第VII部

IP Version6

第 7 部
IP Version6

この章では、IPv6、IPsec、Mobile IPv6 に関する研究に取り組んでいる IPv6 分科会が 2000 年度に活動した内容について報告する。

まず、KAME プロジェクトの成果として、各 BSD へのマージの状況について述べる。次に、2000 年度に立ち上がった USAGI プロジェクトの概要を説明する。

さらに、IPv6 分科会で取り組んだ 2 つの実験に触れた後、3 つのイベントに関して報告する。

最後に下記に示す Intenet-Draft を添付する。

- IPv6 multihoming support at site exit routers
- Possible abuse against IPv6 transition technologies
- An analysis of IPv6 anycast
- Requirements for IPv6 dialup PPP operation
- Guidelines for IPv6 local experiments
- Socket API for IPv6 flow label field
- An IPv6/IPv4 multicast translator based on IGMP/MLD Proxying (IMP)
- A RADIUS attribute for IPv6 dialup PPP with static address assignment
- An Extension of Format for IPv6 Scoped Addresses
- IPv6 SMTP operational requirements
- Socket API for IPv6 traffic class field
- An IPv6-to-IPv4 transport relay translator

第 1 章　KAME のマージ状況

KAME の実装は、4 つの BSD すべてに取り込まれ、以下のように正式にリリースされた。

現在では、2 月に 1 回の stable リリースは意義が薄れたため廃止し、毎週 snap リリースを提供している。

BSD/OS 4.2	2000 年 11 月 29 日
FreeBSD 4.0 (4.2)	2000 年 3 月 14 日
NetBSD 1.5	2000 年 12 月 6 日
OpenBSD 2.8	2000 年 12 月 1 日

第 2 章　USAGI プロジェクト

オープンソースの OS として世界最大のユーザ数を抱えている Linux に、IPv6 の機能が提供されることは意義が大きい。

Linux にも IPv6 が実装されているが、他の OS と比べると、品質が優れているとは言えない。これは、TAHI プロジェクトの試験によっても明らかである。

そこで、WIDE プロジェクトと、それに参加している慶應義塾大学、東京大学、横河電機、さらに日本の Linux IPv6 ユーザ・グループである Linux IPv6 Users JP が中心となって、2000 年 10 月に USAGI (UniverSAl playGround for Ipv6) プロジェクトを開始した。

USAGI プロジェクトの目的は、BSD 系 UNIX を対象としている KAME プロジェクトに習い、高品質の IPv6 コードをオープンソースとして Linux に提供することである。また、Linux 環境で IPv6 を普及、発展させていくことも目指す。

現時点での参加企業ならびに参加大学は、

- IIJ
- NTT ソフトウェア
- 九州大学
- 慶應義塾大学
- 東京大学
- 東芝
- 日立製作所
- 三菱電機情報ネットワーク
- 横河電機
- 総務省 通信総合研究所

であり、実装メンバーは 10 名である。

2.1 成果物

最初の official バージョンを 2000 年 11 月にリリースし、その後 2 週間に 1 度の割合で snapshot をリリースしている。

この snapshot では、カーネル、glibc、および基本コマンド群に対するパッチと、Debian GNU/Linux 向けのバイナリ・パッケージを配布している。まとまった機能が実装された時点で提供される official リリースでは、それに加えて、Rad Hat Linux、Turbo Linux (日本語版)、ならびに Kondara MNU/Linux のバイナリ・パッケージを提供している。

2.2 連絡先

USAGI プロジェクトのホームページは、以下の通り。

http://www.linux-ipv6.org/

USAGI プロジェクトのメーリングリストには、以下の 3 つがある。

usagi-core@linux-ipv6.org

USAGI プロジェクトの開発メンバーのメーリングリスト。開発メンバーに用事のある場合は、個人宛ではなく、ここにメールすること。

usagi-announce@linux-ipv6.org

Snapshot のリリースなどをアナウンスするためのメーリングリスト。英語でアナウンスされる。

usagi-users@linux-ipv6.org

USAGI プロジェクトの成果物を利用しているユーザのためのメーリングリスト。usagi-announce へのアナウンスメールは、こちらにも転送される。公用語は英語。

第 3 章　各種実験

KAME プロジェクトでは、重要な課題のいくつかに対して実験を実施している。

3.1 リナンバー実験

IPv6 では、ホストにアドレス自動設定の機能が標準で装備されている。また、アドレス自動設定に関する情報は、特殊なサーバからではなく、ルータから受け取る。

これは、サイト全体のアドレス付け換えを、ルータのアドレス付け換えの問題に置き換えられることを意味している。さらに、ルータのアドレス付け換えを自動化するルータ・リナンバリング・プロトコルが提案されている。

ルータ・リナンバリング・プロトコルは、標準化され RFC[29] となっているが運用経験はほとんどない。そこで、以下のような目標を掲げて実験を遂行した。

- ルータ・リナンバリング・プロトコルがうまく動作するかの検証
- どのタイミングで何をすればよいかの明確化
- 経験のフィードバック (ガイドラインの作成など)

実験では、4 セグメントからなるサイトを構築した。ルータ・リナンバリング・プロトコルの実装は KAME オリジナル。マルチキャスト経路制御プロトコル PIM-DM (Dense Mode) は、オレゴン大学の実装に基づく KAME オリジナルの実装を使用した。

結論としては、少なくとも経路集約が少なく、出口にフィルターがないサイトでは、良好に動作することを確認した。ここで、良好に動作するとは、以下の項目を含む。

- ルータのアドレス付け換えは OK
- ホストのアドレス付け換えも OK
- SMTP/Web サーバも再起動なしに動く

今後の課題としては、以下のことが挙げられる。

- アドレスに依存している設定ファイルの切り出し
- DNS への登録の自動化
- DNS データベースエントリの寿命とリナンバ周期の関係

3.2 MIPv6 相互接続実験

MIPv6 (Mobile IPv6) の実装として、KAME ベースのものが 3 つ、Linux ベースのものが 1 つ出てきたので、2001 年 1 月 23 日と 2001 年 2 月 19 日に相互接続実験の機会を作った。

相互接続実験には、KAME ベースの Ericsson のコードを除く以下の実装が参加した。

NEC
移動ノード、固定ノード、ホーム・エージェント
SFC
移動ノード、固定ノード、ホーム・エージェント
ヘルシンキ大学 (USAGI)
移動ノード、固定ノード、ホーム・エージェント
KAME
固定ノード (共通部分)

MIP6 では、移動ノードとホーム・エージェント間では IPsec による認証が必須である。また、移動ノードと固定ノード間では、認証は必須ではないが、認証が利用できれば経路を最適に制御できる。

結果を以下にまとめる。

第 1 回相互接続実験

- 認証 (IPsec) を使わない場合は、おおむねうまくいく。ただし、移動ノードとホーム・エージェント間の通信は、規格にそっていないことになる。
- 認証 (IPsec) を使った場合は、これから。

第 2 回相互接続実験

- 認証 (IPsec) を使わない場合は、重大な問題はなし。
- 認証 (IPsec) を使った場合も接続に成功。ただし認証 (IPsec) の設定に難あり。

相互接続実験は、3 つある KAME ベースのスタックを今後どのようにしていくのかを考える機会でもあった。テストの結果を踏まえ、マージ等の検討を進めることが今後の課題である。

また、2001 年 3 月 1 日から 3 月 8 日にかけて Connectathon が開催され、NEC、SFC、TAHI が参加した。

第 4 章　イベント

最後に WIDE/KAME プロジェクトが中心となって開催した IPv6 のイベントを以下に示す。

4.1 N+I 2000 Tokyo

N+I 2000 Tokyo が、2000 年 6 月 7 日から 9 日にかけて幕張メッセで開催された。基調講演で、村井先生が IPv6 時代の到来を強調したこともあって、IPv6 が注目を集めた。

各出展社ブースには IPv6 の接続性が提供されていた。このような試みは、世界をまわる N+I の中でも、東京が初めて。また IPv6 のために 135M の対外線が大手町 (NSPIXP6) まで確保されていた。KAME の core メンバーの一部は、ShowNet NOC メンバーとして参加。ShowNet では GR 2000 が利用された。

公衆端末はすべて IPv6 に対応していた。その内 Windows 2000 を搭載した PC には、マイクロソフトから N+I 用に特別に提供して頂いた IPv6 対応の Internet Explorer を組み込んだ。

マイクロソフトがスポンサーとなり、「IPv6 ShowCase」という IPv6 専用のブースも用意された。このブースには、WIDE Project、TAHI Project、Kondara Project が協力。

参加企業/団体は、下記の通り。

- IIJ
- ウェーブテック・ワンデル・ゴルターマン
- NEC
- Cisco
- デジタルファクトリ
- 日本 HP
- 日立
- 富士通
- マイクロソフト
- 松下電送システム
- ヤマハ

展示としては、バックボーンルータを設置したルー

タラックが 2 本、SOHO ルータが 3 つ、BSD、Linux、Windows 2000 などのホストを用意。

松下電送システムは、IEEE1394 テレビのプロトタイプを展示しており、実際にビデオストリームを IPv6 を使って流した。また、KAME (FreeBSD) と Kondara Linux の間では、Quake で対戦。

また、IPv6 ShowCase では、1 時間毎に IPv6 の技術者による発表があり、各企業の製品の特徴や IPv6 に関する戦略を説明した。これらの発表は大変な好評を博し、隣のブースに迷惑な程観客を集めた。

2001 年も IPv6 ShowCase を開催することが決定している。

4.2 INET 2000

INET に先立つこと 1 週間、慶應義塾大学湘南藤沢キャンパスで、NTW (Network Training Workshop) が開催された。発展途上国の方を招いて、インターネットを構築する技術を集中的に講義した。

WIDE/KAME Project は、6 つのコースの内 2 つ対して、IPv6 を教える時間を頂いた。萩野と山本が講師となり、7 月 11 日と 13 日にそれぞれのコースで教えた。資料は、下記 URL を参照。

http://playground.iijlab.net/material/kazu-ntw-presen/

INET 2000 は、2000 年 7 月 19 日から 21 日にかけて横浜パシフィコで開催された。無論、すべてのネットワークが IPv6 に対応していた。特筆すべきは、無線ネットワーク (IEEE 802.11)。会議場や展示会場のみならず、インターコンチネンタルホテルでも利用可能だった。

N+I と同様に、展示会場では IPv6 ShowCase を設営した。今回は

- マイクロソフト
- 日立
- NEC

がスポンサーとなり、新たに

- SUN
- YDC
- NTT Communications
- 富士通研究所

が参加した。

N+I と違ったデモとしては、グループ通信が挙げられる。NTT Communications、は従来のマルチキャストを使って、パロアルトと会場を結び、IETF の ipngwg の 2 人の議長に参加して頂いた。また、富士通研究所は Xcast という新しいグループ通信の仕組みを使って、ビデオ中継やゲームをデモしていた。

4.3 Global IPv6 Summit in Japan

Global IPv6 Summit in Japan を 2000 年 12 月 18 日 から 19 日に開催した。これは、IPv6 Forum が主催する Global IPv6 Summit の名を借りて、IPv6 に携わる人たちがボランティアベースで運営したイベント。Internet Week 2000 の一部として開催。会場は、グランキューブ大阪 (大阪国際会議場)。

プログラムの概要を以下に示す。

- 村井先生の基調講演
- 日本の IPv6 に関するビジネス・レポート
- アジア諸国への IPv6 の普及状況
- パネル：「IPv6 がビジネスをどう変えるか」
- Steve Deering、Alain Durand 氏の基調講演
- 海外の IPv6 に関するビジネス・レポート
- パネル：「IPv6 への移行ストーリー」

最終的な、参加者数 652 人 (懇親会 267 人) であり、Internet Week 2000 の中では最大のイベントとなった。

詳細は、以下 URL を参照。

http://www.jp.ipv6forum.com/

第 5 章　IPv6 multihoming support at site exit routers

5.1 Problem

IPv6 specifications try to decrease the number of backbone routes, to cope with possible memory overflow problem in the backbone routers. To achieve this, the IPv6 addressing architecture [35] only allows the use of aggregatable addresses. Also, IPv6 network administration rules [39] do not allow non-aggregatable routing an-

nouncements to the backbone.

In IPv4, a multihomed site uses either of the following technique to achieve better reachability:

- Obtain a portable IPv4 address prefix, and announce it from multiple upstream providers.
- Obtain single IPv4 address prefix from ISP A, and announce it from multiple upstream providers the site is connected to.

The above two methodologies are not available in IPv6, but on the other hand IPv6 sites and hosts may obtain multiple simultaneous address prefixes to achieve the same result.

The document provides a way to configure site exit routers and ISP routers, so that the site can achieve better reachability from multihomed connectivity, without violating IPv6 rules. Since the technique uses already-defined routing protocol (BGP or RIPng) and tunnelling of IPv6 packets, the document introduces no new protocol standard.

The document is largely based on RFC2260 [13] by Tony Bates.

5.2 Goals and non-goals

The goal of this document is to achieve better packet delivery from a site to the outside, or from the outside to the site, even when some of the site exit links are down.

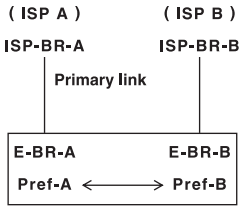
Non goals are:

- Choose the "best" exit link as possible. Note that there can be no common definition of "best" exit link.
- Achieve load-balancing between multiple exit links.

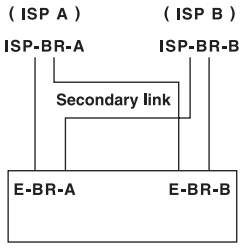
5.3 Basic mechanisms

We use technique described in RFC2260 section 5.2 onto our configuration. To summarize, for IPv4-only networks, RFC2260 says that:

- We assume that our site is connected to 2 ISPs, ISP-A and ISP-B.
- We are assigned IP address prefix, Pref-A and Pref-B, from ISP-A and ISP-B respectively. Hosts near ISP-A will get an address from Pref-A, and vice versa.
- In the site, we locally exchanage routes for Pref-A and Pref-B, so that hosts in the site can communicate with each other without using external link.
- ISP-A and our site is connected by "primary link" between ISP router ISP-BR-A and our router E-BR-A. ISP B and our site is connected by primary link between ISP router ISP-BR-B and our router E-BR-B.



- Establish a secondary link, between ISP-BR-A and E-BR-B, and ISP-BR-B and E-BR-A, respectively. Secondary link usually is IP-over-IP tunnel. It is important to have secondary link on top of different medium than primary link, so that one of them survives link failure. For example, secondary link between ISP-BR-A and E-BR-B should go through different medium than primary link between ISP-BR-A and E-BR- A. If secondary link is an IPv4-over-IPv4 tunnel, tunnel endpoint at E-BR-A needs to be an address in Pref-A, not in Pref-B (tunnelled packet needs to travel from ISP-BR-B to E-BR-A, over the primary link between ISP-BR-A and E-BR-A).



- For inbound packets, E-BR-A will advertise

(1) Pref-A toward ISP-BR-A with strong preference over primary link, and (2) Pref-B toward ISP- BR-B with weak preference over secondary link. Similarly, E-BR-B will advertise (1) Pref-B toward ISP-BR-B with strong preference over

primary link, and (2) Pref-A toward ISP-BR-A with weak preference over secondary link. Note that we always announce Pref-A to ISP-BR-A, and Pref-B to ISP- BR-B.

- For outbound packets, ISP-BR-A will advertise (1) default route (or specific routes) toward E-BR-A with strong preference over primary link, and (2) default route (or specific routes) toward E-BR-B with weak preference over secondary link. Similarly, ISP-BR-B will advertise (1) default route (or specific routes) toward E-BR-B with strong preference over primary link, and (2) default route (or specific routes) toward E-BR-A with weak preference over secondary link.

Under this configuration, both inbound and outbound packet can survive link failure on either side. Routing information with weak preference will be available as backup, for both inbound and outbound cases.

5.4 Extensions for IPv6

RFC2260 is written for IPv4 and BGP. With IPv6 and BGP4+, or IPv6 and RIPng, similar result can be achieved, without violating IPv6 addressing/routing rules.

IPv6 rule conformance

In RFC2260, we announce Pref-A toward ISP-BR-A only, and Pref-B toward ISP-BR-B only. Therefore, there will be no extra routing announcement to the outside of the site. This conforms to the aggregation requirement in IPv6 documents. Also, RFC2260 does not require portable addresses.

Address assignment to the nodes

In IPv4, it is usually assumed that a node will be assigned single IPv4 address. Therefore, RFC2260 assumed that addresses from Pref-A will be assigned to nodes near E-BR-A, and vice versa (second bullet in the previous section).

With IPv6, multiple IPv6 addresses can be assigned to a node. So we can assign (1) one address from Pref-A, (2) one address from Pref-B, or (3) two addresses from both address prefixes, to a single node in the site.

This will allow more flexibility in node configuration. However, this may make source address selection on a node more complex. Source address selection itself is out of scope of the document.

Configuration of links

With IPv6, primary link can be IPv6 native connectivity, RFC1933 [53] IPv6-over-IPv4 configured tunnel, 6to4 [23], IPv6-over-IPv4 encapsulation, or some others.

If tunnel-based connectivity is used in some of primary links, administrators may want to avoid IPv6-over-IPv6 tunnels for secondary links. For example, if:

- primary links to ISP-A and ISP-B are RFC1933 IPv6-over-IPv4 tunnels, and
- ISP-A, ISP-B and the site have IPv4 connectivity with each other,

it makes no sense to configure a secondary link by IPv6-over-IPv6 tunnel, since it will actually be IPv6-over-IPv6-over-IPv4 tunnel. In this case, IPv6-over-IPv4 tunnel should be used for secondary link. IPv6-over-IPv4 configuration has a big advantage against IPv6-over- IPv6-over-IPv4 configuration, as secondary link will be able to have the same path MTU than the primary link.

Using RFC2260 with IPv6 and BGP4+

RFC2260 approach on top of IPv6 will work fine as documented in RFC2260. There will be no ex-

tra twists necessary.

Using RFC2260 with IPv6 and RIPng

It is possible to run RFC2260-like configuration with RIPng [Malkin, 1997] , with careful control of metric. Routers in the figure needs to increase RIPng metric on secondary link, to make primary link a preferred path.

If we denote the RIPng metric for route announcement, from router R1 toward router R2, as metric (R1, R2), the invariants that must hold are:

- metric(E-BR-A, ISP-BR-A) < metric(E-BR-B, ISP-BR-A)
- metric(E-BR-B, ISP-BR-B) < metric(E-BR-A, ISP-BR-B)
- metric(ISP-BR-A, E-BR-A) < metric(ISP-BR-A, E-BR-B)
- metric(ISP-BR-B, E-BR-B) < metric(ISP-BR-B, E-BR-A)

Note that smaller metric means stronger route in RIPng.

5.5 Issues with ingress filters in ISP

If the upstream ISP imposes ingress filters [47] to outbound traffic, story becomes much more complex. A packet with source address taken from Pref-A must go out from ISP-BR-A. Similarly, a packet with source address taken from Pref-B must go out from ISP-BR-B. Since none of the routers in the site network will route packets based on source address, packets can easily be routed to incorrect border router.

One possible way is to negotiate with both ISPs, to allow both Pref-B and Pref-A to be used as source address. This approach does not work if upstream ISP of ISP-A imposes ingress filtering. Since there will be multiple levels of ISP on top of ISP-A, it will be hard to understand which upstream ISP imposes the filter. In reality, this problem will be very rare, as ingress filter is not suitable for use in large ISPs where smaller ISPs are connected beneath.

Another possibility is to use source-based routing at E-BR-A and E-BR-B. Here we assume that IPv6-over-IPv6 tunnel is used for secondary links. When an outbound packet arrives to E-BR-A with source address in Pref-B, E-BR-A will forward it to secondary link (tunnel to ISP-BR-B) based on source-based routing decision. The packet will look like this:

- Outer IPv6 header: source = address of E-BR-A in Pref-A, dest = ISP-BR-B
- Inner IPv6 header: source = address in Pref-B, dest = final dest

Tunneled packet will travel across ISP-BR-A toward ISP-BR-B. The packet can go through ingress filter at ISP-BR-A, since it has outer IPv6 source address in Pref-A. Packet will reach ISP-BR-B and decapsulated before ingress filter is applied. Decapsulated packet can go through ingress filter at ISP-BR-B, since it now has source address in Pref-B (from inner IPv6 header). Notice the following facts when configuring this:

- Not every router implements source-based routing.
- The interaction between normal routing and source-based routing at E-BR-A (and/or E-BR-B) varies by router implementations.
- At ISP-BR-B (and/or ISP-BR-A), the interaction between tunnel egress processing and filtering rules varies by router implementations and filter configurations.

5.6 Observations

The document discussed the cases where a site has two upstream ISPs. The document can easily be extended to the cases where there are 3 or more upstream ISPs.

If you have many upstream providers, you would not make all ISPs backup each other, as it requires O(N^2) tunnels for N ISPs. Rather, it is better to make N/2 pairs of ISPs, and let each pair of ISP

backup each other. It is important to pick pairs which are unlikely to be down simultaneously. In this way, number of tunnels will be O(N).

Suppose that the site is very large and it has ISP links in very distant locations, such as in US and in Japan. In such case, it is wiser to use this technique only among ISP links in US, and only among ISP links in Japan. If you use this technique between ISP link A in US and ISP link B in Japan, the secondary link make packets travel very long path, for example, from host in the site in US, to E-BR-B in Japan, to ISP-BR-B (again in Japan), and then to the final destination in US. This may not make sense for actual use, due to excessive delay.

Similarly, in a large site, addresses must be assigned to end nodes with great care, to minimize delays due to extra path packets may travel. It may be wiser to avoid assigning an address in a prefix assigned from Japanese ISP, to an end node in US.

If one of primary link is down for a long time, administrators may want to control source address selection on end hosts so that secondary link is less likely to be used. This can be achieved by marking unwanted prefix as deprecated. Suppose the primary link toward ISP-A has been down. You will issue router advertisement [Thomson, 1998; Narten, 1998] packets from routers, with preferred lifetime set to 0 in prefix information option for Pref-A. End hosts will consider addresses in Pref-A as deprecated, and will not use any of them as source address for future connections. If an end host in the site makes new connection to outside, the host will use an address in Pref-B as source address, and reply packet to the end host will travel primary link from ISP-BR-B toward E-BR-B.

Some of non-goals (such as "best" exit link selection) can be achieved by combining technique described in this document, with some other techniques. One example of the technique would be the source/destination address selection heuristics on the end nodes.

5.7 Security considerations

The configuration described in the document introduces no new security problem.

If primary links toward ISP-A and ISP-B have different security characteristics (like encrypted link and non-encrypted link), administrators needs to be careful setting up secondary links tunneled on them. Packets may travel unwanted path, if secondary links are configured without care.

第 6 章　Possible abuse against IPv6 transition technologies

6.1 Abuse of IPv4 compatible address

Problem

To implement automatic tunnelling described in RFC1933 [53], IPv4 compatible addresses (like ::123.4.5.6) are used. From IPv6 stack point of view, an IPv4 compatible address is considered to be a normal unicast address. If an IPv6 packet has IPv4 compatible addresses in the header, the packet will be encapsulated automatically into an IPv4 packet, with IPv4 address taken from lowermost 4 bytes of the IPv4 compatible addresses. Since there is no good way to check if embedded

IPv4 address is sane, improper IPv4 packet can be generated as a result. Malicious party can abuse it, by injecting IPv6 packets to an IPv4/v6 dual stack node with certain IPv6 source address, to cause transmission of unexpected IPv4 packets. Consider the following scenario:

- You have an IPv6 transport-capable DNS server, running on top of IPv4/v6 dual stack node. The node is on IPv4 subnet 10.1.1.0/24.
- Malicious party transmits an IPv6 UDP packet to port 53 (DNS), with source address ::10.1.1.255. It does not make difference if it is encapsulated into an IPv4 packet, or is trans-

mitted as a native IPv6 packet.

- IPv6 transport-capable DNS server will transmit an IPv6 packet as a reply, copying the original source address into the destination address. Note that the IPv6 DNS server will treat IPv6 compatible address as normal IPv6 unicast address.
- The reply packet will automatically be encapsulated into IPv4 packet, based on RFC1933 automatic tunnelling. As a result, IPv4 packet toward 10.1.1.255 will be transmitted. This is the subnet broadcast address for your IPv4 subnet, and will (improperly) reach every node on the IPv4 subnet.

Possible solutions

For the following sections, possible soluitions are presented in the order of preference (the author recommends to implement solutions that appear earlier). Note that some of the following are partial solution to the problem. Some of the solutions may overwrap, or be able to coexist, with other solutions. Solutions marked with (*) are already incorporated into [53] which is an updated version of RFC1933. Note that, however, solutions incorporated into [Gilligan, 2000] do not make a complete protection against malicious parties.

- Disable automatic tunnelling support.
- Reject IPv6 packets with IPv4 compatible address in IPv6 header fields. Note that we may need to check extension headers as well.
- Perform ingress filter against IPv6 packet and tunnelled IPv6 packet. Ingress filter should let the packets with IPv4 compatible source address through, only if the source address embeds an IPv4 address belongs to the organization. The approach is a partial solution for avoiding possible transmission of malicious packet, from the organization to the outside. (*)
- Whenever possible, check if the addresses on the packet meet the topology you have. For example, if the IPv4 address block for your site is 43.0.0.0/8, and you have a packet from IPv4-wise outside with encapsulated IPv6 source matches ::43.0.0.0/104, it is likely that someone is doing something nasty. This may not be possible to make

the filter complete, so consider it as a partial solution. (*)

- Require use of IPv4 IPsec, namely authentication header [Kent, 1998] , for encapsulated packet. Even with IPv4 IPsec, reject the packet if the IPv6 compatible address in the IPv6 header does not embed the IPv4 address in the IPv4 header. We cannot blindly trust the inner IPv6 packet based on the existence of IPv4 IPsec association, since the inner IPv6 packet may be originated by other nodes and forwarded by the authenticated peer. The solution may be impractical, since it only solves very small part of the problem with too many requirements.
- Reject inbound/outgoing IPv6 packets, if it has certain IPv4 compatible address in IPv6 header fields. Note that we may need to check extension headers as well. The author recommends to check any IPv4 compatible address that is mapped from/to IPv4 address not suitable as IPv4 peer. They include 0.0.0.0/8, 127.0.0.0/8, 224.0.0.0/4, 255.255.255.255/32, and subnet broadcast addresses. Since the check can never be perfect (we cannot check for subnet broadcast address in remote site, for example) the direction is not recommend. (*)

6.2 Abuse of 6to4 address

6to4 [23] is another proposal for IPv6-over-IPv4 packet encapsulation, and is very similar to RFC1933 automatic tunneling mentioned in the previous section. 6to4 address embeds IPv4 address in the middle (2nd byte to 5th byte). If an IPv6 packet has a 6to4 address as destination ad-

dress, it will be encapsulated into IPv4 packet with the embedded IPv4 address as IPv4 destination.

IPv6 packets with 6to4 address have the same problems as those with IPv4 compatible address. See the previous section for the details of the problems, and possible solutions.

The latest 6to4 draft [23] do incoporate some of the solutions presented in the previous section, however, they do not make a complete protection against malicious parties.

6.3 Abuse of IPv4 mapped address

Problems

IPv6 basic socket API [53] defines the use of IPv4 mapped address with AF_INET6 sockets. IPv4 mapped address is used to handle inbound IPv4 traffic toward AF_INET6 sockets, and outbound IPv4 traffic from AF_INET6 sockets. Inbound case has higher probability of abuse, while outbound case contributes to the abuse as well. Here we briefly describe the kernel behavior in inbound case. When we have an AF_INET6 socket bound to IPv6 unspecified address (::), IPv4 traffic, as well as IPv6 traffic, will be captured by the socket. The kernel will present the address of the IPv4 peer to the userland program by using IPv4 mapped address. For example, if an IPv4 traffic from 10.1.1.1 is captured by an AF_INET6 socket, the userland program will think that the peer is at ::ffff:10.1.1.1. The userland program can manipulate IPv4 mapped address just like it would do against normal IPv6 unicast address.

We have three problems with the specification. First, IPv4 mapped address support complicates IPv4 access control mechanisms. For example, if you would like to reject accesses from IPv4 clients to a certain transport layer service, it is not enough to reject accesses to AF_INET socket. You will need to check AF_INET6 socket for accesses from IPv4 clients (seen as accesses from IPv4 mapped address) as well.

Secondly, malicious party may be able to use IPv6 packets with IPv4 mapped address, to bypass access control. Consider the following scenario:

- Attacker throws unencapsulated IPv6 packets, with ::ffff:127.0.0.1 as source address.
- The access control code in the server thinks that this is from localhost, and grants accesses.

Lastly, malicious party can make servers generate unexpected IPv4 traffic. This can be accomplished by sending IPv6 packet with IPv4 mapped address as a source (similar to abuse of IPv4 compatible address), or by presenting IPv4 mapped address to servers (like FTP bounce attack [6] from IPv6 to IPv4). The problem is slightly different from the problems with IPv4 compatible addresses and 6to4 addresses, since it does not make use of tunnels. It makes use of certain behavior of userland applications.

The confusion came from the dual use of IPv4 mapped address, for node- internal representation for remote IPv4 destination/source, and for real IPv6 destination/source.

Possible solutions

- In IPv6 addressing architecutre document [Hinden, 1998] , disallow the use of IPv4 mapped addresses on the wire. The change will conflict with SIIT [116] , which is the only protocol which tries to use IPv4 mapped address on IPv6 native packet. The dual use of IPv4 mapped address (as a host-internal representation of IPv4 destinations, and as a real IPv6 address) is the prime source of the problem.
- Reject IPv6 packets, if it has IPv4 mapped address in IPv6 header fields. Note that we may need to check extension headers such as routing headers, as well. IPv4 mapped address is internal representation in a node, so doing this will raise no conflicts with existing protocols. We recommend to check the condition in IPv6 input

packet processing, and transport layer processing (TCP input and UDP input) to be sure.

- Reject DNS replies, or other host name database replies, which contain IPv4 mapped address. Again, IPv4 mapped address is internal represntation in a node and should never appear on external host name databases.
- Do not route inbound IPv4 traffic to AF_INET6 sockets. When an application would like to accept IPv4 traffic, it should explicitly open AF_INET sockets. You may want to run two applications instead, one for an AF_INET socket, and another for an AF_INET6 socket. Or you may want to make the functionality optional, off by default, and let the userland applications explicitly enable it. This greatly simplifies access control issues. This approach conflicts with what IPv6 basic API document says, however, it should raise no problem with properly-written IPv6 applications. It only affects server programs, ported by assuming the behavior of AF_INET6 listening socket against IPv4 traffic.
- When implementing TCP or UDP stack, explicitly record the wire packet format (IPv4 or IPv6) into connection table. It is unwise to guess the wire packet format, by existence of IPv6 mapped address in the address pair.
- We should separately fix problems like FTP bounce attack.
- Applications should always check if the connection to AF_INET6 socket is from an IPv4 node (IPv4 mapped address), or IPv6 node. It should then treat the connection as from IPv4 node (not from IPv6 node with special adderss), or reject the connection. This is, however, dangerous to assume that every application implementers are aware of the issue. The solution is not recommended (this is not a solution actually).

6.4 Attacks by combining different address formats

Malicious party can use different address formats simultaneously, in a single packet. For example, suppose you have implemented checks for abuse against IPv4 compatible address in automatic tunnel egress module. Bad guys may try to send a native IPv6 packet with 6to4 destination address with IPv4 compatible source address, to bypass security checks against IPv4 compatible address in tunnel decapsulation module. Your implementation will not be able to detect it, since the packet will not visit egress module for automatic tunnels.

Analyze code path with great care, and reject any packets that does not look sane.

6.5 Attacks using source address-based authentication

Problems

IPv6-to-IPv4 translators [116, 156, 59] usually relay, or rewrite, IPv6 packet into IPv4 packet. The IPv4 source address in the IPv4 packet will not represent the ultimate source node (IPv6 node). Usually the IPv4 source address represents translator box instead. If we use the IPv4 source address for authentication at the destination IPv4 node, all traffic relayed/translated by the translator box will mistakenly be considered as authentic.

The problem applies to IPv4-to-IPv6 translators as well. The problem is similar to proxied services, like HTTP proxy.

Possible solutions

- Do not use translators, for protocols that use IP source address as authentication credental (for example, rlogin [92]).
- translators must implement some sort of access control, to reject any IPv6 traffic from malicious IPv6 nodes.
- Do not use source address based authentica-

tion. IP source address should not be used as an authentication credental from the first place, since it is very easy for malicious parties to spoof IP source address.

6.6 Conclusions

IPv6 transition technologies have been proposed, however, some of them looks immune against abuse. The document presented possible ways of abuse, and possible solutions against them. The presented solutions should be reflected to the revision of specifications referenced.

For coming protocols, the author would like to propose a set of guilelines for IPv6 transition technologies:

- Tunnels must explicitly be configured. Manual configuration, or automatic configuration with proper authentication, should be okay.
- Do not embed IPv4 addresses into IPv6 addresses, for tunnels or other cases. It leaves room for abuse, since we cannot practically check if embedded IPv4 address is sane.
- Do not define an IPv6 address format that does not appear on the wire. It complicates access control issues.

The author hopes to see more deployment of native IPv6 networks, where tunnelling technologies become unnecessary.

6.7 Security considerations

The document talks about security issues in existing IPv6 related protocol specifications. Possible solutions are provided.

第 7 章　An analysis of IPv6 anycast

7.1 IPv6 anycast

"Anycast" is a communication model for IP, just like unicast and multicast are. RFC1546 [125] documents IPv4 anycast, and it defines the term "anycast."

Anycast can be understood best by comparing with unicast and multicast. IP unicast allows a source node to transmit IP datagrams to a single destination node. The destination node is identified by an unicast address. IP multicast allows a source node to transmit IP datagrams to a group of destination nodes. The destination nodes are identfied by a multicast group, and we use a multicast address to identify the multicast group.

IP anycast allows a source node to transmit IP datagrams to a single destination node, out of a group of destination nodes. IP datagram will reach the closest destination node in the set of destination nodes, based on routing measure of distance. The source node does not need to care about how to pick the closest destination node, as the routing system will figure it out (in other words, the source node has no control over the selection). The set of destionation nodes is identified by an anycast address.

Anycast was adopted by IPv6 specification suite. RFC2373 [35] defines the IPv6 anycast address, and its constraints in the usage. The following sections try to analyze RFC2373 rules, and understand limitations with them. At the end of the draft we compile a couple of suggestions to exisitng proposals, for extending the usage of the IPv6 anycast.

7.2 Limitations/properties in the current proposals

Identifying anycast destination

For anycast addresses, RFC2373 uses the same address format as unicast addresses. Therefore, without other specific configurations, a sender cannot usually identify if the node is sending a packet to anycast destination, or unicast destination. This is different from experimental IPv4 anycast [125] , where anycast address is distinguishable from unicast addresses.

Nondeterministic packet delivery

If multiple packets carry an anycast address in IPv6 destionation address header, these packets may not reach the same destination node, depending on stability of the routing table. The property leads to a couple of interesting symptoms.

If we can assume that the routing table is stable enough during a protocol exchange, multiple packets (with anycast address in destination) will reach the same destination node just fine. However, there is no guarantee.

If routing table is not stable enough, or you would like to take a more strict approach, a client can only send one packet with anycast address in the destination address field. For example, consider the following packet exchange. The following exchange can lead us to failure, as we are not sure if the 1st and 2nd anycast packet have reached the same destination.

```
query:  client unicast (Cu) -> server anycast (Sa)
reply:  server unicast (Su) -> client unicast (Cu)
query:  client unicast (Cu) -> server anycast (Sa)
          It may reach a different server!
reply:  server unicast (Su) -> client unicast (Cu)
```

Because of the non-determinism, if we take a strict approach, we can use no more than 1 packet with anycast destination address, in a set of protocol exchange. If we use more than 2 packets, 1st and 2nd packet may reach different server and may cause unexpected results. If the protocol is completely stateless, and we can consider the first roundtrip and second roundtrip separate, it is okay. For stateful protocols, it is suggested to use anycast for the first packet in the exchange, to discover unicast address of the (nearest) server. After we have discovered the unicast address of the server, we should use the server's unicast address for the protocol exchange.

Also because of non-determinism, if we are to assign an IPv6 anycast address to servers, those servers must provide uniform services. For example, if server A and server B provide different quality of service, and people wants to differentiate between A and B, we cannot use single IPv6 anycast address to identify both A and B.

Note that, the property is not a bad thing; the property lets us use anycast addresses for load balancing. Also, packets will automatically be delivered to the nearest node with anycast address assigned.

Here are situations where multiple packets with anycast destination address can lead us to problems:

- Fragmented IPv6 packets. Fragments may reach multiple different destinations, and will prevent reassembly.

Because the sending node cannot differentiate between anycast addresses and unicast addresses, it is hard for the sending node to control the use of fragmentation.

Anycast address assignment to hosts

RFC2373 suggests to assign anycast addresses to a node, only when the node is a router. This is to avoid injecting host routes for anycast address, into the IPv6 routing system. If no hosts have anycast address on them, it is easier for us to route an IP datagram to anycast destination. We just need to follow existing routing entries, and we will eventually hit a router that has the anycast address. If we follow RFC2373 restriction strictly, we could only place anycast addresses onto routers.

Anycast address in source address

Under RFC2373, anycast address IPv6 anycast address can not be put into IPv6 source address. This is basically because an IPv6 anycast address does not identify single source node.

IPsec

IPsec and IKE identify nodes by using source/destination address pairs. Due to the combination of issues presented above, it is very hard to use IPsec on packets with anycast address in source address, destination address, or both.

Even with manual keying, IPsec trust model

with anycast address is confusing. As IPsec uses IPv6 destination address to identify which IPsec key to be used, we need to use the same IPsec key for all of the anycast destinations that share an anycast address.

Dynamic IPsec key exchange (like IKE) is almost impossible. First of all, to run IKE session between two nodes, the two nodes need to be able to communicate with each other. With nondeterministic packet delivery provided by anycast, it is not quite easy. Even if we could circumvent the issue with IKE, we have exactly the same problem as manual keying case for actual communication.

7.3 Possible improvements and protocol changes

Assigning anycast address to hosts (non-router nodes)

Under RFC2373 rule, we can only assign anycast addresses to routers, not to hosts. The restriction was put into the RFC because it was felt insecure to permit hosts to inject host routes to anycast address.

If we try to ease the restriction and assign anycast addresses to IPv6 hosts (non-routers), we would need to inject host routes for the anycast addresses, with prefix length set to /128, into the IPv6 routing system. We will inject host routes from each of the nodes with anycast addresses, to make packets routed to a topologically-closest node. Or, we may be able to inject host routes from routers adjacent to the servers (not from the servers themselvers).

Here are possible ways to allow anycast addresses to be assigned to hosts. We would need to diagnose each of the following proposals carefully, as they have different pros and cons. The most serious issue would be the security issue with service blackhole attack (malicious party can blackhole packets toward anycast addresses, by making false advertisement).

- Let the host with anycast address to participate into routing information exchange. The host does not need to fully participate; it only needs to announce the anycast address to the routing system. To

secure routing exchange, administrators need to configure secret information that protects the routing exchange to the host, as well as other routers.

- Develop a protocol for a router, to discover hosts with anycast address on the same link. The router will then advertise the anycast address to the routing system. This could be done by an extension to IPv6 Neighbor Discovery or an extension to IPv6 Multicast Listener Discovery [56].

The impact of host routes depends on the scope of the anycast address usage. For example, if we use site-local anycast address to identify a set of servers, the propagation of host route is limited inside the site. If the site administration policy permits it, and the site routers can handle the additional routing entries, the additional host routes are okay. So, we can safely assign anycast address to non-router nodes (hosts), and inject host route for the anycast address, into the site IPv6 routing system. It is still questionable to inject host routes into worldwide IPv6 routing system, as it has problems in terms of scalability. Also, because of IPv6 route aggregation rules [Rockell, 2000] it is normally impossible to propagate IPv6 host routes worldwide.

Anycast address in destination address

With anycast, it is hard to identify a single node out of nodes that share an anycast address. Suppose a client C would like to communicate a specific server with anycast address, Si. Si shares the same anyast address with other servers, S1 to Sn. It is hard for C to selectively communicate with Si.

One possible workaround is to use IPv6 routing

header. By specifying an unicast address of Si as an intermediate hop, C can deliver the packet to Si, not to other Sn.

Note that we now have lost the robustness provided by the use of anycast address. If Si goes down, the communication between C and Si will be lost. C cannot enjoy the failure resistance provided by redundant servers, S1 to Sn. Designers should carefully diagnose if any state is managed by C and/or Si, and decide if it is a good idea to use the workaround presented here.

Anycast address in source address

Under RFC2373 rule, anycast address cannot be put into source address. Here is a possible workaround, however, it could not win a consensus in the past ipngwg meetings:

- When we try to use anycast address in the source address, use an (non- anycast) unicast address as the IPv6 source address, and attach home address option with anycast address. In ipngwg discussions, however, there seem to be a consensus that the home address option should have the same semantics as the source address in the IPv6 header, so we cannot put anycast address into the home address option.

7.4 Upper layer protocol issus

Use of UDP with anycast

Many of the UDP-based protocols use source and destination address pair to identify the traffic. One example would be DNS over UDP; most of the DNS client implementation checks if the source address of the reply is the same as the destination address of the query, in the fear of the fabricated reply from bad guy.

```
query:  client unicast (Cu) -> server unicast (Su*)
reply:  server unicast (Su*) -> client unicast (Cu)
addresses marked with (*) must be equal.
```

If we use server's anycast address as the destination of the query, we cannot meet the requirement due to RFC2373 restriction (anycast address cannot be used as the source address of packets). Effectively, client will consider the reply is fabricated (hijack attempt), and drops the packet.

```
query:  client unicast (Cu) -> server anycast (Sa)
reply:  server unicast (Su) -> client unicast (Cu)
```

Note that, however, bad guys can still inject fabricated results to the client, even if the client checks the source address of the reply. The check does not improve security of the exchange at all. So, regarding to this issue, we conclude as follows:

- To use anycast address on the UDP protocol exchange, client side should not check the source address of the incoming packet. Packet pairs must be identified by using UDP port numbers or upper-layer protocol mechanisms (like cookies). The source address check itself has no real protection.
- If you need to secure UDP protocol exchange, it is suggested to verify the authenticity of the reply, by using upper-layer security mechanisms like DNSSEC (note that we cannot use IPsec with anycast).

Use of TCP with anycast

We cannot simply use anycast for TCP exchanges, as we identify a TCP connection by using address/port pair for the source/destination node. It is desired to implement some of the following, to enable the use of IPv6 anycast in TCP. Note, however, security requirement is rather complicated for the following protocol modifications.

- Define a TCP option which lets us to switch peer's address from IPv6 anycast address, to IPv6 unicast address. There are couple of proposals in the past.
- Define an additional connection setup protocol that resolves IPv6 unicast address from IPv6 anycast address. We first resolve IPv6 unicast address using the new protocol, and

then, make a TCP connection using the IPv6 unicast address. IPv6 node information query/reply [30] could be used for this.

7.5 Summary

The draft tried to diagnose the limitation in currntly-specified IPv6 anycast, and explored couple of ways to extend its use. Some of the proposed changes affects IPv6 anycast in general, some are useful in certain use of IPv6 anycast. To take advantage of anycast addresses, protocol designers would need to diagnose their requirements to anycast address, and introduce some of the tricks described in the draft.

Use of IPsec with anycast address still needs a great amount of analysis.

7.6 Security consideration

The document should introduce no new security issues.

For secure anycast operation, we may need to enable security mechanisms in other protocols. For example, if we were to inject /128 routes from end hosts by using a routing protocol, we may need to configure the routing protocol to exchange routes securely, to prevent malicious parties from injecting bogus routes. With anycast, it is very important to prevent malicious parties from injecting bogus routes, as it allows them to effectively suck all traffic torward anycast address.

第 8 章　Requirements for IPv6 dialup PPP operation

8.1 Problem domain

With the deployment of IPv6 [65] , it becomes more apparent that we have different operational requirements in IPv6 dialup PPP operation, from IPv4 dialup PPP operation. For example, it would be desirable to see static address allocation, rather than dynamic address allocation, whenever possible. With IPv4 this has been impossible due to address space shortage, and IPCP [107] dynamic address allocation has been used. With IPv6 it is possible to perform static address allocation from ISPs to downstream customers, as there's enough address to spare.

The document tries to summarize possible design choices in IPv6 dialup PPP operation. Actual operational practices should be documented separately.

8.2 Design choices

The usage pattern

- Static clients. Computers located in home and offices do not usually change its configurations, nor upstream ISPs. It would be desirable to make a static address allocation in this case.
- Roaming clients. Roaming clients, like travelling users with notebook PC, have different requirement from static clients. It is not usually possible to make a static address allocation, as travelling users may connet to multiple ISPs from different countries.

Address space

It is desired to assign /48 address space, regardless from usage pattern or size of the downstream site. It is to make future renumbering in downstream site easier on ISP change. /128 assignment MUST NOT be made, as it will advocate IPv6-to-IPv6 NAT.

Address allocation

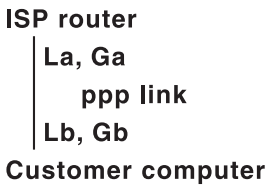
- Static address allocation. ISPs will allocate a static address space (/48) to a downstream customer, on contract time. There will be no protocol involved in address allocation - allocation will be informed by paper.
- Static address allocation, with some automation. It may be possible to define a com-

mon protocol for configuring customer-side router(s) from the upstream ISP, eliminating necessity of paper-based allocation and configuration labor in the customer site. Note that router renumbering protocol is not always suitable for this. Router renumbering protocol assumes that the routers and control node to be in the same administrative domain.

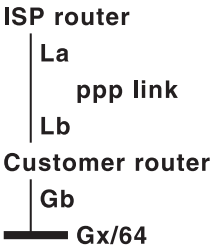
- Dynamic address allocation.

Where to assign address

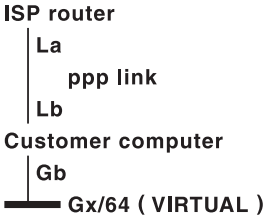
- Assign address to ppp interface. The form assigns /128 to the customer computer, or /64 onto the PPP link. The form of address assignment should not be used, as it advocates IPv6-to-IPv6 NAT. In the following diagram, "Lx" denotes link-local address, and "Gx" denotes global address.



- Assign address to the interface behind the customer router. The form assigns /64 to the network segment behind customer router.

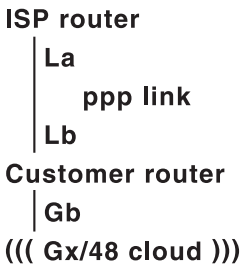


In the cases where the customer has only a single computer, it is possible to have virtual network segment behind the customer computer.



Note that, however, /64 assignment will make it harder for customer site to evolve in the future. /64 assignment is not recommended.

- Assign address to the cloud behind the customer router (/48). In this case, the upstream ISP has no idea about the topology in the customer site. Actually, it is not necessary for the upstream ISP to know about the address usage in the customer site. Static address assignment is highly recommended in this case, as it is painful to renumber the whole /48 cloud every time we reconnect the dialup PPP link between the ISP and the customer site.



Routing

- Static routing. ISPs will configure static route, pointing to the customer site, for the address space assigned to the customer site. Customer router (or customer computer) will install default route, pointing to the ISP router via PPP link.
- Simple dynamic routing. ISPs can exchange routes by using simple dynamic routing protocols like RIPng. This allows the customer site to adapt to PPP link status better. This also makes it easier to detect PPP link disconnection. If the ISP announces non-default routes to the downstream customer, it may help downstream to configure multihomed connectivity (connection to multiple upstream ISPs) [57] ISPs may want to filter out bogus routing announcements from the downstream.

第 9 章　Guidelines for IPv6 local experiments

9.1 Problem space

There are potential IPv6 users who would like to perform experiments locally, in their IPv6 network disjoint from the worldwide IPv6 network at large (or 6bone).

Site-local address [35] could be used where appropriate. However, site-local address has several operational differences from global address (like below), and it is harder for novice users to configure site-local address right than global address. Also, due to the differences, some of the things the user learnt from local experiments may not be directly relevant when they get connected to the worldwide IPv6 network - it reduces usefulness of their local experiments.

- Site-local addresses are "scoped" address, while global addresses are not.
- Configuration must correctly identify site border routers. This is an additional requirement.
- There are proposals on scoped routing exist [Deering, 2000] , however, implementation status is still rather disappointing.

For experiments over single link, link-local address could be used. However, again, link-local address is a scoped address, and has radical operational differences from global IPv6 (or IPv4) address.

9.2 Recommendations

First of all, do not cook up IPv6 prefix on your own. You cannot pick random prefix number, that can jeopadize the whole point of experiment.

Next, it is recommended to use global addresses for early stage of experiments. As presented in the previous section, scoped (site- local/link-local) IPv6 addresses have different operational characteritics from global IPv6 addresses, or global IPv4 address. In the later stage of experients, you may want to play with scoped addresses, and try to understand how they behave.

In the following text of the draft, we list possible routes a disconnected IPv6 network may want to take. Once the site gets connected to worldwide IPv6 network at large, the site MUST be renumbered to the addresses assigned by the (real) upstream.

A site with 6bone site/IPv6 ISP nearby

If it is possible, try to contact nearest upstream 6bone site, or upstream ISP, to assign you an IPv6 prefix. By getting IPv6 address space properly, the site will have less problem when they get conneted to the worldwide IPv6 network. The address space can (supposedly) be used for future IPv6 upstream connectivity, if you connect your site to the upstream which assigned the address space to you. If you pick a different upstream, the site MUST be renumbered.

To measure "nearness" between you and the upstream IPv6-over-IPv4 tunnel [53] provider (like a 6bone site), use IPv4 hop counts.

A site with global IPv4 connectivity

Whenever the site has a global IPv4 address with it, the site should use the 6to4 IPv6 address prefix [23] derived from the IPv4 address space, for local experiments. The prefix will be 2002:xxyy:zzuu::/48, where "xxyy:zzuu" is a hexadecimal notation of an global IPv4 address that belongs to the site.

However, such a /48 prefix can never be routed globally in the world IPv6 network in a normal sense. When the site wants to get external IPv6 connectivity, it must if possible renumber to a normal IPv6 prefix from its ISP (or 6bone upstream). Otherwise it must find a 6to4 relay router to connect it to the IPv6 world. For detailed discussion about how packets from 6to4 site are handled, please refer to 6to4 document.

Completely disconnected site

If the site has no permanent global IPv4 address with it (like dialup customer site), the site has two choices.

- The site may use site local address space. The operation needs great care as presented above.
- The site may use the address prefix: 3ffe:0501:ffff::/48. The address prefix was curved out from WIDE 6bone prefix. The site MUST be renumbered, before the site gets connected to the worldwide IPv6 network. The address is provided as the last-resort solution. The site should first try to use other ways.

In both cases, the assigned prefix MUST NOT be advertised to the worldwide IPv6 network, from anywhere.

Other comments

If there are multiple administrative domains in the site, the site is responsible for its internal coordination (the draft cannot solve your local politics).

第10章 Socket API for IPv6 flow label field

10.1 Background

The IPv6 flow label field is a 20bit field in the IPv6 header. The field has no IPv4 counterpart. The IPv6 specification [65] supplies suggested usage of the field.

The field is intended to identify a "flow", a set of packets from a particular source to a particular destination. The flow label field is set by the originating IPv6 node, in a pseudorandom manner. The value will help intermediate routers to identify "flows", without looking into payload data or chasing an extension header chain.

For the flow label field to be useful, the source node should carefully pick the value, to satisfy the following constraints:

- The value should be pseudorandom, to help routers make a hash table with it.
- The value should not be used for multiple different flows at the same time.
- The value should not be reused for some amonut of time, after a flow is terminated (otherwise, intermediate routers may mistakenly identify flows).

IPv6 specification does not define whether the field can be rewritten by intermediate routers, or the field should be kept untouched. It was to let future QoS protocols make the choice. For example, RSVP [Braden, 1997] assumes that the field is kept untouched until the packet reaches the final destination. In this document, following the assumption in the RSVP document, we assume that the field should not be modified by intermediate routers.

There is no known application which needs to inspect the flow label field on inbound packet. Also, there is no known application which wants to put a specific value to the flow label field.

10.2 Outbound traffic

After the connect(2) system call is issued for a socket with a specific IPv6 address (non-unspecified address), the kernel will automatically fill in the flow label field, with a value selected for the socket. The value will be selected on connect(2), and will be used for subsequent outgoing packets from the socket. The kernel is responsible to pick a suitable (pseudorandom and unique) value for the flow label field.

If no connect(2) system call was issued for a socket, the packets from the socket will have an unspecified flow label value (zero). When multiple connect(2) system calls were issued for a socket, a new value must be picked for the flow label field, every time the connect(2) system call was issued.

With getsockname(2), an application can grab the flow label value picked by the kernel, into the sin6_flowinfo member. sin6_flowinfo member carries the value in network byteorder. The topmost 12 bits of the sin6_flowinfo member must be set to 0.

Sample code would be as follows:

```
struct sockaddr_in6 src, dst, altdst;
u_int32_t value;
  /* the value for flow label */
int s;            /* socket */
socklen_t slen;

slen = sizeof(dst);
dst.sin6_flowinfo = 0;  /* must be zero */
connect(s, (struct sockaddr *)&dst, slen);

/* sent with the flow label field filled */
send(s, buf, buflen);

/* obtain the flow label value */
slen = sizeof(src);
getsockname(s, (struct sockaddr *)&src,
            &slen);
printf("flowlabel=%x\n",
       ntohl(src.sin6_flowinfo &
       IPV6_FLOWLABEL_MASK));
```

If an application wishes to disable the kernel behavior and wishes to use an unspecified value (zero) in the flow label field, the application should issue the following setsockopt(2), prior to the connect(2) system call. The default value for the socket option is implementation- dependent. A portable application should inspect the initial setting by using getsockopt(2).

```
const int off = 0;
const int on = 1;
int s;  /* socket */

/* disables automatic flow label */
setsockopt(s, IPPROTO_IPV6,
           IPV6_AUTOFLOWLABEL,
           &off, sizeof(off));
/* enables automatic flow label */
setsockopt(s, IPPROTO_IPV6,
           IPV6_AUTOFLOWLABEL, &on,
           sizeof(on));
```

The kernel should honor the definition of "flow" when filling in flow label field. For example, let us consider the suggestion in the IPv6 specification [65] . In the following cases, the kernel should make a special consideration. The kernel should fill the flow label field with an unspecified value (zero), or pick a new value.

(a) The packet goes to a different IPv6 destination, from the destination specified previously with a connect(2) system call. The situation happens with sendto(2) or sendmsg(2) system calls with a destination specified.

(b) The packet uses a different IPv6 source address than before. It happens when a bind(2) system call is issued.

(c) The packet carries different IPv6 extension headers from we have previously used. The situation could be detected by the use of IPv6 advanced API setsockopt(2), or by the presense of ancillary data items on sendmsg(2).

10.3 Inbound traffic

We define no option to inspect the flow label field on inbound traffic, at this moment.

Even though we are able to grab the outgoing flow label value with getsockname(2), the value should not affect the socket selection against inbound traffic.

Note: Even if we are to define some mechanism to inspect the value on inbound packets, we should not use the sin6_flowinfo member for this. There are many applications which do connect(2) or sendto(2), with the value returned from recvfrom(2) or getpeername(2).

10.4 sin6_flowinfo field

The draft defines no valid operation where a value is passed, from an application to the kernel, via the sin6_flowinfo member. When the application issues system calls to the kernel, the application should fill the sin6_flowinfo member with 0, as suggested in IPv6 basic API.

10.5 Issues

- Interaction with RSVP. Is getsockname(2)

enough to implement RSVP application?

- Is it necessary for an application to specify the flow label value manually? In this case, how should we check if the value is suitable enough? (how to check the number collision?)
- The document assumes that the granularity of flows is equal to the granularity of sockets, or connect(2) system calls. As we still do not have wide consensus about what the word "flow" means, this could be controversial; for example, some may want multiple flows for a TCP session, some may want to consider multiple TCP sessions as a single flow.

10.6 Security consideration

The document introduces no new security issue.

The presense of a flow label value may help wiretappers to identify a flow out of packets on the wire.

第11章 An IPv6/IPv4 multicast translator based on IGMP/MLD Proxying (IMP)

11.1 Introduction

It is expected that many IPv4 nodes will remain, for its success, for a long time after the transition to IPv6 starts. On the other hand IPv6-only nodes will appear, for cost reasons or as a result of exhaustion of the IPv4 address space, before IPv4 nodes disap- pear. Therefore, it is highly desirable to develop a mechanism which enables direct communication between IPv4 nodes and IPv6 nodes, in order to advance the transition smoothly. [116] and [156] have already proposed such mechanisms, but they are applied only to unicast communication, not to multicast. So it is necessary to provide another mechanism for multicast.

This memo describes an entire scheme of multicast communication between IPv4 nodes and IPv6 nodes. The scheme is composed by a mul- ticast translator and an address mapper who are located at the site boundary between IPv4 and IPv6. It is not necessary to modify IPv4 nodes and IPv6 nodes.

This memo uses the words defined in [128], [35], and [53].

11.2 Components

This section describes components needed for the mechanism.

The system consists of a multicast translator, and an address mapper. In order to allow IPv4 nodes and IPv6 nodes to directly communicate using multicast, they need to be installed on the site boundary between IPv4 and IPv6. Figure 1 illustrates the network system interconnected by them.

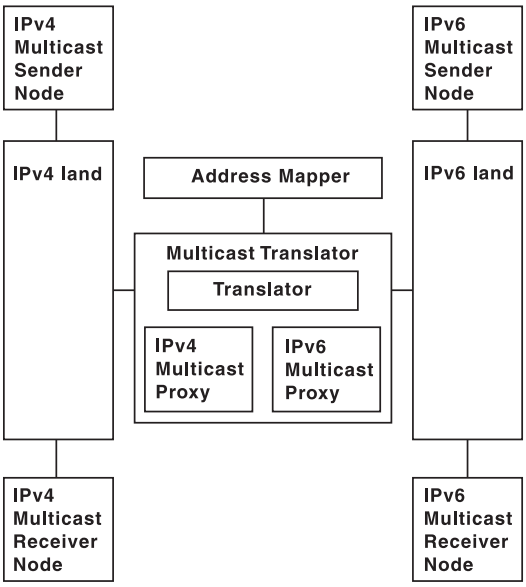


Fig.1 Network system

Multicast Translator

It locates between an IPv4 land and an IPv6 land, and translates IPv4 multicast packets into IPv6 multicast packets and vice versa. It consists of the following three sub-components.

1. Translator

   It is a component which translates IPv4 multicast packets into IPv6 multicast packets and vice versa. There are several trans- lation types.

Gateway

It terminates data bound for an IPv4 multicast group at application layer, and relays the data to an IPv6 multicast group and vice versa.

Header conversion router

When receiving an IPv4 multicast packet, it converts the IPv4 header into an IPv6 header, fragments the IPv6 packet if necessary, and then forwards the packet. Likewise, when receiving an IPv6 multicast packet, it converts the IPv6 header into an IPv4 header, and then forwards the IPv4 packet.

2. IPv4 Multicast Proxy

   It joins IPv4 multicast groups as a proxy of IPv6 receiver nodes. Thereby it receives packets bound for the IPv4 multicast groups, and then hands the packets to the translator.

3. IPv6 Multicast Proxy

   It joins IPv6 multicast groups as a proxy of IPv4 receiver nodes. Thereby it receives packets bound for the IPv6 multicast groups, and then hands the packets to the translator.

Address mapper

It maintains each unicast address spool for IPv4 and IPv6. The IPv4 spool, for example, consists of private addresses [133] bound for the multicast translator. An example of the IPv6 spool is IPv6 address space assigned to virtual IPv6 organization on the IPv4 land.

Also, it maintains a mapping table which consists of pairs of an IPv4 address and an IPv6 address. When the translator (or the IPv4 Proxy or the IPv6 Proxy) requests it to assign an IPv6 address corresponding to an IPv4 address, it selects a proper IPv6 address out of the table, and returns the address to the translator. When there is not a proper entry for an IPv4 unicast address, it selects and returns an IPv6 unicast address out of the spool, and registers a new entry into the table. When there is not a proper entry for an IPv4 multicast group address, it registers a new entry, which con- sists of the IPv4 multicast group address and that of IPv6 corresponding to the IPv4 address, into the table. The IPv6 address is a special type of one proposed in this memo. See section 4.

When the translator (or the IPv4 Proxy or the IPv6 Proxy) requests it to assign an IPv4 address corresponding to an IPv6 address, it works like the above.

11.3 Interaction Examples

This section explains communication from one IPv4 multicast sender node to one or more IPv6 multicast receiver nodes, and communica- tion from one IPv6 multicast sender node to one or more IPv4 multi- cast receiver nodes, respectively.

Communication from IPv4 to IPv6

The following subsection explains communication from one IPv4 mul- ticast sender node, called "sender4," to one or more IPv6 multicast receiver nodes, called "receiver6."

Preceding the communication, the administrator of the multicast translator carries out the setup to translate IPv4 multicast pack- ets, which are sent by "sender4", into IPv6. According to the direction of the administrator, the IPv4 multicast proxy joins the IPv4 multicast group as a proxy of "receiver6", and then registers a new entry, which consists of the IPv4 multicast group address and that of IPv6 corresponding to the IPv4 address, into the mapping table. The IPv6 address is a special type of one proposed in this memo, and takes the structure which is identified by a prefix of ffxx::/96 and holds the IPv4 address in the low-order 32-bits. See section 4.

The communication is triggered by "sender4." "sender4" sends an IPv4 multicast packet.

When the packet arrives at the multicast translator, the IPv4 mul- ticast proxy receives it and hands it to the translator. The trans- lator tries to translate it into an IPv6 packet but does not

know how to translate the IPv4 source address and the IPv4 destination address. So the translator requests the mapper to tell mapping entries for them. The mapper checks its mapping table with each of them and finds only a mapping entry for the IPv4 destination address.

But there is not a mapping entry for the IPv4 source address, so the mapper selects an IPv6 address out of the IPv6 spool and regis- ters a new entry, which consists of the IPv4 address and the IPv6 address, into the mapping table. And then the mapper returns the IPv6 destination address and the IPv6 source address to the trans- lator.

After that the translator translates the packet to IPv6, fragments it if necessary, and forwards it. Note: The translation from the IPv4 source address to the IPv6 source address is unicast one.

Finally it arrives at "receiver6."

Figure 2 illustrates the interaction communicating from IPv4 to IPv6.

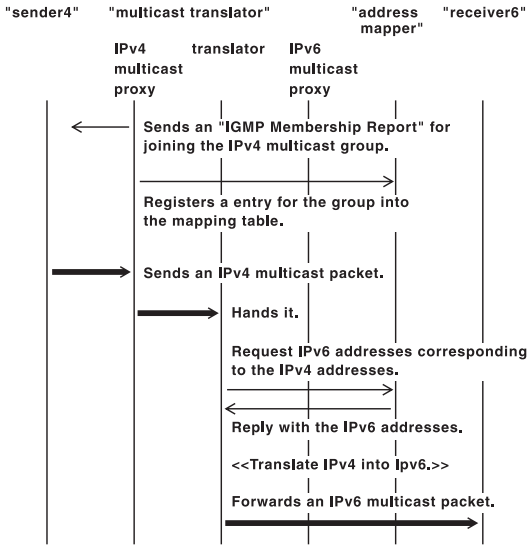


Fig.2 The interaction communicating from IPv4 to IPv6.

Communication from IPv6 to IPv4

The following subsection explains communication from one IPv6 mul- ticast sender node, called "sender6," to one or more IPv4 multicast receiver nodes, called "receiver4."

Preceding the communication, the administrator of the multicast translator carries out the setup to translate IPv6 multicast pack- ets, which are sent by "sender6" to a special type of IPv6 address proposed in this memo, into IPv4. In the case, the IPv6 multicast proxy joins the IPv6 multicast group as a proxy of "receiver4", and then registers a new entry, which consists of the IPv6 multicast group address and that of IPv4 corresponding to the IPv6 address, into the mapping table. The IPv4 address is the low-order 32-bits of the IPv6 address.

Subsequent interaction is symmetric to the case described in Sec- tion 3.1.

Figure 3 illustrates the interaction communicating from IPv6 to IPv4.

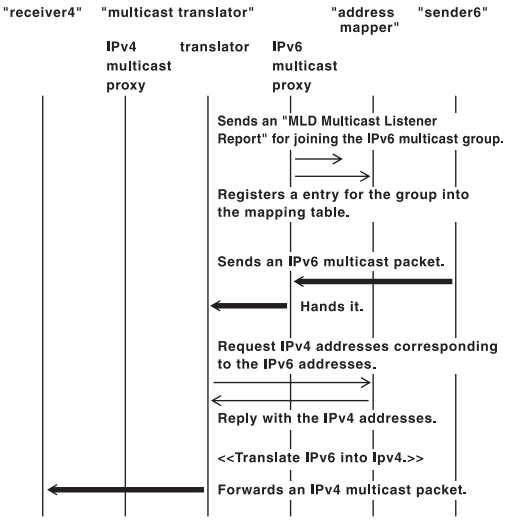


Fig.3 The interaction communicating from IPv6 to IPv4.

11.4 Addressing for IPv4/IPv6 multicast communication

The mechanism uses a special type of an IPv6 address which is termed an "IPv4-compatible" IPv6 multicast group address. The address is identified by an prefix for IPv6 multicast (ffxx::/96), and holds an IPv4 multicast group address in the low-order 32-bits. Its format is:

96-bits	32-bits
ffxx:0:0:0:0:0	IPv4 multicast group address

11.5 Applicability and Limitations

This section considers applicability and limitations.

Applicability

The multicast translator based on the mechanism locates at the site boundary between IPv4 and IPv6, and allows them to communicate directly. Therefore, the mechanism can be useful during a long term, until IPv4 nodes disappear after IPv6-only nodes appear.

It can be applicable to small-scale network systems, and to the extent of division networks in intranets where its administrator can operate the setup easily on demand by receivers.

Limitations

In common with NAT [86], IP conversion needs to translate IP addresses embedded in application layer protocols. So it is hard to translate all such applications completely.

It cannot be applicable to large-scale network systems like world- wide Internet because it needs the setup by its administrator. In order to apply it to large-scale network systems, it needs developing a new standard protocol between multicast translators and receivers for carrying out the setup automatically on demand by receivers.

11.6 Security considerations

Header conversions of AH [95] and ESP [96] may be cryptographi- cally impossible in header conversion router approach. It is a big disadvantage. On the other hand it will be possible to use both AH and ESP in proxy gateway approach.

第12章　A RADIUS attribute for IPv6 dialup PPP with static address assignment

12.1 Usage model

In this document we cover the following cases in IPv6 dialup PPP operation. No other cases are addressed in the document. Please refer to [58] for more about requirements in/categorization of IPv6 dialup PPP operation,

- The downstream customer is a “static client,” i.e. their computers does not change the geographical location.
- Address space is statically assigned to the downstream customer. In this document, we assume that we have assigned 3ffe:0501:ffff::/48 to the downstream customer.
- The /48 Address space is assigned for the network cloud behind the customer router. It should be noted that ISP router will have no idea about the network topology in the customer site. In the following diagram, Lx denotes an IPv6 link-local address, and Gx denotes an IPv6 global address.

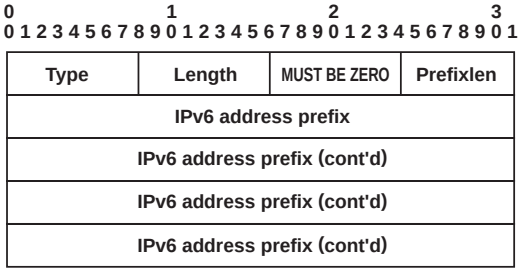
```
ISP router — RADIUS server
  |La
  |   ppp link
  |Lb
Customer router
  |Gb
(((3ffe:0501:ffff::/48 cloud)))
```

- Routing can be statically configured, or dynamic routing protocol like RIPng [103] should be used.
- We carry IPv6 packets using IPv6 PPP [Haskin, 1998]. No IPv6-over-IPv4 tunnelling is used.
- IPv4 PPP address allocation issue is outside of the scope of the document. We can safely ignore it without loss of generality, thanks to multiprotocol nature of PPP.

12.2 RADIUS attribute for carrying IPv6 address space information

To exchange the information about customer IPv6 address space between the ISP router and RADIUS server, we need a common RADIUS attribute for IPv6 address space. The attribute is

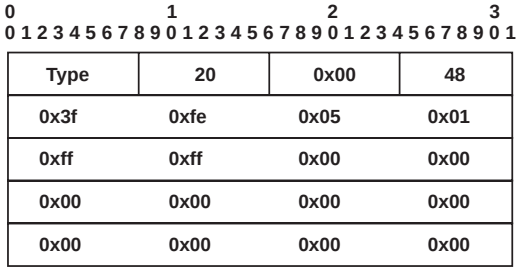
defined as follows:



Type	Length	MUST BE ZERO	Prefixlen
IPv6 address prefix			
IPv6 address prefix (cont'd)			
IPv6 address prefix (cont'd)			
IPv6 address prefix (cont'd)			

- Type: TBD
- Length: 20
- Prefixlen: IPv6 prefix length for the IPv6 address space. Between 0 to 128 (in decimal). For normal use, 48 (in decimal).
- IPv6 address prefix: Binary representation of IPv6 address, in network byte order. Bits outside of prefix length must be zero.

For example, if we exchange "3ffe:0501:ffff::/48" with it, the attribute will look like follows:



Type	20	0x00	48
0x3f	0xfe	0x05	0x01
0xff	0xff	0x00	0x00
0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00

12.3 Scenario

Contract

The ISP and the downstream customer will sign a IPv6 dialup PPP contract. The ISP will assign the downstream customer a global IPv6 address space. The ISP will inform the downstream customer of the following items:

- The phone number which the customer should dial (the number reaches ISP router), and
- The global IPv6 address space (3ffe:0501:ffff::/48).

The customer and ISP will exchange PPP authentication information in secrecy.

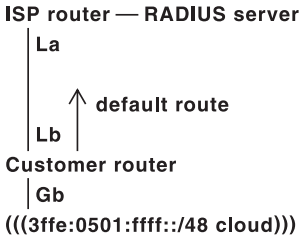
ISP configuration

The ISP will put the following information into RADIUS database.

- Account name of the user (which is the key for looking up RADIUS database),
- PPP authentication information, and
- The global IPv6 address space assigned to the customer (3ffe:0501:ffff::/48).

Customer site configuration

The customer will configure the customer site using the address space assigned. Customer installs default routing entry toward the ISP router.



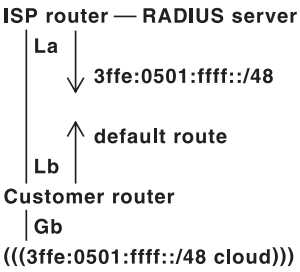
Establishing a PPP connection

On PPP link establishment, the following events would happen.

- The customer router calls up the ISP router.
- The customer router authenticates itself to the ISP router. The ISP router contacts RADIUS server for authentication information, to check if the user is legitimate.
- If the user is found to be legitimate, the PPP link will be established between the routers. IPV6CP [60] is used to avoid duplicated link-local address on the PPP link.
- The ISP router will grab information on IPv6 address space from RADIUS database (NOTE: the ISP router can grab the authentication information and the IPv6 address space information, in a single query).

If we use static routing the ISP router is con-

figured with a routing entry for the address space, pointing to the customer router. ISP router advertises the routing information to the ISP backbone, if necessary.



If we use dynamic routing, instead of configuring a static routing entry, the ISP router is configured to accept legitimate IPv6 routing announcements (that is, 3ffe:0501:ffff::/48) from the customer router. The ISP router advertises default route, and possibly more specific routes, to the customer router. The customer router will advertise 3ffe:0501:ffff::/48 to the ISP router (NOTE: the IPv6 address prefix is pre-configured to the customer router. The customer router knows the prefix as the ISP gave the information at the contract time).

Tearing down

On PPP link disconnection, the following events would happen.

- The customer router, or the ISP router, wishes to disconnect the PPP connection.
- PPP link will be teared down using normal PPP disconnection procedure.
- The ISP router removes the routing entry for the global IPv6 address space for the customer.

12.4 Discussions

It is possible to use the proposed attribute for non-dialup IPv6 PPP connections, to ease management in the ISP side.

It is possible to define the proposed RADIUS attribute as a vendor-type attribute under "Vendor-Specific" type (26). The authors are not sure which is the way to go.

第13章　An Extension of Format for IPv6 Scoped Addresses

13.1 Introduction

There are several types of scoped addresses defined in the "IPv6 Addressing Architecture" [64]. Since uniqueness of a scoped address is guaranteed only within a corresponding area of the scope, the semantics for a scoped address is ambiguous on a scope boundary. For example, when a user specifies to send a packet from a node to a link-local address of another node, the user must specify the link of the destination as well, if the node is attached to more than one link.

This characteristic of scoped addresses may introduce additional cost to scope-aware applications; a scope-aware application may have to provide a way to specify an instance of a scope for each scoped address (e.g. a specific link for a link-local address) that the application uses. Also, it is hard for a user to "cut and paste" a scoped address due to the ambiguity of its scope.

Applications that are supposed to be used in end hosts like telnet, ftp, and ssh, are not usually aware of scoped addresses, especially of link-local addresses. However, an expert user (e.g. a network administrator) sometimes has to give even link-local addresses to such applications.

Here is a concrete example. Consider a multi-linked router, called "R1", that has at least two point-to-point interfaces. Each of the interfaces is connected to another router, called "R2" and "R3". Also assume that the point-to-point interfaces are "unnumbered", that is, they have link-local addresses only.

Now suppose that the routing system on R2 hangs up and has to be reinvoked. In this situation, we may not be able to use a global address of R2, because this is a routing trouble and we cannot expect that we have enough routes for global

reachability to R2.

Hence we have to login R1 first, and then try to login R2 using link-local addresses. In such a case, we have to give the link-local address of R2 to, for example, telnet. Here we assume the address is fe80::2.

Note that we cannot just type like

```
% telnet fe80::2
```

here, since R1 has more than one interface (i.e. link) and hence the telnet command cannot detect which link it should try to connect.

Although R1 could spray neighbor solicitations for fe80::2 on all links that R1 attaches in order to detect an appropriate link, we cannot completely rely on the result. This is because R3 might also assign fe80::2 to its point-to-point interface and might return a neighbor advertisement faster than R2. There is currently no mechanism to (automatically) resolve such conflict. Even if we had one, the administrator of R3 might not accept to change the link-local address especially when R3 belongs to a different organization from R1's.

This document defines an extension of the format for scoped addresses in order to overcome this inconvenience. Using the extended format with some appropriate library routines will make scope-aware applications simpler.

13.2 Proposal

The proposed format for scoped addresses is as follows:

```
<scoped_address>%<scope_id>
```

where `<scoped_address>` is a literal IPv6 address, `<scope_id>` is a string to identify the scope of the address, and '`%`' is a delimiter character to distinguish between `<scoped_address>` and `<scope_id>`.

The following subsections describe detail definitions and concrete examples of the format.

Scoped Addresses

The proposed format is applied to all kinds of unicast and multicast scoped addresses, that is, all non-global unicast and multicast addresses.

The format should not be used for global addresses. However, an implementation which handles addresses (e.g. name to address mapping functions) MAY allow users to use such a notation (see also Appendix C).

Scope Identifiers

An implementation SHOULD support at least numerical identifiers as <scope_id>, which are non-negative decimal numbers. Positive identifiers MUST uniquely specifies a single instance of scope for a given scoped address. An implementation MAY use zero to have a special meaning, for example, a meaning that no instance of scope is specified.

An implementation MAY support other kinds of strings as <scope_id> unless the strings conflict with the delimiter character. The precise semantics of such additional strings is implementation dependent.

One possible candidate of such strings would be interface names, since interfaces uniquely disambiguate any type of scopes [55]. In particular, if an implementation can assume that there is a one-to-one mapping between links and interfaces (and the assumption is usually reasonable,) using interface names as link identifiers would be natural.

An implementation could also use interface names as <scope_id> for larger scopes than links, but there might be some confusion in such use. For example, when more than one interface belongs to a same site, a user would be confused about which interface should be used. Also, a mapping function from an address to a name would encounter a same kind of problem when it prints a scoped address with an interface name as a scope identifier. This document does not specify how these cases should be treated and leaves it implementation dependent.

It cannot be assumed that a same identifier is common to all nodes in a scope zone. Hence the proposed format MUST be used only within a node and MUST NOT be sent on a wire.

Examples

Here are examples. The following addresses

```
fe80::1234 (whose link identifier is 1)
fec0::5678 (whose site identifier is 2)
ff02::9abc (whose link identifier is 5)
ff08::def0
  (whose organization identifier is 10)
```

would be represented as follows:

```
fe80::1234%1
fec0::5678%2
ff02::9abc%5
ff08::def0%10
```

If we use interface names as <scope_id>, the followings could also be represented as follows:

```
fe80::1234%ne0
fec0::5678%ether2
ff02::9abc%pvc1.3
ff08::def0%interface10
```

where the interface "ne0" belongs to link 1, "ether2" belongs to site 2, and so on.

Omitting Scope Identifiers

This document does not intend to invalidate the original format for scoped addresses, that is, the format without the scope identifier portion. An implementation SHOULD rather provide a user with a "default" instance of each scope and allow the user to omit scope identifiers.

Also, when an implementation can assume that there is no ambiguity of any type of scopes on a node, it MAY even omit the whole functionality to handle the proposed format. An end host with a single interface would be an example of such a case.

13.3 Combinations of Delimiter Characters

There are other kinds of delimiter characters defined for IPv6 addresses. In this section, we describe how they should be combined with the proposed format for scoped addresses.

The IPv6 addressing architecture [64] also defines the syntax of IPv6 prefixes. If the address portion of a prefix is scoped one and the scope should be disambiguated, the address portion SHOULD be in the proposed format. For example, the prefix fec0:0:0:1::/64 on a site whose identifier is 2 should be represented as follows:

```
fec0:0:0:1::%2/64
```

There is the preferred format for literal IPv6 addresses in URL's [63]. When a user types the preferred format for an IPv6 scoped address and the scope should be explicitly specified, the address part in brackets SHOULD be in the proposed format. Thus, for instance, the user should type as follows:

```
http://[fec0:0:0:2::1234%10]:80/index.html
```

13.4 Related Issues

In this document, it is assumed that an identifier of a scope is not necessarily common in a scope zone. However, it would be useful if a common notation is introduced (e.g. an organization name for a site). In such a case, the proposed format could be commonly used to designate a single interface (or a set of interfaces for a multicast address) in a scope zone.

When the network configuration of a node changes, the change may affect <scope_id>. Suppose that the case where numerical identifiers are sequentially used as <scope_id>. When a network interface card is newly inserted in the node, some identifiers may have to be renumbered accordingly. This would be inconvenient, especially when addresses with the numerical identifiers are stored in non-volatile storage and reused after rebooting.

13.5 Security Considerations

The use of this approach to represent IPv6 scoped addresses does not introduce any known new security concerns, since the use is restricted within a single node.

13.6 Appendix A. Interaction with API

The proposed format would be useful with some library functions defined in the "Basic Socket API" [148], the functions which translate a nodename to an address, or vice versa.

For example, if getaddrinfo() parses a literal IPv6 address in the proposed format and fills an identifier according to ¡scopde_id¿ in the sin6_scope_id field of a sockaddr_in6 structure, then an application would be able to just call getaddrinfo() and would not have to care about scopes.

Also, if getnameinfo() returns IPv6 scoped addresses in the proposed format, a user or an application would be able to reuse the result by a simple "cut and paste" method.

Note that the ipng working group is now revising the basic socket API in order to support scoped addresses appropriately. When the revised version is available, it should be preferred to the description of this section.

13.7 Appendix B. Implementation Experiences

The WIDE KAME IPv6 stack implements the extension to the getaddrinfo() and the getnameinfo() functions described in Appendix A of this document. The source code is available as free software, bundled in the KAME IPv6 stack kit.

The current implementation assumes that there is one-to-one mapping between links and interfaces, and hence it uses interface names as <scope_id> for links.

For instance, the implementation shows its routing table as follows:

```
Internet6:
Destination Gateway             Flags Intface
default     fe80::fe32:93d1%ef0 UG    ef0
```

This means that the default router is fe80::fe32:93d1 on the link identified by the interface "ef0". A user can "cut and paste" the result in order to telnet to the default router like this:

```
% telnet fe80::fe32:93d1%ef0
```

even on a multi-linked node.

As another example, we show how the implementation can be used for the problem described in Section 1.

We first confirm the link-local address assigned to the point-to-point interface of R2:

```
(on R1)% ping ff02::1

PING(56=40+8+8 bytes) fe80::1 --> ff02::1
16 bytes from fe80::1%lo0, icmp_seq=0 hlim=64 time=0.474 ms
16 bytes from fe80::2%pvc0, icmp_seq=0 hlim=64 time=0.374 ms(DUP!)
...
(we assume here that the name of the point-to-point interface
on R1 toward R2 is "pvc0" and that the link-local address on
the interface is "fe80::1".)
```

So the address should be fe80::2. Then we can login R2 using the address by the telnet command without ambiguity:

```
% telnet fe80::2%pvc0
```

Though the implementation supports the extended format for all type of scoped addresses, our current experience is limited to link-local addresses. For other type of scopes, we need more experience.

13.8 Appendix C. A Comprehensive Description of KAME's getXXXinfo Functions

The following tables describe the behavior of the KAME's implementation we mentioned in Appendix B using concrete examples. Note that those tables are not intended to be standard specifications of the extensions but are references for other implementors.

Those tables summarize what value the getXXXinfo functions return against various arguments. For each of two functions we first explain typical cases and then show non-typical ones.

The tables for getaddrinfo() have four columns. The first two are arguments for the function, and the last two are the results. The tables for getnameinfo() also have four columns. The first three are arguments, and the last one is the results.

Columns "Hostname" contain strings that are numeric or non-numeric IPv6 hostnames.

Columns "NI_NUMERICHOST" show if the NI_NUMERICHOST is set to flags for the corresponding getXXXinfo function. The value "1" means the flag is set, and "0" means the flag is clear. "-" means that the field is not related to the result.

Columns "sin6_addr" contain IPv6 binary ad-

dresses in the textual format, which mean the values of the sin6_addr field of the corresponding sockaddr_in6 structure.

Columns “sin6_scope_id” contain numeric numbers, which mean the values of the sin6_scope_id field of the corresponding sockaddr_in6 structure.

If necessary, we use an additional column titled “N/B” to note something special.

If an entry of a result column has the value “Error,” it means the corresponding function fails.

In the examples, we assume the followings: - The hostname “foo.kame.net” has a AAAA DNS record “3ffe:501::1”. We also assume the reverse map is configured correctly. - There is no FQDN representation for scoped addresses. - The numeric link identifier for the interface “ne0” is 5. - We have an interface belonging to a site whose numeric identifier is 10. - The numeric identifier “20” is invalid for any type of scopes. - We use the string “none” as an invalid non-numeric scope identifier.

Typical cases for getaddrinfo():

```
Hostname NI_NUMERICHOST sin6_addr sin6_scope_id
"foo.kame.net" 0 3ffe:501::1  0
"3ffe:501::1" - 3ffe:501::1  0
"fec0::1"fe80::1"fe80::1
```

Typical cases for getnameinfo():

```
sin6_addr sin6_scope_id NI_NUMERICHOST Hostname N/B
3ffe:501::1 0 0 "foo.kame.net"
3ffe:501::1 0 1 "3ffe:501::1"
fec0::1 10 - "fec0::1%10"
fe80::1 5 - "fe80::1%ne0" (*1)
```

(*1) Regardless of the NI_NUMERICHOST flag, we always show an interface name as the <scope_id> portion for a link-local address if the identifier is valid.

Non-typical cases for getaddrinfo():

```
Hostname NI_NUMERICHOST sin6_addr
sin6_scope_id N/B
"foo.kame.net" 1 Error
"foo.kame.net%20" - Error (*2)
"foo.kame.net%none" - Error (*2)
"3ffe:501::1%none" - Error
"3ffe:501::1%0" - 3ffe:501::1 0 (*3)
"3ffe:501::1%20" - 3ffe:501::1 20 (*3)
"fec0::1%none" - Error
"fec0::1" - fec0::1 0 (*4)
"fec0::1%0" - fec0::1 0 (*5)
"fec0::1%20" - fec0::1 20 (*6)
"fe80::1%none" - Error
"fe80::1" - fe80::1 0 (*4)
"fe80::1%0" - fe80::1 0 (*5)
"fe80::1%20" - fe80::1 20 (*6)
```

(*2) <scope_id> against an FQDN is invalid.

(*3) We do not expect that <scope_id> is specified for a global address, but we don’t regard it as invalid.

(*4) We usually expect that a scoped address is specified with <scope_id>, but if no identifier is specified we just set 0 to the sin6_scope_id field.

(*5) Explicitly specifying 0 as <scope_id> is not meaningful, but we just treat the value as opaque.

(*6) The <scope_id> portion is opaque to getaddrinfo() even if it is invalid. It is kernel’s responsibility to raise errors, if there is any connection attempt that the kernel cannot handle.

Non-typical cases for getnameinfo():

```
sin6_addr sin6_scope_id NI_NUMERICHOST
Hostname N/B
3ffe:501::1 20 1 "3ffe:501::1%20" (*7)
3ffe:501::1 20 0 "foo.kame.net" (*8)
fec0::1 20 - "fec0::1%20"
fec0::1 0 - "fec0::1" (*9)
fe80::1 20 - "fe80::1%20"
fe80::1 0 - "fe80::1" (*9)
```

(*7) We do not expect that a global IPv6 address has a non-zero scope identifier. But if it is the case, we just treat it as opaque.

(*8) Despite the above, if the NI_NUMERICHOST is clear, we resolve the address to a hostname and print the name without scope information. We might have to reconsider this behavior.

(*9) We usually expect that a scoped address has a non-zero scope identifier. But if the identifier is 0, we simply print the address portion

without scope information.

第14章 IPv6 SMTP operational requirements

14.1 Summary of IPv4 MX operation

For reference purpose, the section outlines how mail message delivery is performed in IPv4-only environment [125].

In IPv4 SMTP operation, we register MX records like below, for "sample.org." domain:

```
sample.org.       IN MX 1  mx1.sample.org.
                  IN MX 10 mx10.sample.org.
mx1.sample.org.   IN A  1.0.0.1
mx10.sample.org.  IN A  1.0.0.2
```

When an MTA delivers a message to a particular destination (say it is to foo@sample.org), the MTA would send DNS queries to lookup DNS database in the following order:

- Lookup MX record for "sample.org."
- If an MX record is returned, try to lookup `A` record on the righthand side of the MX record.
- If a CNAME record is returned, try to chase the CNAME chain. Eventually we will reach some `A` record.
- If MX lookup failed with NO_DATA, it means that there is no MX record but there can be other record for "sample.org." Lookup `A` record for "sample.org.".
- If MX lookup failed with HOST_NOT_FOUND, it means that there is no record at all for "sample.org." This means a delivery failure.

14.2 MX records and IPv6 SMTP operation

The following sections talk about how to make IPv4 SMTP and IPv6 SMTP coexist, under dual-stack environment during the transition period between IPv4 to IPv6. In the future, when we have completely migrated to IPv6-only network, we can forget about IPv4/v6 SMTP interaction.

As IPv6 DNS lookup RFCs [Thomson, 1995; Crawford, 2000] use IN class for both IPv4 and IPv6, we will use IN MX records for both IPv4 and IPv6.

For simplicity, the document lists DNS records for IPv6 address as `AAAA` records, not as A6 records [30]. In reality, we can use a chain of A6 records, instead of `AAAA` records.

There are couple of technologies defined for IPv4 and IPv6 transition. The document concentrates on issues with dual stack environment. Translators do not need special consideration from SMTP point of view; If we have SMTP traffic from IPv6 MTA to IPv4 MTA over an IPv6-to-IPv4 translator, the traffic will be considered as a normal IPv4 SMTP traffic, from the IPv4 MTA point of view. We may, however, need some consideration on translators for protocols like IDENT [149] .

14.3 SMTP sender algorithm in dual stack environment

When we lookup MX records for the domain in IPv4/v6 dual stack environment, we will see records like below:

```
sample.org.       IN MX   1  mx1.sample.org.
                  IN MX   10 mx10.sample.org.
mx1.sample.org.   IN A    1.0.0.1
                        ; IPv4/v6 dual stack
                  IN AAAA 3ffe:501:ffff::1
mx10.sample.org.  IN AAAA 3ffe:501:ffff::2
                                ; IPv6 only
```

For single MX record, we have many possibility for the final lookup result, including: (a) single, or multiple `A` records for IPv4 destination, (b) single, or multiple `AAAA` records for IPv6 destination, (c) mixture of A and `AAAA` records. As we can define multiple MX records with different preference value, we also need to go through multiple addresses based on multiple MXes. We need to cope with domains without MX records, and failure recovery cases too.

The algorithm for a SMTP sender would be like this.

1. Lookup MX record for the destination domain. If a CNAME record is returned, go back to step (1) with the queried result. If MX records are returned, go to step (2) with the result. If NO_DATA is returned, go to step (3) as there is no MX record. If HOST_NOT_FOUND is returned, there is no domain, raise permanent email delivery failure (finish).
2. We have multiple MX records with us. Loop steps from (3) to (8), based on MX preference values, in ascending order.
3. If the source MTA has IPv4 capability, lookup `A` record. Keep the resulting address till step (5).
4. If the source MTA has IPv6 capability, lookup `AAAA` record.
5. Reorder queried result based on implementation-dependent preference between A and `AAAA` records.
6. Loop steps from (7) to (8), for all the addresses (or part of the list of addresses) we have. If no reachable destination is found, and if we are going through a list of MX records, go back to (3) and try the next MX record. If we do not have a list of MX records, or we have reached the end of the list of MX records, raise temporary delivery failure (finish).
7. Try to make a TCP connection to the destination. If it fails, try the next address we have. If it succeeds, go to step (8).
8. Try a SMTP protocol negotiation. If SMTP protocol negotiation fails with TEMPFAIL (4xx), go back to (3) and try the next MX record. If it succeeds, SMTP delivery was successful (finish).

14.4 MX configuration in receipient domain

Ensuring reachability for both protocol versions

If a site has IPv4/v6 dual stack reachability, the site SHOULD configure both A and `AAAA` records onto its MX hosts. It will help both IPv4 and IPv6 senders to reach the site efficienlty.

Reachability between primary and secondary MX

When we configure MX records onto DNS database in dual-stack environment, we need to be careful about reachability between MX hosts. Suppose we try to gather all inbound email to primary MX host, mx1.sample.org.

```
sample.org. IN MX 1   mx1.sample.org.
            IN MX 10  mx10.sample.org.
            IN MX 100 mx100.sample.org.
```

If mx1.sample.org is an IPv6 only node and the rest are IPv4 only node, we have no reachability between primary MX host and the rest. Once an email reaches one of secondary MX host, the email will never reach the primary MX.

```
; the configuration is troublesome.
; no secondary MX can reach mx1.sample.org.
sample.org. IN MX 1   mx1.sample.org.
                          ; IPv6 only
            IN MX 10  mx10.sample.org.
                          ; IPv4 only
            IN MX 100 mx100.sample.org.
                          ; IPv4 only
```

The easiest possible configuration is to configure the primary MX host as an IPv4/v6 dual stack node. By doing so, secondaries will have no problem reaching the primary MX host.

```
; the configuration works just fine.
; emails reaches from secondary MX
; to primary with no trouble.
sample.org. IN MX 1   mx1.sample.org.
                 ; IPv4/v6 dual stack
            IN MX 10  mx10.sample.org.
                          ; IPv4 only
            IN MX 100 mx100.sample.org.
                          ; IPv6 only
```

There are many other ways to ensure the reachability between secondary MX and primary MX. For example, we could configure secondary MX to route emails statically, without considering DNS MX configuration. Or we could estalish alterna-

tive email routing path (i.e. UUCP, or via IPv4/v6 translator) between secondary MX and the primary MX.

14.5 Open issues

- How to interpret scoped address on MTAs. As we relay emails between MTAs, interpretation of scoped address can be different between MTAs, as intermediate MTAs may be in different scope zone as the originator.

If we get scoped IPv6 address as a result of DNS lookups, how MTAs should behave? If we consider scoped address in "route-addr" specification [31] like

<itojun@kame.net@oshokuji.org@itojun.org>

it gets more trickier.

14.6 Security consideration

The document should have no new security problem.

第15章 Socket API for IPv6 traffic class field

15.1 Background

The IPv6 traffic class field is a 8bit field in the IPv6 header. The field serves just like the IPv4 type of service (TOS) field. There are two types of proposed use of the field: (1) topmost 6 bits for the differentiated services (diffserv) field [113] , and (2) lowermost 2 bits for explicit congestion notification (ECN) [131]. Those two proposals plan to rewrite the field at intermediate routers.

There is a certain set of applications which need to manipulate and inspect the traffic class field. Here are some examples.

- ECN implementations outside of the kernel (like UDP ECN).
- A diffserv-aware application, which tries to mark low-priority traffic (such as non-important packets in a video traffic) on its own. In this case, the application does not need to inspect the field on outbound traffic.
- Debugging tools for differentiated services.

15.2 Inbound traffic

When an application is interested in inspecting the traffic class field on packets, the application should set the IPV6_RECVTCLASS socket option to 1:

```
/* enable */
const int on = 1;
setsockopt(fd, IPPROTO_IPV6,
           IPV6_RECVTCLASS, &on,
           sizeof(on));
```

Subsequent incoming traffic will be accompanied with an ancillary data item that carries an unsigned octet value. The ancillary data item will be tagged with the level IPPROTO_IPV6 and type IPV6_TCLASS. An application can obatain the value of the traffic class field by the following operation, after the recvmsg(2) system call:

```
struct cmsghdr *cm;
u_int8_t tclass;
if (cm->cmsg_len
     == CMSG_LEN(sizeof(u_int8_t)) &&
    cm->cmsg_level == IPPROTO_IPV6 &&
    cm->cmsg_type == IPV6_TCLASS)
    tclass = *(u_int8_t *)CMSG_DATA(cm);
else
    tclass = 0x00;
/* could not obtain traffic class value */
```

By setting the socket option to 0, the behavior is disabled:

```
/* disable */
const int off = 0;
setsockopt(fd, IPPROTO_IPV6,
           IPV6_RECVTCLASS, &off,
           sizeof(off));
```

For TCP sockets, an ancillary data item will be present only when the traffic class value is changed. See section 4.1 (TCP Implications) of

[148] for details.

15.3 Outbound traffic

To control the value of the traffic class field for a single packet transmission, you can use an ancillary data item, just like presented above, with a sendmsg(2) system call. The level of the ancillary data item must be IPPROTO_IPV6, and the type must be IPV6_TCLASS.

```
int s;  /* socket */
u_int8_t tclass;
struct sockaddr_in6 *dst;
struct msghdr m;
struct cmsghdr *cm;
struct iovec iov[2];
u_char cmsgbuf[256];
 /* must be > CMSG_SPACE(sizeof(tclass)) */

 /* set the data buffer to send */
memset(m, 0, sizeof(m));
memset(iov, 0, sizeof(iov));
m.msg_name = (caddr_t)dst;
m.msg_namelen = sizeof(dst);
iov[0].iov_base = buf;
iov[0].iov_len = len;
m.msg_iov = iov;
m.msg_iovlen = 1;

 /* set ancillary data for
    the traffic class field */
memset(cmsgbuf, 0, sizeof(cmsgbuf));
cm = (struct cmsghdr *)cmsgbuf;
m.msg_control = cm;
m.msg_controllen
  = CMSG_SPACE(sizeof(tclass));
cm->cmsg_len
  = CMSG_LEN(sizeof(tclass));
cm->cmsg_level = IPPROTO_IPV6;
cm->cmsg_type = IPV6_TCLASS;
memcpy(CMSG_DATA(cm), &tclass,
       sizeof(tclass));

sendmsg(s, &m, 0);
```

If you want to put specific value to the traffic class field on multiple packets, you can use a "sticky" option:

```
u_int8_t tclass;
setsockopt(fd, IPPROTO_IPV6,
           IPV6_TCLASS, &tclass,
           sizeof(tclass));
```

15.4 Conflict resolution

There are two entities which may modify the traffic class field, in the kernel of the originating node: a kernel IPv6 code with diffserv marking enabled, and an ECN-capable TCP stack. Those entities may modify the traffic class field, even if an application tries to manipulate the value. It may present a difficult constraint to the API. For outbound traffic, even if an application specifies the value to be put into the traffic class field, in-kernel mechanism(s) may need to modify the field. The specified value may not be reflected into the packet on the wire (example: outbound processing in an ECN-capable TCP stack). For inbound traffic, even if the kernel presents the value on the field to the application, the value may not be the same as the value on the packet on the wire, due to manipulation in the kernel (example: traffic received by a diffserv egress node itself).

The following text proposes a suggested behavior. One of the goals of the suggestion is to allow applications to implement UDP ECN by themselves. The behavior may need more discussions:

Outbound traffic

If there is no conflict (for example, the TCP stack is not ECN- capable), the kernel should honor the value an application specified, and put the specified value into the traffic class field as is. If there is a conflict, the kernel should override the value specified by the application, for the part of the field (bits) the kernel is using. For example, if the kernel has an ECN- capable TCP stack but does not support diffserv, the kernel should override ECN bits only.

Inbound traffic

Kernel should present the traffic class value ap-

peared on the wire as is to applications. Note that, in some cases, the kernel may want to alter specific bits in the field, before presenting the value to the userland. For example, if the kernel implements TCP ECN and would like to make it transparent to the user programs, the kernel may want to hide ECN bits.

From diffserv and ECN protocol specifications, the traffic class field may be rewritten by intermediate routers. So even if the sender specifies a value, the value may be altered before the packet reaches the final destination.

15.5 Issues

- Revise conflict resolution rule?

15.6 Security consideration

The API could be used for attempted theft of service. An attacker may try to inject packets, with some specific value in traffic class field, into a diffserv cloud. Refer to RFC2474 [113] section 7.1 for detail. Note that the theft of diffserv service is possible even without the API.

第16章 An IPv6-to-IPv4 transport relay translator

16.1 Problem domain

When you deploy an IPv6-only network, you still want to gain access to IPv4-only network resources outside, such as IPv4-only web servers. To solve this problem, many IPv6-to-IPv4 translation technologies are proposed, mainly in the IETF ngtrans working group. The memo describes a translator based on the transport relay technique to solve the same problem.

In this memo, we call this kind of translator "TRT" (transport relay translator). A TRT system locates between IPv6-only hosts and IPv4 hosts and translates TCP,UDP/IPv6 to TCP,UDP/IPv4, vice versa.

Advantages of TRT are as follows:

- TRT is designed to require no extra modification on IPv6-only initiating hosts, nor that on IPv4-only destination hosts. Some other translation mechanisms need extra modifications on IPv6-only initiating hosts, limiting possibility of deployment.
- The IPv6-to-IPv4 header converters have to take care of path MTU and fragmentation issues. However, TRT is free from this problem.

Disadvantages of TRT are as follows:

- TRT supports bidirectional traffic only. The IPv6-to-IPv4 header converters may be able to support other cases, such as unidirectional multicast datagrams.
- TRT needs a stateful TRT system between the communicating peers, just like NAT systems. While it is possible to place multiple TRT systems in a site (see Appendix A), a transport layer connection goes through particular, a single TRT system. The TRT system thus can be considered a single point of failure, again like NAT systems. Some other mechanisms, such as SIIT [116], use stateless translator systems which can avoid a single point of failure.

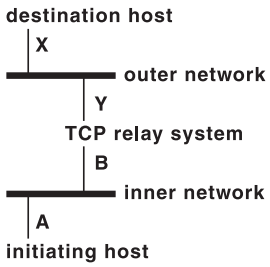
The memo assumes that traffic is initiated by an IPv6-only host destined to an IPv4-only host. The memo can be extended to handle opposite direction, if an apprpriate address mapping mechanism is introduced.

16.2 IPv4-to-IPv4 transport relay

To help understanding of the proposal in the next section, here we describe the transport relay in general. The transport relay technique itself is not new, as it has been used in many of firewall-related products.

TCP relay

TCP relay systems have been used in firewall-related products. These products are designed to achieve the follwing goals: (1) disallow forwarding of IP packets across a system, and (2) allow TCP,UDP traffic to go through the system indirectly. For example, consider a network constructed like the following diagram. "TCP relay system" in the diagram does not forward IP packet across the inner network to the outer network, vice versa. It only relays TCP traffic on a specific port, from the inner network to the outer network, vice versa. (Note: The diagram has only two subnets, one for inner and one for outer. Actually both sides can be more complex, and there can be as many subnets and routers as you wish)



When the initiating host (whose IP address is A) tries to make a TCP connection to the destination host (X), TCP packets are routed toward the TCP relay system based on routing decision. The TCP relay system receives and accepts the packets, even though the TCP relay system does not own the destination IP address (X). The TCP relay system pretends to having IP address X, and establishes TCP connection with the initiating host as X. The TCP relay system then makes a another TCP connection from Y to X, and relays traffic from A to X, and the other way around.

Thus, two TCP connections are established in the picture: from A to B (as X), and from Y to X, like below:

```
TCP/IPv4: the initiating host (A)
          --> the TCP relay system (as X)
    address on IPv4 header: A -> X
TCP/IPv4: the TCP relay system (Y)
          --> the destination host (X)
    address on IPv4 header: Y -> X
```

The TCP relay system needs to capture some of TCP packets that is not destined to its address. The way to do it is implementation dependent and outside the scope of this memo.

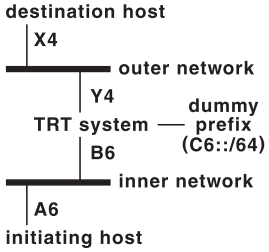
UDP relay

If you can recognize UDP inbound and outbound traffic pair in some way, UDP relay can be implemented in similar manner as TCP relay. An implementation can recognize UDP traffic pair like NAT systems does, by recording address/port pairs onto an table and managing table entries with timeouts.

16.3 IPv6-to-IPv4 transport relay translator

We propose a transport relay translator for IPv6-to-IPv4 protocol translation, TRT. In the following description, TRT for TCP is described. TRT for UDP can be implemented in similar manner.

For address mapping, we reserve an IPv6 prefix referred to by C6::/64. C6::/64 should be a part of IPv6 unicast address space assigned to the site. Routing information must be configured so that packets to C6::/64 are routed toward the TRT system. The following diagram shows the network configuration. The subnet marked as "dummy prefix" does not actually exist. Also, now we assume that the initiating host to be IPv6-only, and the destination host to be IPv4-only.



When the initiating host (whose IPv6 address is A6) wishes to make a connection to the destination host (whose IPv4 address is X4), it needs to make an TCP/IPv6 connection toward C6::X4. For example, if C4::/64 equals to fec0:0:0:1::/64, and X4 equals to 10.1.1.1, the destination address

to be used is fec0:0:0:1::10.1.1.1. The packet is routed toward the TRT system, and is captured by it. The TRT system accepts the TCP/IPv6 connection between A6 and C6::X4, and communicate with the initiating host, using TCP/IPv6. Then, the TRT system investigates the lowermost 32bit of the destination address (IPv6 address C6::X4) to get the real IPv4 destination (IPv4 address X4). It makes an TCP/IPv4 connection from Y4 to X4, and forward traffic across the two TCP connections.

There are two TCP connections. One is TCP/IPv6 and another is TCP/IPv4, in the picture: from A6 to B6 (as C6::X4), and Y4 to X4, like below:

```
TCP/IPv6: the initiating host (A6)
          --> the TRT system (as C6::X4)
    address on IPv6 header: A6 -> C6::X4
TCP/IPv4: the TRT system (Y4)
          --> the destination host (X4)
    address on IPv4 header: Y4 -> X4
```

16.4 Address mapping

As seen in the previous section, an initiating host must use a special form of IPv6 address to connect to an IPv4 destination host. The

special form can be resolved from a hostname by static address mapping table on the initiating host (like /etc/hosts in UNIX), special DNS server implementation, or modified DNS resolver implementation on initiating host.

16.5 Notes to implementers

TRT for UDP must take care of path MTU issues on the UDP/IPv6 side. This is implementation dependent and outside of the scope of this memo. A simple solution would be to always fragment packets from the TRT system to UDP/IPv6 side to IPv6 minimum MTU (1280 octets), to eliminate the need for path MTU discovery.

Though the TRT system only relays TCP,UDP traffic, it needs to check ICMPv6 packets destined to C6::X4 as well, so that it can recognize path MTU discovery messsages and other notifications between A6 and C6::X4.

When forwarding TCP traffic, a TRT system needs to handle urgent data [129] carefully.

To relay NAT-unfriendly protocols [68] a TRT system may need to modify data content.

Scalability issues must carefully be considered when you deploy TRT systems to a large IPv6 site. Scalability parameters would be (1) number of connections the operating system kernel can accept, (2) number of connections a userland process can forward (equals to number of filehandles per process), and (3) number of transport relaying processes on a TRT system. Design decision must be made to use proper number of userland processes to support proper number of connections.

To make TRT for TCP more scalable in a large site, it is possible to have multiple TRT systems in a site. This can be done by taking the following steps: (1) configure multiple TRT systems, (2) configure different dummy prefix to them, (3) and let the initiating host pick a dummy prefix randomly for load-balancing. (3) can be implemented as follows; If you install special DNS server to the site, you may (3a) configure DNS servers differently to return different dummy prefixes and tell initiating hosts of different DNS servers. Or you can (3b) let DNS server pick a dummy prefix randomly for load-balancing. The load- balancing is possible because you will not be changing destination address (hence the TRT system), once a TCP connection is established.

For address mapping, the authors recommend use of a special DNS server for large-scale installation, and static mapping for small-scale installation. It is not always possible to have special resolver on the initiating host, and assuming it would cause deployment problems.

16.6 Security considerations

Malicious party may try to use TRT systems for anonymizing the source IP address of traffic to IPv4 destinations. TRT systems should implement some sorts of access control to avoid such improper usage.

A careless TRT implementation may be subject to buffer overflow attack, but this kind of issue is implementation dependent and outside the scope of this memo.

A transport relay system intercepts TCP connection between two nodes. This may not be a legitimate behavior for an IP node. The draft does not try to claim it to be legitimate.