries.
Clients should make sure that the list has a sufficient size, to
avoid unnecessary exchanges of bubbles.

The client must regularly perform the maintenance procedure in order
to guarantee that the Teredo service port remains usable.  The need
to use this procedure or not depends on the delay since the last
interaction with the Teredo server.  The refresh procedure takes as a
parameter the "Teredo refresh interval".  This parameter is initially
set to 30 seconds; it can be updated as a result of the optional
"interval determination procedure".  The randomized refresh interval
is set to a value randomly chosen between 75% and 100% of the refresh
interval.

In order to avoid triangle routing for stations that are located
behind the same NAT, the Teredo clients MAY use the optional local
client discovery procedure defined in Section 5.2.8. Using this
procedure will also enhance connectivity when the NAT cannot do
"hairpin" routing, i.e., cannot redirect a packet sent from one
internal host to the mapped address and port of another internal
host.

## 5.2.1.  Qualification Procedure

The purposes of the qualification procedure are to establish the
status of the local IPv4 connection and to determine the Teredo IPv6
client prefix of the local Teredo interface.  The procedure starts
when the service is in the "initial" state, and it results in a
"qualified" state if successful, and in an "off-line" state if
unsuccessful.

```
             /---------\
             | Initial |
             \---------/
                  |
             +----+----------+
             | Set ConeBit=1 |
             +----+----------+
                  |
                  +<---------------------------------------------+
                  |                                              |
             +----+----+                                         |
             | Start   |<------+                                 |
             +----+----+       |                   +----------+----+
                  |            |                   | Set ConeBit=0 |
                  v            |                   +----------+----+
              /---------\ Timer | N                            ^
              |Starting |-------+ attempts /----------------\Yes|
              \---------/----------------->| ConeBit == 1 ? |---+
                  | Response                \----------------/
                  |                            | No
                  V                            V
         /---------------\ Yes          /----------\
         | ConeBit == 1? |-----+        | Off line |
         \---------------/     |        \----------/
            No |               v
               |         /----------\
               |         | Cone NAT |
          +-----+-----+   \----------/
          | New Server|
          +-----+-----+
                  |
             +----+----+
             | Start   |<------+
             +----+----+       |
                  |            |
                  v            |
              /---------\ Timer |
              |Starting |-------+ N attempts /----------\
              \---------/------------------->| Off line |
                  | Response                 \----------/
                  |
                  V
```
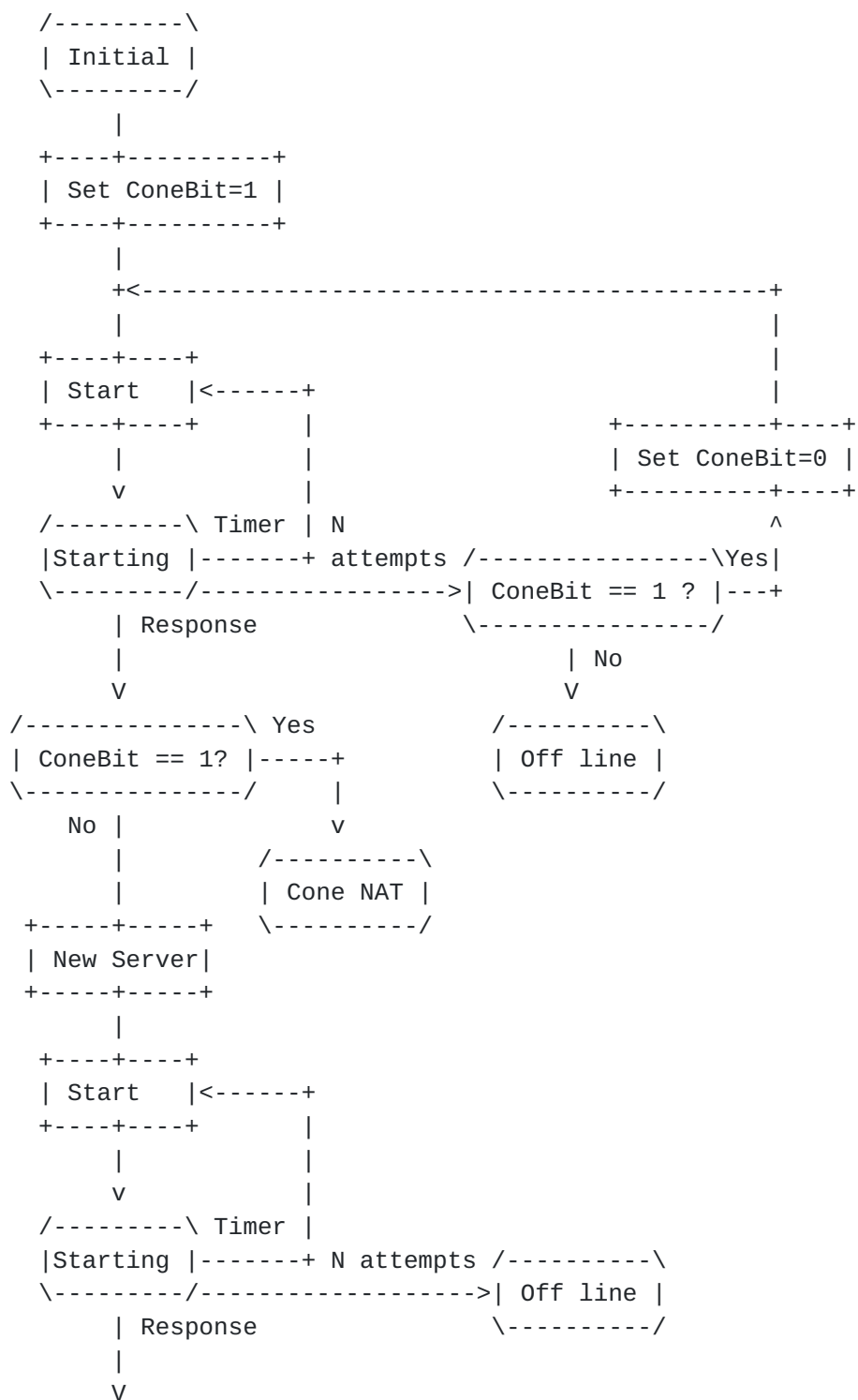
```
      /------------\ No      /---------------\
     | Same port? |-------->| Symmetric NAT |
      \------------/         \---------------/
          | Yes
          V
   /----------------------\
  | Restricted Cone NAT  |
   \----------------------/
```

Initially, the Teredo connectivity status is set to "Initial".

When the interface is initialized, the system first performs the
"start action" by sending a Router Solicitation message, as defined
in [RFC2461].  The client picks a link-local address and uses it as
the IPv6 source of the message; the cone bit in the address is set to
1 (see Section 4 for the address format); the IPv6 destination of the
RS is the all-routers multicast address; the packet will be sent over
UDP from the service port to the Teredo server's IPv4 address and
Teredo UDP port.  The connectivity status moves then to "Starting".

In the starting state, the client waits for a router advertisement
from the Teredo server.  If no response comes within a time-out T,
the client should repeat the start action, by resending the Router
Solicitation message.  If no response has arrived after N
repetitions, the client concludes that it is not behind a cone NAT.
It sets the cone bit to 0, and repeats the procedure.  If after N
other timer expirations and retransmissions there is still no
response, the client concludes that it cannot use UDP, and that the
Teredo service is not available; the status is set to "Off-line".  In
accordance with [RFC2461], the default time-out value is set to T=4
seconds, and the maximum number of repetitions is set to N=3.

If a response arrives, the client checks that the response contains
an origin indication and a valid router advertisement as defined in
[RFC2461], that the IPv6 destination address is equal to the link-
local address used in the router solicitation, and that the router
advertisement contains exactly one advertised Prefix Information
option.  This prefix should be a valid Teredo IPv6 server prefix: the
first 32 bits should contain the global Teredo IPv6 service prefix,
and the next 32 bits should contain the server's IPv4 address.  If
this is the case, the client learns the Teredo mapped address and
Teredo mapped port from the origin indication.  The IPv6 source
address of the Router Advertisement is a link-local server address of
the Teredo server.  (Responses that are not valid advertisements are
simply discarded.)

If the client has received an RA with the cone bit in the IPv6
destination address set to 1, it is behind a cone NAT and is fully
qualified.  If the RA is received with the cone bit set to 0, the
client does not know whether the local NAT is restricted or
symmetric.  The client selects the secondary IPv4 server address, and
repeats the procedure, the cone bit remaining to the value zero.  If
the client does not receive a response, it detects that the service
is not usable.  If the client receives a response, it compares the
mapped address and mapped port in this second response to the first
received values.  If the values are different, the client detects a
symmetric NAT: it cannot use the Teredo service.  If the values are
the same, the client detects a port-restricted or restricted cone
NAT: the client is qualified to use the service.  (Teredo operates
the same way for restricted and port-restricted NAT.)

If the client is qualified, it builds a Teredo IPv6 address using the
Teredo IPv6 server prefix learned from the RA and the obfuscated
values of the UDP port and IPv4 address learned from the origin
indication.  The cone bit should be set to the value used to receive
the RA, i.e., 1 if the client is behind a cone NAT, 0 otherwise.  The
client can start using the Teredo service.

## 5.2.2.  Secure Qualification

The client may be required to perform secured qualification.  The
client will perform exactly the algorithm described in Section 5.2.1,
but it will incorporate an authentication encapsulation in the UDP
packet carrying the router solicitation message, and it will verify
the presence of a valid authentication parameter in the UDP message
that carries the router advertisement provided by the sender.

In these packets, the nonce value is chosen by the client, and is
repeated in the response from the server; the client identifier is a
value with which the client was configured.

A first level of protection is provided by just checking that the
value of the nonce in the response matches the value initially sent
by the client.  If they don't match, the packet MUST be discarded.
If no other protection is used, the authentication payload does not
contain any identifier or authentication field; the ID-len and AU-len
fields are set to a null value.  When stronger protection is
required, the authentication payload contains the identifier and
location fields, as explained in the following paragraphs.

The confirmation byte is set to 0 by the client.  A null value
returned by the server indicates that the client's key is still
valid; a non-null value indicates that the client should obtain a new
key.

When stronger authentication is provided, the client and the server
are provisioned with a client identifier, a shared secret, and the
identification of an authentication algorithm.  Before transmission,
the authentication value is computed according to the specified
algorithm; on reception, the same algorithm is used to compute a
target value from the content of the receive packet.  The receiver
deems the authentication successful if the two values match.  If they
don't, the packet MUST be discarded.

To maximize interoperability, this specification defines a default
algorithm in which the authentication value is computed according the
HMAC specification [RFC2104] and the SHA1 function [FIPS-180].
Clients and servers may agree to use HMAC combined with a different
function, or to use a different algorithm altogether, such as for
example AES-XCBC-MAC-96 [RFC3566].

The default authentication algorithm is based on the HMAC algorithm
according to the following specifications:

- the hash function shall be the SHA1 function [FIPS-180].
- the secret value shall be the shared secret with which the client
  was configured.

The clear text to be protected includes:

- the nonce value,
- the confirmation byte,
- the origin indication encapsulation, if it is present,
- the IPv6 packet.

The HMAC procedure is applied to the concatenation of these four
components, without any additional padding.

## 5.2.3.  Packet Reception

The Teredo client receives packets over the Teredo interface.  The
role of the packet reception procedure, besides receiving packets, is
to maintain the date and time of the last interaction with the Teredo
server and the "list of recent peers".

When a UDP packet is received over the Teredo service port, the
Teredo client checks that it is encoded according to the packet
encoding rules defined in Section 5.1.1, and that it contains either
a valid IPv6 packet or the combination of a valid origin indication
encapsulation and a valid IPv6 packet, possibly protected by a valid
authentication encapsulation.  If this is not the case, the packet is
silently discarded.

An IPv6 packet is deemed valid if it conforms to [RFC2460]: the
protocol identifier should indicate an IPv6 packet and the payload
length should be consistent with the length of the UDP datagram in
which the packet is encapsulated.  In addition, the client should
check that the IPv6 destination address correspond to its own Teredo
address.

Then, the Teredo client examines the IPv4 source address and UDP port
number from which the packet is received.  If these values match the
IPv4 address of the server and the Teredo port, the client updates
the "date and time of the last interaction with the Teredo server" to
the current date and time; if an origin indication is present, the
client should perform the "direct IPv6 connectivity test" described
in Section 5.2.9.

If the IPv4 source address and UDP port number are different from the
IPv4 address of the server and the Teredo port, the client examines
the IPv6 source address of the packet:

1) If there is an entry for the source IPv6 address in the list of
peers whose status is trusted, the client compares the mapped IPv4
address and mapped port in the entry with the source IPv4 address and
source port of the packet.  If the values match, the packet is
accepted; the date and time of the last reception from the peer is
updated.

2) If there is an entry for the source IPv6 address in the list of
peers whose status is not trusted, the client checks whether the
packet is an ICMPv6 echo reply.  If this is the case, and if the
ICMPv6 data of the reply matches the nonce stored in the peer entry,
the packet should be accepted; the status of the entry should be
changed to "trusted", the mapped IPv4 and mapped port in the entry
should be set to the source IPv4 address and source port from which
the packet was received, and the date and time of the last reception
from the peer should be updated.  Any packet queued for this IPv6
peer (as specified in Section 5.2.4) should be de-queued and
forwarded to the newly learned IPv4 address and UDP port.

3) If the source IPv6 address is a Teredo address, the client
compares the mapped IPv4 address and mapped port in the source
address with the source IPv4 address and source port of the packet.
If the values match, the client MUST create a peer entry for the IPv6
source address in the list of peers; it should update the entry if
one already existed; the mapped IPv4 address and mapped port in the
entry should be set to the value from which the packet was received,
and the status should be set to "trusted".  If a new entry is
created, the last transmission date is set to 30 seconds before the
current date, and the number of bubbles to zero.  If the packet is a

bubble, it should be discarded after this processing; otherwise, the
packet should be accepted.  In all cases, the client must de-queue
and forward any packet queued for that destination.

4) If the IPv4 destination address through which the packet was
received is the Teredo IPv4 Discovery Address, the source address is
a valid Teredo address, and the destination address is the "all nodes
on link" multicast address, the packet should be treated as a local
discovery bubble.  If no local entry already existed for the source
address, a new one is created, but its status is set to "not
trusted".  The client SHOULD reply with a unicast Teredo bubble, sent
to the source IPv4 address and source port of the local discovery
bubble; the IPv6 source address of the bubble will be set to local
Teredo IPv6 address; the IPv6 destination address of the bubble
should be set to the IPv6 source address of the local discovery
bubble.  (Clients that do not implement the optional local discovery
procedure will not process local discovery bubbles.)

5) If the source IPv6 address is a Teredo address, and the mapped
IPv4 address and mapped port in the source address do not match the
source IPv4 address and source port of the packet, the client checks
whether there is an existing "local" entry for that IPv6 address.  If
there is such an entry, and if the local IPv4 address and local port
indicated in that entry match the source IPv4 address and source

port of the packet, the client updates the "local" entry, whose
status should be set to "trusted".  If the packet is a bubble, it
should be discarded after this processing; otherwise, the packet
should be accepted.  In all cases, the client must de-queue and
forward any packet queued for that destination.

6) In the other cases, the packet may be accepted, but the client
should be conscious that the source address may be spoofed; before
processing the packet, the client should perform the "direct IPv6
connectivity test" described in Section 5.2.9.

Whatever the IPv4 source address and UDP source port, the client that
receives an IPv6 packet MAY send a Teredo bubble towards that target,
as specified in Section 5.2.6.

## 5.2.4.  Packet Transmission

When a Teredo client has to transmit a packet over a Teredo
interface, it examines the destination IPv6 address.  The client
checks first if there is an entry for this IPv6 address in the list
of recent Teredo peers, and if the entry is still valid: an entry
associated with a local peer is valid if the last reception date and
time associated with that list entry is less that 30 seconds from the

current time; an entry associated with a non-local peer is valid if
the last reception date and time associated with that list entry is
less that 30 seconds from the current time.  (Local peer entries can
only be present if the client uses the local discovery procedure
discussed in Section 5.2.8.)

The client then performs the following:

1) If there is an entry for that IPv6 address in the list of peers,
and if the status of the entry is set to "trusted", the IPv6 packet
should be sent over UDP to the IPv4 address and UDP port specified in
the entry.  The client updates the date of last transmission in the
peer entry.

2) If the destination is not a Teredo IPv6 address, the packet is
queued, and the client performs the "direct IPv6 connectivity test"
described in Section 5.2.9. The packet will be de-queued and
forwarded if this procedure completes successfully.  If the direct
IPv6 connectivity test fails to complete within a 2-second time-out,
it should be repeated up to 3 times.

3) If the destination is the Teredo IPv6 address of a local peer
(i.e., a Teredo address from which a local discovery bubble has been
received in the last 600 seconds), the packet is queued.  The client
sends a unicast Teredo bubble to the local IPv4 address and local
port specified in the entry, and a local Teredo bubble to the Teredo
IPv4 discovery address.

4) If the destination is a Teredo IPv6 address in which the cone bit
is set to 1, the packet is sent over UDP to the mapped IPv4 address
and mapped UDP port extracted from that IPv6 address.

5) If the destination is a Teredo IPv6 address in which the cone bit
is set to 0, the packet is queued.  If the client is not located
behind a cone NAT, it sends a direct bubble to the Teredo
destination, i.e., to the mapped IP address and mapped port of the
destination.  In all cases, the client sends an indirect bubble to
the Teredo destination, sending it over UDP to the server address and
to the Teredo port.  The packet will be de-queued and forwarded when
the client receives a bubble or another packet directly from this
Teredo peer.  If no bubble is received within a 2-second time-out,
the bubble transmission should be repeated up to 3 times.

In cases 4 and 5, before sending a packet over UDP, the client MUST
check that the IPv4 destination address is in the format of a global
unicast address; if this is not the case, the packet MUST be silently

discarded.  (Note that a packet can legitimately be sent to a non-
global unicast address in case 1, as a result of the local discovery
procedure.)

The global unicast address check is designed to thwart a number of
possible attacks in which an attacker tries to use a Teredo host to
attack either a single local IPv4 target or a set of such targets.
For the purpose of this specification, and IPv4 address is deemed to
be a global unicast address if it does not belong to or match:

- the "local" subnet 0.0.0.0/8,
- the "loopback" subnet 127.0.0.0/8,
- the local addressing ranges 10.0.0.0/8,
- the local addressing ranges 172.16.0.0/12,
- the local addressing ranges 192.168.0.0/16,
- the link local block 169.254.0.0/16,
- the block reserved for 6to4 anycast addresses 192.88.99.0/24,
- the multicast address block 224.0.0.0/4,
- the "limited broadcast" destination address 255.255.255.255,
- the directed broadcast addresses corresponding to the subnets to
  which the host is attached.

A list of special-use IPv4 addresses is provided in [RFC3330].

For reliability reasons, clients MAY decide to ignore the value of
the cone bit in the flag, skip the "case 4" test and always perform
the "case 5", i.e., treat all Teredo peers as if they were located
behind non-cone NAT.  This will result in some increase in traffic,
but may avoid reliability issues if the determination of the NAT
status was for some reason erroneous.  For the same reason, clients
MAY also decide to always send a direct bubble in case 5, even if
they do not believe that they are located behind a non-cone NAT.

## 5.2.5.  Maintenance

The Teredo client must ensure that the mappings that it uses remain
valid.  It does so by checking that packets are regularly received
from the Teredo server.

At regular intervals, the client MUST check the "date and time of the
last interaction with the Teredo server" to ensure that at least one
packet has been received in the last Randomized Teredo Refresh
Interval.  If this is not the case, the client SHOULD send a router
solicitation message to the server, as specified in Section 5.2.1;
the client should use the same value of the cone bit that resulted in
the reception of an RA during the qualification procedure.

When the router advertisement is received, the client SHOULD check
its validity as specified in Section 5.2.1; invalid advertisements
are silently discarded.  If the advertisement is valid, the client
MUST check that the mapped address and port correspond to the current
Teredo address.  If this is not the case, the mapping has changed;
the client must mark the old address as invalid and start using the
new address.

### 5.2.6.  Sending Teredo Bubbles

The Teredo client may have to send a bubble towards another Teredo
client, either after a packet reception or after a transmission
attempt, as explained in Sections 5.2.3 and 5.2.4. There are two
kinds of bubbles: direct bubbles, which are sent directly to the
mapped IPv4 address and mapped UDP port of the peer, and indirect
bubbles, which are sent through the Teredo server of the peer.

When a Teredo client attempts to send a direct bubble, it extracts
the mapped IPv4 address and mapped UDP port from the Teredo IPv6
address of the target.  It then checks whether there is already an
entry for this IPv6 address in the current list of peers.  If there
is no entry, the client MUST create a new list entry for the address,
setting the last reception date and the last transmission date to 30
seconds before the current date, and the number of bubbles to zero.

When a Teredo client attempts to send an indirect bubble, it extracts
the Teredo server IPv4 address from the Teredo prefix of the IPv6
address of the target (different clients may be using different
servers); the bubble will be sent to that IPv4 address and the Teredo
UDP port.

Bubbles may be lost in transit, and it is reasonable to enhance the
reliability of the Teredo service by allowing multiple transmissions;
however, bubbles will also be lost systematically in certain NAT
configurations.  In order to strike a balance between reliability and
unnecessary retransmissions, we specify the following:

- The client MUST NOT send a bubble if the last transmission date
  and time is less than 2 seconds before the current date and time;

- The client MUST NOT send a bubble if it has already sent 4 bubbles
  to the peer in the last 300 seconds without receiving a direct
  response.

In the other cases, the client MAY proceed with the transmission of
the bubble.  When transmitting the bubble, the client MUST update the
last transmission date and time to that peer, and must also increment
the number of transmitted bubbles.

5.2.7.  **Optional Refresh Interval Determination Procedure**

   In addition to the regular client resources described in the
   beginning of this section, the refresh interval determination
   procedure uses an additional UDP port, the Teredo secondary port, and
   the following variables:

   - Teredo secondary connectivity status,
   - Mapped address and port number of the Teredo secondary port,
   - Teredo secondary IPv6 prefix associated with the secondary port,
   - Teredo secondary IPv6 address derived from this prefix,
   - Date and time of the last interaction on the secondary port,
   - Maximum Teredo Refresh Interval.
   - Candidate Teredo Refresh Interval.

   The secondary connectivity status, mapped address and prefix are
   determined by running the qualification procedure on the secondary
   port.  When the client uses the interval determination procedure, the
   qualification procedure MUST be run for the secondary port
   immediately after running it on the service port.  If the secondary
   qualification fails, the interval determination procedure will not be
   used, and the interval value will remain to the default value, 30
   seconds.  If the secondary qualification succeeds, the maximum
   refresh interval is set to 120 seconds, and the candidate Teredo
   refresh interval is set to 60 seconds, i.e., twice the Teredo refresh
   interval.  The procedure is then performed at regular intervals,
   until it concludes:

   1) wait until the candidate refresh interval is elapsed after the
      last interaction on the secondary port.

   2) send a Teredo bubble to the Teredo secondary IPv6 address, through
      the service port.

   3) wait for reception of the bubble on the secondary port.  If a
      timer of 2 seconds elapses without reception, repeat step 2 at
      most three times.  If there is still no reception, the candidate
      has failed; if there is a reception, the candidate has succeeded.

   4) if the candidate has succeeded, set the Teredo refresh interval to
      the candidate value, and set a new candidate value to the minimum
      of twice the new refresh interval, or the average of the refresh
      interval and the maximum refresh interval.

5) if the candidate has failed, set the maximum refresh interval to
   the candidate value.  If the current refresh interval is larger
   than or equal to 75% of the maximum, the determination procedure
   has concluded; otherwise, set a new candidate value to the average
   of the refresh interval and the maximum refresh interval.

6) if the procedure has not concluded, perform the maintenance
   procedure on the secondary port, which will reset the date and
   time of the last interaction on the secondary port, and may result
   in the allocation of a new Teredo secondary IPv6 address; this
   would not affect the values of the refresh interval, candidate
   interval, or maximum refresh interval.

The secondary port MUST NOT be used for any other purpose than the
interval determination procedure.  It should be closed when the
procedure ends.

## 5.2.8.  Optional Local Client Discovery Procedure

It is desirable to enable direct communication between Teredo clients
that are located behind the same NAT, without forcing a systematic
relay through a Teredo server.  It is hard to design a general
solution to this problem, but we can design a partial solution when
the Teredo clients are connected through IPv4 to the same link.

A Teredo client who wishes to enable local discovery SHOULD join the
IPv4 multicast group identified by Teredo IPv4 Discovery Address.
The client SHOULD wait for discovery bubbles to be received on the
Teredo IPv4 Discovery Address.  The client SHOULD send local
discovery bubbles to the Teredo IPv4 Discovery Address at random
intervals, uniformly distributed between 200 and 300 seconds.  A
local Teredo bubble has the following characteristics:

- IPv4 source address: the IPv4 address of the sender

- IPv4 destination address: the Teredo IPv4 Discovery Address

- IPv4 ttl: 1

- UDP source port: the Teredo service port of the sender

- UDP destination port: the Teredo UDP port

- UDP payload: a minimal IPv6 packet, as follows

- IPv6 source: the global Teredo IPv6 address of the sender

- IPv6 destination: the all-nodes on-link multicast address

- IPv6 payload type: 59 (No Next Header, as per [RFC2460])

- IPv6 payload length: 0

- IPv6 hop limit: 1

The local discovery procedure carries a denial of service risk, as
malevolent nodes could send fake bubbles to unsuspecting parties, and
thus capture the traffic originating from these parties.  The risk is
mitigated by the filtering rules described in Section 5.2.5, and also
by "link only" multicast scope of the Teredo IPv4 Discovery Address,
which implies that packets sent to this address will not be forwarded
across routers.

To benefit from the "link only multicast" protection, the clients
should silently discard all local discovery bubbles that are received
over a unicast address.  To further mitigate the denial of service
risk, the client MUST silently discard all local discovery bubbles
whose IPv6 source address is not a well-formed Teredo IPv6 address,
or whose IPv4 source address does not belong to the local IPv4
subnet; the client MAY decide to silently discard all local discovery
bubbles whose Teredo IPv6 address do not include the same mapped IPv4
address as its own.

If the bubble is accepted, the client checks whether there is an
entry in the list of recent peers that correspond to the mapped IPv4
address and mapped UDP port associated with the source IPv6 address
of the bubble.  If there is such an entry, the client MUST update the
local peer address and local peer port parameters to reflect the IPv4
source address and UDP source port of the bubble.  If there is no
entry, the client MUST create one, setting the local peer address and
local peer port parameters to reflect the IPv4 source address and UDP
source port of the bubble, the last reception date to the current
date and time, the last transmission date to 30 seconds before the
current date, and the number of bubbles to zero.  The state of the
entry is set to "not trusted".

Upon reception of a discovery bubble, clients reply with a unicast
bubble as specified in Section 5.2.3.

### 5.2.9.  Direct IPv6 Connectivity Test

The Teredo procedures are designed to enable direct connections
between a Teredo host and a Teredo relay.  Teredo hosts located
behind a cone NAT will receive packets directly from relays; other
Teredo hosts will learn the original addresses and UDP ports of third
parties through the local Teredo server.  In all of these cases,
there is a risk that the IPv6 address of the source will be spoofed

by a malevolent party.  Teredo hosts must make two decisions, whether
to accept the packet for local processing and whether to transmit
further packets to the IPv6 address through the newly

learned IPv4 address and UDP port.  The basic rule is that the hosts
should be generous in what they accept and careful in what they send.
Refusing to accept packets due to spoofing concerns would compromise
connectivity and should only be done when there is a near certainty
that the source address is spoofed.  On the other hand, sending
packets to the wrong address should be avoided.

When the client wants to send a packet to a native IPv6 node or a
6to4 node, it should check whether a valid peer entry already exists
for the IPv6 address of the destination.  If this is not the case,
the client will pick a random number (a nonce) and format an ICMPv6
Echo Request message whose source is the local Teredo address, whose
destination is the address of the IPv6 node, and whose Data field is
set to the nonce.  (It is recommended to use a random number at least
64 bits long.)  The nonce value and the date at which the packet was
sent will be documented in a provisional peer entry for the IPV6
destination.  The ICMPv6 packet will then be sent encapsulated in a
UDP packet destined to the Teredo server IPv4 address and to the
Teredo port.  The rules of Section 5.2.3 specify how the reply to
this packet will be processed.

## 5.2.10.  Working around symmetric NAT

The client procedures are designed to enable IPv6 connectivity
through the most common types of NAT, which are commonly called "cone
NAT" and "restricted cone NAT" [RFC3489].  Some NATs employ a
different design; they are often called "symmetric NAT".  The
qualification algorithm in Section 5.2.1 will not succeed when the
local NAT is a symmetric NAT.

In many cases, it is possible to work around the limitations of these
NATs by explicitly reserving a UDP port for Teredo service on a
client, using a function often called "DMZ" in the NAT's manual.
This port will become the "service port" used by the Teredo hosts.
The implementers of Teredo functions in hosts must make sure that the
value of the service port can be explicitly provisioned, so that the
user can provision the same value in the host and in the NAT.

The reservation procedure guarantees that the port mapping will
remain the same for all destinations.  After the explicit
reservation, the qualification algorithm in Section 5.2.1 will
succeed, and the Teredo client will behave as if behind a "cone NAT".

When different clients use Teredo behind a single symmetric NAT, each of these clients must reserve and use a different service port.

## 5.3.  Teredo Server Specification

The Teredo server is designed to be stateless.  The Teredo server waits for incoming UDP packets at the Teredo Port, using the IPv4 address that has been selected for the service.  In addition, the server is able to receive and transmit some packets using a different IPv4 address and a different port number.

The Teredo server acts as an IPv6 router.  As such, it will receive Router Solicitation messages, to which it will respond with Router Advertisement messages as explained in Section 5.3.2.  It may also receive other packets, for example, ICMPv6 messages and Teredo bubbles, which are processed according to the IPv6 specification.

By default, the routing functions of the Teredo server are limited. Teredo servers are expected to relay Teredo bubbles, ICMPv6 Echo requests, and ICMPv6 Echo replies, but they are not expected to relay other types of IPv6 packets.  Operators may, however, decide to combine the functions of "Teredo server" and "Teredo relay", as explained in Section 5.4.

## 5.3.1.  Processing of Teredo IPv6 Packets

Before processing the packet, the Teredo server MUST check the validity of the encapsulated IPv6 source address, the IPv4 source address, and the UDP source port:

1)  If the UDP content is not a well-formed Teredo IPv6 packet, as defined in Section 5.1.1, the packet MUST be silently discarded.

2)  If the UDP packet is not a Teredo bubble or an ICMPv6 message, it SHOULD be discarded.  (The packet may be processed if the Teredo server also operates as a Teredo relay, as explained in Section 5.4.)

3)  If the IPv4 source address is not in the format of a global unicast address, the packet MUST be silently discarded (see Section 5.2.4 for a definition of global unicast addresses).

4)  If the IPv6 source address is an IPv6 link-local address, the IPv6 destination address is the link-local scope all routers multicast address (FF02::2), and the packet contains an ICMPv6 Router Solicitation message, the packet MUST be accepted.  It MUST be discarded if the server requires secure qualification and the authentication encapsulation is absent or verification fails.

5)  If the IPv6 source address is a Teredo IPv6 address, and if the
IPv4 address and UDP port embedded in that address match the IPv4
source address and UDP source port, the packet SHOULD be accepted.

6)  If the IPv6 source address is not a Teredo IPv6 address, and if
the IPv6 destination address is a Teredo address allocated through
this server, the packet SHOULD be accepted.

7)  In all other cases, the packet MUST be silently discarded.

The Teredo server will then check the IPv6 destination address of the
encapsulated IPv6 packet:

If the IPv6 destination address is the link-local scope all routers
multicast address (FF02::2), or the link-local address of the server,
the Teredo server processes the packet; it may have to process Router
Solicitation messages and ICMPv6 Echo Request messages.

If the destination IPv6 address is not a global scope IPv6 address,
the packet MUST NOT be forwarded.

If the destination address is not a Teredo IPv6 address, the packet
should be relayed to the IPv6 Internet using regular IPv6 routing.

If the IPv6 destination address is a valid Teredo IPv6 address as
defined in Section 2.13, the Teredo Server MUST check that the IPv4
address derived from this IPv6 address is in the format of a global
unicast address; if this is not the case, the packet MUST be silently
discarded.

If the address is valid, the Teredo server encapsulates the IPv6
packet in a new UDP datagram, in which the following parameters are
set:

- The destination IPv4 address is derived from the IPv6 destination.

- The source IPv4 address is the Teredo server IPv4 address.

- The destination UDP port is derived from the IPv6 destination.

- The source UDP port is set to the Teredo UDP Port.

If the destination IPv6 address is a Teredo client whose address is
serviced by this specific server, the server should insert an origin
indication in the first bytes of the UDP payload, as specified in
Section 5.1.1.  (To verify that the client is served by this server,
the server compares bits 32-63 of the client's Teredo IPv6 address to
the server's IPv4 address.)

5.3.2.  **Processing of Router Solicitations**

   When the Teredo server receives a Router Solicitation message (RS,
   [RFC2461]), it retains the IPv4 address and UDP port from which the
   solicitation was received; these become the Teredo mapped address and
   Teredo mapped port of the client.  The router uses these values to
   compose the origin indication encapsulation that will be sent with
   the response to the solicitation.

   The Teredo server responds to the router solicitation by sending a
   Router Advertisement message [RFC2461].  The router advertisement
   MUST advertise the Teredo IPv6 prefix composed from the service

   prefix and the server's IPv4 address.  The IPv6 source address should
   be set to a Teredo link-local server address associated to the local
   interface; this address is derived from the IPv4 address of the
   server and from the Teredo port, as specified in Section 4; the cone
   bit is set to 1.  The IPv6 destination address is set to the IPv6
   source address of the RS.  The Router Advertisement message must be
   sent over UDP to the Teredo mapped address and Teredo mapped port of
   the client; the IPv4 source address and UDP source port should be set
   to the server's IPv4 address and Teredo Port.  If the cone bit of the
   client's IPv6 address is set to 1, the RA must be sent from a
   different IPv4 source address than the server address over which the
   RS was received; if the cone bit is set to zero, the response must be
   sent back from the same address.

   Before sending the packet, the Teredo server MUST check that the IPv4
   destination address is in the format of a global unicast address; if
   this is not the case, the packet MUST be silently discarded (see
   Section 5.2.4 for a definition of global unicast addresses).

   If secure qualification is required, the server MUST insert a valid
   authentication parameter in the UDP packet carrying the router
   advertisement.  The client identifier and the nonce value used in the
   authentication parameter MUST be the same identifier and nonce as
   received in the router solicitation.  The confirmation byte MUST be
   set to zero if the client identifier is still valid, and a non-null
   value otherwise; the authentication value SHOULD be computed using
   the secret that corresponds to the client identifier.

5.4.  **Teredo Relay Specification**

   Teredo relays are IPv6 routers that advertise reachability of the
   Teredo service IPv6 prefix through the IPv6 routing protocols.  (A
   minimal Teredo relay may serve just a local host, and would not
   advertise the prefix beyond this host.)  Teredo relays will receive
   IPv6 packets bound to Teredo clients.  Teredo relays should be able

to receive packets sent over IPv4 and UDP by Teredo clients; they may
apply filtering rules, e.g., only accept packets from Teredo clients
if they have previously sent traffic to these Teredo clients.

The receiving and sending rules used by Teredo relays are very
similar to those of Teredo clients.  Teredo relays must use a Teredo
service port to transmit packets to Teredo clients; they must
maintain a "list of peers", identical to the list of peers maintained
by Teredo clients.

### 5.4.1.  Transmission by Relays to Teredo Clients

When a Teredo relay has to transmit a packet to a Teredo client, it
examines the destination IPv6 address.  By definition, the Teredo
relays will only send over UDP IPv6 packets whose IPv6 destination
address is a valid Teredo IPv6 address.

Before processing these packets, the Teredo Relay MUST check that the
IPv4 destination address embedded in the Teredo IPv6 address is in
the format of a global unicast address; if this is not the case, the
packet MUST be silently discarded (see Section 5.2.4 for a definition
of global unicast addresses).

The relay then checks if there is an entry for this IPv6 address in
the list of recent Teredo peers, and if the entry is still valid.
The relay then performs the following:

1) If there is an entry for that IPv6 address in the list of peers,
and if the status of the entry is set to "trusted", the IPv6 packet
should be sent over UDP to the mapped IPv4 address and mapped UDP
port of the entry.  The relay updates the date of last transmission
in the peer entry.

2) If there is no trusted entry in the list of peers, and if the
destination is a Teredo IPv6 address in which the cone bit is set to
1, the packet is sent over UDP to the mapped IPv4 address and mapped
UDP port extracted from that IPv6 address.

3) If there is no trusted entry in the list of peers, and if the
destination is a Teredo IPv6 address in which the cone bit is set to
0, the Teredo relay creates a bubble whose source address is set to a
local IPv6 address, and whose destination address is set to the
Teredo IPv6 address of the packet's destination.  The bubble is sent
to the server address corresponding to the Teredo destination.  The
entry becomes trusted when a bubble or another packet is received
from this IPv6 address; if no such packet is received before a time-
out of 2 seconds, the Teredo relay may repeat the bubble, up to three
times.  If the relay fails to receive a bubble after these

repetitions, the entry is removed from the list of peers.  The relay
MAY queue packets bound to untrusted entries; the queued packets
SHOULD be de-queued and forwarded when the entry becomes trusted;
they SHOULD be deleted if the entry is deleted.  To avoid denial of
service attacks, the relays SHOULD limit the number of packets in
such queues.

In cases 2 and 3, the Teredo relay should create a peer entry for the
IPv6 address; the entry status is marked as trusted in case 2 (cone
NAT) and not trusted in case 3.  In case 3, if the Teredo relay
happens to be located behind a non-cone NAT, it should also send a
bubble directly to the mapped IPv4 address and mapped port number of
the Teredo destination.  This will "open the path" for the return
bubble from the Teredo client.

For reliability reasons, relays MAY decide to ignore the value of the
cone bit in the flag, and always perform the "case 3", i.e., treat
all Teredo peers as if they were located behind a non-cone NAT.  This
will result in some increase in traffic, but may avoid

reliability issues if the determination of the NAT status was for
some reason erroneous.  For the same reason, relays MAY also decide
to always send a direct bubble to the mapped IPv4 address and mapped
port number of the Teredo destination, even if they do not believe
that they are located behind a non-cone NAT.

## 5.4.2.  Reception from Teredo Clients

The Teredo relay may receive packets from Teredo clients; the packets
should normally only be sent by clients to which the relay previously
transmitted packets, i.e., clients whose IPv6 address is present in
the list of peers.  Relays, like clients, use the packet reception
procedure to maintain the date and time of the last interaction with
the Teredo server and the "list of recent peers".

When a UDP packet is received over the Teredo service port, the
Teredo relay checks that it contains a valid IPv6 packet as specified
in [RFC2460].  If this is not the case, the packet is silently
discarded.

Then, the Teredo relay examines whether the IPv6 source address is a
valid Teredo address, and if the mapped IPv4 address and mapped port
match the IPv4 source address and port number from which the packet
is received.  If this is not the case, the packet is silently
discarded.

The Teredo relay then examines whether there is an entry for the IPv6
source address in the list of recent peers.  If this is not the case,

the packet may be silently discarded.  If this is the case, the entry
status is set to "trusted"; the relay updates the "date and time of
the last interaction" to the current date and time.

Finally, the relay examines the destination IPv6 address.  If the
destination belongs to a range of IPv6 addresses served by the relay,
the packet SHOULD be accepted and forwarded to the destination.  In
the other cases, the packet SHOULD be silently discarded.

### 5.4.3.  Difference between Teredo Relays and Teredo Servers

Because Teredo servers can relay Teredo packets over IPv6, all Teredo
servers must be capable of behaving as Teredo relays.  There is,
however, no requirement that Teredo relays behave as Teredo servers.

The dual role of server and relays implies an additional complexity
for the programming of servers: the processing of incoming packets
should be a combination of the server processing rules defined in
Section 5.3.1, and the relay processing rules defined in Section
5.4.2.  (Section 5.3 only specifies the rules implemented by a pure
server, not a combination relay+server.)

### 5.5.  Implementation of Automatic Sunset

Teredo is designed as an interim transition mechanism, and it is
important that it should not be used any longer than necessary.  The
"sunset" procedure will be implemented by Teredo clients, servers,
and relays, as specified in this section.

The Teredo-capable nodes MUST NOT behave as Teredo clients if they
already have IPv6 connectivity through any other means, such as
native IPv6 connectivity.  In particular, nodes that have a global
IPv4 address SHOULD obtain connectivity through the 6to4 service
rather than through the Teredo service.  The classic reason why a
node that does not need connectivity would still enable the Teredo
service is to guarantee good performance when interacting with Teredo
clients; however, a Teredo-capable node that has IPv4 connectivity
and that has obtained IPv6 connectivity outside the Teredo service
MAY decide to behave as a Teredo relay, and still obtain good
performance when interacting with Teredo clients.

The Teredo servers are expected to participate in the sunset
procedure by announcing a date at which they will stop providing the
service.  This date depends on the availability of alternative
solutions to their clients, such as "dual-mode" gateways that behave
simultaneously as IPv4 NATs and IPv6 routers.  Most Teredo servers
will not be expected to operate more than a few years.  Teredo relays
are expected to have the same life span as Teredo servers.

6.  Further Study, Use of Teredo to Implement a Tunnel Service

   Teredo defines a NAT traversal solution that can be provided using
   very little resource at the server.  Ongoing IETF discussions have
   outlined the need for both a solution like Teredo and a more
   controlled NAT traversal solution, using configured tunnels to a
   service provider [RFC3904].  This section provides a tentative
   analysis of how Teredo could be extended to also support a configured
   tunnel service.

   It may be possible to design a tunnel server protocol that is
   compatible with Teredo, in the sense that the same client could be
   used either in the Teredo service or with a tunnel service.  In fact,
   this could be done by configuring the client with:

   - The IPv4 address of a Teredo server that acts as a tunnel broker
   - A client identifier
   - A shared secret with that server
   - An agreed-upon authentication algorithm.

   The Teredo client would use the secure qualification procedure, as
   specified in Section 5.2.2. Instead of returning a Teredo prefix in
   the router advertisement, the server would return a globally routable
   IPv6 prefix; this prefix could be permanently assigned to the client,
   which would provide the client with a stable address.  The server
   would have to keep state, i.e., memorize the association between the
   prefix assigned to the client and the mapped IPv4 address and mapped
   UDP port of the client.

   The Teredo server would advertise reachability of the client prefix
   to the IPv6 Internet.  Any packet bound to that prefix would be
   transmitted to the mapped IPv4 address and mapped UDP port of the
   client.

   The Teredo client, when it receives the prefix, would notice that
   this prefix is a global IPv6 prefix, not in the form of a Teredo
   prefix.  The client would at that point recognize that it should
   operate in tunnel mode.  A client that operates in tunnel mode would
   execute a much simpler transmission procedure: it would forward any
   packet sent to the Teredo interface to the IPv4 address and Teredo
   UDP port of the server.

   The Teredo client would have to perform the maintenance procedure
   described in Section 5.2.5. The server would receive the router
   solicitation, and could notice a possible change of mapped IPv4
   address and mapped UDP port that could result from the
   reconfiguration of the mappings inside the NAT.  The server should
   continue advertising the same IPv6 prefix to the client, and should

   update the mapped IPv4 address and mapped UDP port associated to this
   prefix, if necessary.

   There is as yet no consensus that a tunnel-mode extension to Teredo
   should be developed.  This section is only intended to provide
   suggestions to the future developers of such services.  Many details
   would probably have to be worked out before a tunnel-mode extension
   would be agreed upon.

## 7.  Security Considerations

   The main objective of Teredo is to provide nodes located behind a NAT
   with a globally routable IPv6 address.  The Teredo nodes can use IP
   security (IPsec) services such as Internet Key Exchange (IKE),
   Authentication Header (AH), or Encapsulation Security Payload (ESP)
   [RFC4306, RFC4302, RFC4303], without the configuration restrictions
   still present in "Negotiation of NAT-Traversal in the IKE" [RFC3947].
   As such, we can argue that the service has a positive effect on
   network security.  However, the security analysis must also envisage
   the negative effects of the Teredo services, which we can group in
   four categories: security risks of directly connecting a node to the
   IPv6 Internet, spoofing of Teredo servers to enable a man-in-the-
   middle attack, potential attacks aimed at denying the Teredo service
   to a Teredo client, and denial of service attacks against non-Teredo
   participating nodes that would be enabled by the Teredo service.

   In the following, we review in detail these four types of issues, and
   we present mitigating strategies for each of them.

## 7.1.  Opening a Hole in the NAT

   The very purpose of the Teredo service is to make a machine reachable
   through IPv6.  By definition, the machine using the service will give
   up whatever firewall service was available in the NAT box, however
   limited this service may be [RFC2993].  The services that listen to
   the Teredo IPv6 address will become the potential target of attacks
   from the entire IPv6 Internet.  This may sound scary, but there are
   three mitigating factors.

   The first mitigating factor is the possibility to restrict some
   services to only accept traffic from local neighbors, e.g., using
   link-local addresses.  Teredo does not support communication using
   link-local addresses.  This implies that link-local services will not
   be accessed through Teredo, and will be restricted to whatever other
   IPv6 connectivity may be available, e.g., direct traffic with
   neighbors on the local link, behind the NAT.

The second mitigating factor is the possible use of a "local
firewall" solution, i.e., a piece of software that performs locally
the kind of inspection and filtering that is otherwise performed in a
perimeter firewall.  Using such software is recommended.

The third mitigating factor is the availability of IP security
(IPsec) services such as IKE, AH, or ESP [RFC4306, RFC4302, RFC4303].
Using these services in conjunction with Teredo is a good policy, as
it will protect the client from possible attacks in intermediate
servers such as the NAT, the Teredo server, or the Teredo relay.
(However, these services can be used only if the parties in the
communication can negotiate a key, which requires agreeing on some
credentials; this is known to be a hard problem.)

## 7.2.  Using the Teredo Service for a Man-in-the-Middle Attack

The goal of the Teredo service is to provide hosts located behind a
NAT with a globally reachable IPv6 address.  There is a possible
class of attacks against this service in which an attacker somehow
intercepts the router solicitation, responds with a spoofed router
advertisement, and provides a Teredo client with an incorrect
address.  The attacker may have one of two objectives: it may try to
deny service to the Teredo client by providing it with an address
that is in fact unreachable, or it may try to insert itself as a
relay for all client communications, effectively enabling a variety
of "man-in-the-middle" attack.

### 7.2.1.  Attacker Spoofing the Teredo Server

The simple nonce verification procedure described in Section 5.2.2
provides a first level of protection against attacks in which a third
party tries to spoof the server.  In practice, the nonce procedure
can be defeated only if the attacker is "on path".

If client and server share a secret and agree on an authentication
algorithm, the secure qualification procedure described in Section
5.2.2 provides further protection.  To defeat this protection, the
attacker could try to obtain a copy of the secret shared between
client and server.  The most likely way to obtain the shared secret
is to listen to the traffic and mount an offline dictionary attack;
to protect against this attack, the secret shared between client and
server should contain sufficient entropy.  (This probably requires
some automated procedure for provisioning the shared secret and the
algorithm.)

If the shared secret contains sufficient entropy, the attacker would
have to defeat the one-way function used to compute the
authentication value.  This specification suggests a default

algorithm combining HMAC and MD5.  If the protection afforded by MD5
was not deemed sufficient, clients and servers can agree to use a
different algorithm, e.g., SHA1.

Another way to defeat the protection afforded by the authentication
procedure is to mount a complex attack, as follows:

1) Client prepares router solicitation, including authentication
encapsulation.

2) Attacker intercepts the solicitation, and somehow manages to
prevent it from reaching the server, for example, by mounting a
short-duration DoS attack against the server.

3) Attacker replaces the source IPv4 address and source UDP port of
the request by one of its own addresses and port, and forwards the
modified request to the server.

4) Server dutifully notes the IPv4 address from which the packet is
received, verifies that the Authentication encapsulation is correct,
prepares a router advertisement, signs it, and sends it back to the
incoming address, i.e., the attacker.

5) Attacker receives the advertisement, takes note of the mapping,
replaces the IPv4 address and UDP port by the original values in the
intercepted message, and sends the response to the client.

6) Client receives the advertisement, notes that the authentication
header is present and is correct, and uses the proposed prefix and
the mapped addresses in the origin indication encapsulation.

The root cause of the problem is that the NAT is, in itself, a man-
in-the-middle attack.  The Authentication encapsulation covers the
encapsulated IPv6 packet, but does not cover the encapsulating IPv4
header and UDP header.  It is very hard to devise an effective
authentication scheme, since the attacker does not do anything else
than what the NAT legally does!

However, there are several mitigating factors that lead us to avoid
worrying too much about this attack.  In practice, the gain from the
attack is either to deny service to the client or to obtain a "man-
in-the-middle" position.  However, in order to mount the attack, the
attacker must be able to suppress traffic originating from the
client, i.e., have denial of service capability; the attacker must
also be able to observe the traffic exchanged between client and
inject its own traffic in the mix, i.e., have man-in-the-middle
capacity.  In summary, the attack is very hard to mount, and the gain
for the attacker in terms of "elevation of privilege" is minimal.

A similar attack is described in detail in the security section of
[RFC3489].

**7.2.2.  Attacker Spoofing a Teredo Relay**

An attacker may try to use Teredo either to pass itself for another
IPv6 host or to place itself as a man-in-the-middle between a Teredo
host and a native IPv6 host.  The attacker will mount such attacks by
spoofing a Teredo relay, i.e., by convincing the Teredo host that
packets bound to the native IPv6 host should be relayed to the IPv4
address of the attacker.

The possibility of the attack derives from the lack of any
algorithmic relation between the IPv4 address of a relay and the
native IPv6 addresses served by these relay.  A Teredo host cannot
decide just by looking at the encapsulating IPv4 and UDP header
whether or not a relay is legitimate.  If a Teredo host decided to
simply trust the incoming traffic, it would easily fall prey to a
relay-spoofing attack.

The attack is mitigated by the "direct IPv6 connectivity test"
specified in Section 5.2.9. The test specifies a relay discovery
procedure secured by a nonce.  The nonce is transmitted from the
Teredo host to the destination through Teredo server, which the
client normally trusts.  The response arrives through the "natural"
relay, i.e., the relay closest to the IPv6 destination.  Sending
traffic to this relay will place it out of reach of attackers that
are not on the direct path between the Teredo host and its IPv6 peer.

End-to-end security protections are required to defend against
spoofing attacks if the attacker is on the direct path between the
Teredo host and its peer.

**7.2.3.  End-to-End Security**

The most effective line of defense of a Teredo client is probably not
to try to secure the Teredo service itself: even if the mapping can
be securely obtained, the attacker would still be able to listen to
the traffic and send spoofed packets.  Rather, the Teredo client
should realize that, because it is located behind a NAT, it is in a

situation of vulnerability; it should systematically try to encrypt
its IPv6 traffic, using IPsec.  Even if the IPv4 and UDP headers are
vulnerable, the use of IPsec will effectively prevent spoofing and
listening of the IPv6 packets by third parties.  By providing each
client with a global IPv6 address, Teredo enables the use of IPsec

   without the configuration restrictions still present in "Negotiation
   of NAT-Traversal in the IKE" [RFC3947] and ultimately enhances the
   security of these clients.

### 7.3.  Denial of the Teredo service

   Our analysis outlines five ways to attack the Teredo service.  There
   are countermeasures for each of these attacks.

### 7.3.1.  Denial of Service by a Rogue Relay

   An attack can be mounted on the IPv6 side of the service by setting
   up a rogue relay and letting that relay advertise a route to the
   Teredo IPv6 prefix.  This is an attack against IPv6 routing, which
   can also be mitigated by the same kind of procedures used to
   eliminate spurious route advertisements.  Dual-stack nodes that
   implement "host local" Teredo relays are impervious to this attack.

### 7.3.2.  Denial of Service by Server Spoofing

   In Section 7.2, we discussed the use of spoofed router advertisements
   to insert an attacker in the middle of a Teredo conversation.  The
   spoofed router advertisements can also be used to provision a client
   with an incorrect address, pointing to either a non-existing IPv4
   address or the IPv4 address of a third party.

   The Teredo client will detect the attack when it fails to receive
   traffic through the newly acquired IPv6 address.  The attack can be
   mitigated by using the authentication encapsulation.

### 7.3.3.  Denial of Service by Exceeding the Number of Peers

   A Teredo client manages a cache of recently used peers, which makes
   it stateful.  It is possible to mount an attack against the client by
   provoking it to respond to packets that appear to come from a large
   number of Teredo peers, thus trashing the cache and effectively
   denying the use of direct communication between peers.  The effect
   will last only as long as the attack is sustained.

### 7.3.4.  Attacks against the Local Discovery Procedure

   There is a possible denial of service attack against the local peer
   discovery procedure, if attackers can manage to send spoofed local
   discovery bubbles to a Teredo client.  The checks described in
   Section 5.2.8 mitigate this attack.  Clients who are more interested
   in security than in performance could decide to disable the local
   discovery procedure; however, if local discovery is disabled, traffic
   between local nodes will end up being relayed through a server

external to the local network, which has questionable security
implications.

### 7.3.5.  Attacking the Teredo Servers and Relays

It is possible to mount a brute force denial of service attack
against the Teredo servers by sending them a very large number of
packets.  This attack will have to be brute force, since the servers
are stateless, and can be designed to process all the packets that
are sent on their access line.

The brute force attack against the Teredo servers is mitigated if
clients are ready to "failover" to another server.  Bringing down the
servers will, however, force the clients that change servers to
renumber their Teredo address.

It is also possible to mount a brute force attack against a Teredo
relay.  This attack is mitigated if the relay under attack stops
announcing the reachability of the Teredo service prefix to the IPv6
network: the traffic will be picked up by the next relay.

An attack similar to that described in Section 7.3.2 can be mounted
against a relay.  An IPv6 host can send IPv6 packets to a large
number of Teredo destinations, forcing the relay to establish state
for each of these destinations.  Teredo relays can obtain some
protection by limiting the range of IPv6 clients that they serve, and
by limiting the amount of state used for "new" peers.

### 7.4.  Denial of Service against Non-Teredo Nodes

There is a widely expressed concern that transition mechanisms such
as Teredo can be used to mount denial of service attacks, by
injecting traffic at locations where it is not expected.  These
attacks fall in three categories: using the Teredo servers as a
reflector in a denial of service attack, using the Teredo server to
carry a denial of service attack against IPv6 nodes, and using the
Teredo relays to carry a denial of service attack against IPv4 nodes.
The analysis of these attacks follows.  A common mitigating factor in
all cases is the "regularity" of the Teredo traffic, which contains
highly specific patterns such as the Teredo UDP port, or the Teredo
IPv6 prefix.  In case of attacks, these patterns can be used to
quickly install filters and remove the offending traffic.

We should also note that none of the listed possibilities offer any
noticeable amplification.

### 7.4.1.  Laundering DoS attacks from IPv4 to IPv4

   An attacker can use the Teredo servers as reflectors in a denial of
   service attack aimed at an IPv4 target.  The attacker can do this in
   one of two ways.  The first way is to construct a Router Solicitation

   message and post it to a Teredo server, using as IPv4 source address
   the spoofed address of the target; the Teredo server will then send a
   Router advertisement message to the target.  The second way is to
   construct a Teredo IPv6 address using the Teredo prefix, the address
   of a selected server, the IPv4 of the target, and an arbitrary UDP
   port, and to then send packets bound to that address to the selected
   Teredo server.

   Reflector attacks are discussed in [REFLECT], which outlines various
   mitigating techniques against such attacks.  One of these mitigations
   is to observe that "the traffic generated by the reflectors [has]
   sufficient regularity and semantics that it can be filtered out near
   the victim without the filtering itself constituting a denial-of-
   service to the victim ('collateral damage')".  The traffic reflected
   by the Teredo servers meets this condition: it is clearly
   recognizable, since it originates from the Teredo UDP port; it can be
   filtered out safely if the target itself is not a Teredo user.  In
   addition, the packets relayed by servers will carry an Origin
   indication encapsulation, which will help determine the source of the
   attack.

### 7.4.2.  DoS Attacks from IPv4 to IPv6

   An attacker may use the Teredo servers to launch a denial of service
   attack against an arbitrary IPv6 destination.  The attacker will
   build an IPv6 packet bound for the target and will send that packet
   to the IPv4 address and UDP port of a Teredo server, to be relayed
   from there to the target over IPv6.

   The address checks specified in Section 5.3.1 provide some protection
   against this attack, as they ensure that the IPv6 source address will
   be consistent with the IPv4 source address and UDP source port used
   by the attacker: if the attacker cannot spoof the IPv4 source
   address, the target will be able to determine the origin of the
   attack.

   The address checks ensure that the IPv6 source address of packets
   forwarded by servers will start with the IPv6 Teredo prefix.  This is
   a mitigating factor, as sites under attack could use this to filter
   out all packets sourced from that prefix during an attack.  This will
   result in a partial loss of service, as the target will not be able
   to communicate with legitimate Teredo hosts that use the same prefix.

However, the communication with other IPv6 hosts will remain
unaffected, and the communication with Teredo hosts will be able to
resume when the attack has ceased.

### 7.4.3.  DoS Attacks from IPv6 to IPv4

An attacker with IPv6 connectivity may use the Teredo relays to
launch a denial of service attack against an arbitrary IPv4
destination.  The attacker will compose a Teredo IPv6 address using
the Teredo prefix, a "cone" flag set to 1, the IPv4 address of the
target, and an arbitrary UDP port.

In the simplest variation of this attack, the attacker sends IPv6
packets to the Teredo destination using regular IPv6 routing.  The
packets are picked by the nearest relay, which will forward them to
the IPv4 address of the target.  In a more elaborate variant, the
attacker tricks a Teredo into sending packets to the target, either
by sending a first packet with a spoofed IPv6 address and letting the
Teredo host reply or by publishing a spoofed IPv6 address in a name
service.

There are three types of IPv4 addresses that an attacker may embed in
the spoofed Teredo address.  It may embed a multicast or broadcast
address, an local unicast address, or a global unicast address.

With multicast or broadcast addresses, the attacker can use the
multiplying effect of multicast routing.  By sending a single packet,
it can affect a large number of hosts, in a way reminiscent of the
"smurf" attack.

By using local addresses, the attacker can reach hosts that are not
normally reachable from the Internet, for example, hosts connected to
the a Teredo relay by a private subnet.  This creates an exposure
for, at a minimum, a denial of service attack against these otherwise
protected hosts.  This is similar to attack variants using source
routing to breach a perimeter.

The address checks specified in Section 5.2.4, 5.3.1, and 5.4.1
verify that packets are relayed only to a global IPv4 address.  They
are designed to eliminate the possibility of using broadcast,
multicast or local addresses in denial of service or other attacks.
In what follows, we will only consider attacks targeting globally
reachable unicast addresses.

The attacks can be targeted at arbitrary UDP ports, such as, for
example, the DNS port of a server.  The UDP payload must be a well-
formed IPv6 packet, and is thus unlikely to be accepted by any well-
written UDP service; in most case, the only effect of the attack will
be to overload the target with random traffic.

A special case occurs if the attack is directed to an echo service.
The service will echo the packets.  Since the echo service sees the
request coming from the IPv4 address of the relay, the echo replies
will be sent back to the same relay.  According to the rules
specified in Section 5.4, these packets will be discarded by the
Teredo relay.  This is not a very efficient attack against the Teredo
relays -- establishing a legitimate session with an actual Teredo
host would create more traffic.

The IPv6 packets sent to the target contain the IPv6 address used by
the attacker.  If ingress filtering is used in the IPv6 network, this

address will be hard to spoof.  If ingress filtering is not used, the
attacker can be traced if the IPv6 routers use a mechanism similar to
ICMP Traceback.  The ICMP messages will normally be collected by the
same relays that forward the traffic from the attacker; the relays
can use these messages to identify the source of an ongoing attack.
The details of this solution will have to be developed in further
research.

## 8.  IAB Considerations

The IAB has studied the problem of "Unilateral Self Address Fixing"
(UNSAF), which is the general process by which a client attempts to
determine its address in another realm on the other side of a NAT
through a collaborative protocol reflection mechanism [RFC3424].
Teredo is an example of a protocol that performs this type of
function.  The IAB has mandated that any protocols developed for this
purpose document a specific set of considerations.  This section
meets those requirements.

## 8.1.  Problem Definition

From [RFC3424], any UNSAF proposal must provide a precise definition
of a specific, limited-scope problem that is to be solved with the
UNSAF proposal.  A short-term fix should not be generalized to solve
other problems; this is why "short term fixes usually aren't".

The specific problem being solved by Teredo is the provision of IPv6
connectivity for hosts that cannot obtain IPv6 connectivity natively
and cannot make use of 6to4 because of the presence of a NAT between
them and the 6to4 relays.

8.2.  Exit Strategy

   From [RFC3424], any UNSAF proposal must provide the description of an
   exit strategy/transition plan.  The better short term fixes are the
   ones that will naturally see less and less use as the appropriate
   technology is deployed.

   Teredo comes with its own built-in exit strategy: as soon as a client
   obtains IPv6 connectivity by other means, either 6to4 or native IPv6,
   it can cease using the Teredo service.  In particular, we expect that
   the next generation of home routers will provide an IPv6 service in
   complement to the current IPv4 NAT service, e.g., by relaying
   connectivity obtained from the ISP, or by using a configured or
   automatic tunnel service.

   As long as Teredo is used, there will be a need to support Teredo
   relays so that the remaining Teredo hosts can communicate with native
   IPv6 hosts.  As Teredo usage declines, the traffic load on the relays
   will decline.  Over time, managers will observe a reduced traffic
   load on their relays and will turn them off, effectively increasing
   the pressure on the remaining Teredo hosts to upgrade to another form
   of connectivity.

   The exit strategy is facilitated by the nature of Teredo, which
   provides an IP-level solution.  IPv6-aware applications do not have
   to be updated to use or not use Teredo.  The absence of impact on the
   applications makes it easier to migrate out of Teredo: network
   connectivity suffices.

   There would appear to be reasons why a Teredo implementation might
   decide to continue usage of the Teredo service even if it already has
   obtained connectivity by some other means, for example:

   1. When a client is dual homed, and it wishes to improve the service
   when communicating with other Teredo hosts that are "nearby" on the
   IPv4 network.  If the client only used its native IPv6 service, the
   Teredo hosts would be reached only through the relay.  By maintaining
   Teredo, the Teredo hosts can be reached by direct transmission over
   IPv4.

   2. If, for some reason, the Teredo link is providing the client with
   better service than the native IPv6 link, in terms of bandwidth,
   packet loss, etc.

   The design of Teredo mitigates the dual-homing reason.  A host that
   wishes to communicate with Teredo peers can implement a "host-based
   relay", which is effectively an unnumbered Teredo interface.  As
   such, the dual-homed host will obtain Teredo connectivity with those

hosts that must use Teredo, but will not inadvertently encourage
other dual-homed hosts to keep using the Teredo service.

The bubbles and the UDP encapsulation used by Teredo introduce a
significant overhead.  It would take exceptional circumstances for
native technologies to provide a lesser service than Teredo.  These
exceptional circumstances, or other unforeseen reasons, may induce
hosts to keep using the Teredo service despite the availability of
native IPv6 connectivity.  However, these circumstances are likely to
be rare and transient.  Moreover, if the primary reason to use Teredo
fades away, one can expect that Teredo relays will be progressively
turned off and that the quality of the Teredo service will
progressively degrade, reducing the motivation to use the Teredo
service.

## 8.3.  Brittleness Introduced by Teredo

From [RFC3424], any UNSAF proposal must provide a discussion of
specific issues that may render systems more "brittle".  For example,
approaches that involve using data at multiple network layers create
more dependencies, increase debugging challenges, and make it harder
to transition.

Teredo introduces brittleness into the system in several ways: the
discovery process assumes a certain classification of devices based
on their treatment of UDP; the mappings need to be continuously
refreshed; and addressing structure may cause some hosts located
behind a common NAT to be unreachable from each other.

There are many similarities between these points and those introduced
by Simple Traversal of the UDP Protocol through NAT (STUN) [RFC3489];
however, Teredo is probably somewhat less brittle than STUN.  The
reason is that all Teredo packets are sent from the local IPv4 Teredo
service port, including discovery, bubbles, and actual encapsulated
packets.  This is different from STUN, where NAT type detection and
binding allocation use different local ports (ephemeral, in both
cases).

Teredo assumes a certain classification of devices based on their
treatment of UDP (e.g., cone, protected cone and symmetric).  There
could be devices that would not fit into one of these molds, and
hence would be improperly classified by Teredo.

The bindings allocated from the NAT need to be continuously
refreshed.  Since the timeouts for these bindings are very
implementation specific, the refresh interval cannot easily be

determined.  When the binding is not being actively used to receive
traffic, but to wait for an incoming message, the binding refresh
will needlessly consume network bandwidth.

The use of the Teredo server as an additional network element
introduces another point of potential security attack.  These attacks
are largely prevented by the security measures provided by Teredo,
but not entirely.

The use of the Teredo server as an additional network element
introduces another point of failure.  If the client cannot locate a
Teredo server, or if the server should be unavailable due to failure,
the Teredo client will not be able to obtain IPv6 connectivity.

The communication with non-Teredo hosts relies on the availability of
Teredo relays.  The Teredo design assumes that there are multiple
Teredo relays; the Teredo service will discover the relay closest to
the non-Teredo peer.  If that relay becomes unavailable, or is
misbehaving, communication between the Teredo hosts and the peers
close to that relay will fail.  This reliability issue is somewhat
mitigated by the possibility to deploy many relays, arbitrarily close
from the native IPv6 hosts that require connectivity with Teredo
peers.

Teredo imposes some restrictions on the network topologies for proper
operation.  In particular, if the same NAT is on the path between two
clients and the Teredo server, these clients will only be able to
interoperate if they are connected to the same link, or if the common
NAT is capable of "hairpinning", i.e., "looping" packets sent by one
client to another.

There are also additional points of brittleness that are worth
mentioning:

- Teredo service will not work through NATs of the symmetric variety.

- Teredo service depends on the Teredo server running on a network
  that is a common ancestor to all Teredo clients; typically, this is
  the public Internet.  If the Teredo server is itself behind a NAT,
  Teredo service will not work to certain peers.

- Teredo introduces jitter into the IPv6 service it provides, due to
  the queuing of packets while bubble exchanges take place.  This
  jitter can negatively impact applications, particularly latency
  sensitive ones, such as Voice over IP (VoIP).

8.4.  Requirements for a Long-Term Solution

   From [RFC3424], any UNSAF proposal must identify requirements for
   longer-term, sound technical solutions -- contribute to the process
   of finding the right longer-term solution.

   Our experience with Teredo has led to the following requirements for
   a long-term solution to the NAT problem: the devices that implement
   the IPv4 NAT services should in the future also become IPv6 routers.

9.  IANA Considerations

   This memo documents a request to IANA to allocate a 32-bit Teredo
   IPv6 service prefix, as specified in Section 2.6, and a Teredo IPv4
   multicast address, as specified in Section 2.17.

10.  Acknowledgements

   Many of the ideas in this memo are the result of discussions between
   the author and Microsoft colleagues, notably Brian Zill, John Miller,
   Mohit Talwar, Joseph Davies, and Rick Rashid.  Several encapsulation
   details are inspired from earlier work by Keith Moore.  The example
   in Section 5.1 and a number of security precautions were suggested by
   Pekka Savola.  The local discovery procedure was suggested by Richard
   Draves and Dave Thaler.  The document was reviewed by members of the
   NGTRANS and V6OPS working groups, including Brian Carpenter, Cyndi
   Jung, Keith Moore, Thomas Narten, Anssi Porttikivi, Pekka Savola, Eng
   Soo Guan, and Eiffel Wu.  Eric Klein, Karen Nielsen, Francis Dupont,
   Markku Ala-Vannesluoma, Henrik Levkowetz, and Jonathan Rosenberg
   provided detailed reviews during the IETF last call.

11.  References

11.1.  Normative References

   [RFC768]    Postel, J., "User Datagram Protocol", STD 6, RFC 768,
               August 1980.

   [RFC791]    Postel, J., "Internet Protocol", STD 5, RFC 791, September
               1981.

   [RFC1918]   Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G.,
               and E. Lear, "Address Allocation for Private Internets",
               BCP 5, RFC 1918, February 1996.

   [RFC2104]   Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-
               Hashing for Message Authentication", RFC 2104, February
               1997.

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

   [RFC2460]   Deering, S. and R. Hinden, "Internet Protocol, Version 6
               (IPv6) Specification", RFC 2460, December 1998.

   [RFC2461]   Narten, T., Nordmark, E., and W. Simpson, "Neighbor
               Discovery for IP Version 6 (IPv6)", RFC 2461, December
               1998.

   [RFC2462]   Thomson, S. and T. Narten, "IPv6 Stateless Address
               Autoconfiguration", RFC 2462, December 1998.

   [RFC3056]   Carpenter, B. and K. Moore, "Connection of IPv6 Domains
               via IPv4 Clouds", RFC 3056, February 2001.

   [RFC3424]   Daigle, L. and IAB, "IAB Considerations for UNilateral
               Self-Address Fixing (UNSAF) Across Network Address
               Translation", RFC 3424, November 2002.

   [RFC3566]   Frankel, S. and H. Herbert, "The AES-XCBC-MAC-96 Algorithm
               and Its Use With IPsec", RFC 3566, September 2003.

   [FIPS-180]  "Secure Hash Standard", Computer Systems Laboratory,
               National Institute of Standards and Technology, U.S.
               Department Of Commerce, May 1993.

Huitema                       Standards Track                  [Page 51]

## 11.2.  Informative References

   [RFC2993]   Hain, T., "Architectural Implications of NAT", RFC 2993,
               November 2000.

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

   [RFC3489]   Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy.
               "STUN - Simple Traversal of User Datagram Protocol (UDP)
               Through Network Address Translators (NATs)", RFC 3489,
               March 2003.

   [RFC3904]   Huitema, C., Austein, R., Satapati, S., and R. van der
               Pol, "Evaluation of IPv6 Transition Mechanisms for
               Unmanaged Networks", RFC 3904, September 2004.

   [RFC3947]   Kivinen, T., Swander, B., Huttunen, A., and V. Volpe,
               "Negotiation of NAT-Traversal in the IKE", RFC 3947,
               January 2005.

   [RFC4302]   Kent, S., "IP Authentication Header", RFC 4302, December
               2005.

   [RFC4303]   Kent, S., "IP Encapsulating Security Payload (ESP)", RFC
               4303, December 2005.

   [RFC4306]   Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC
               4306, December 2005.

   [REFLECT]   V. Paxson, "An analysis of using reflectors for
               distributed denial of service attacks", Computer
               Communication Review, ACM SIGCOMM, Volume 31, Number 3,
               July 2001, pp 38-47.

Author's Address

   Christian Huitema
   Microsoft Corporation
   One Microsoft Way
   Redmond, WA 98052-6399

   EMail: huitema@microsoft.com

Full Copyright Statement

Intellectual Property

Acknowledgement