第 11 部

トンネリング技術

第 1 章

導入

インターネットワーキング技術の随所でトンネリングの利用が見られる。しかしトンネリング技術は裏方の汚い技術とされ例外的に扱われいる場合が多く、表舞台に登場することは少ない。ddt WG はトンネリング技術に光を当て、トンネリング技術をインターネットの枠組にすっきりと収めるための研究、及びトンネリングの実装手法、応用方法の研究を行なう。

本章ではその導入として、トンネリング技術の紹介、及び現状の問題点とその解決策案について述べる。

1.1 トンネリングとは

トンネリングとは、ネットワーク層のプロトコル x (例えば IP) を伝達するために、他のネットワーク層以上のプロトコル y (例えば X.25) をデータリンク層のプロトコルとみなして利用する技術である。このとき、プロトコル x のデータグラムは、プロトコル y のデータとして扱われる。この状態を $x\ over\ y$ と呼ぶ。

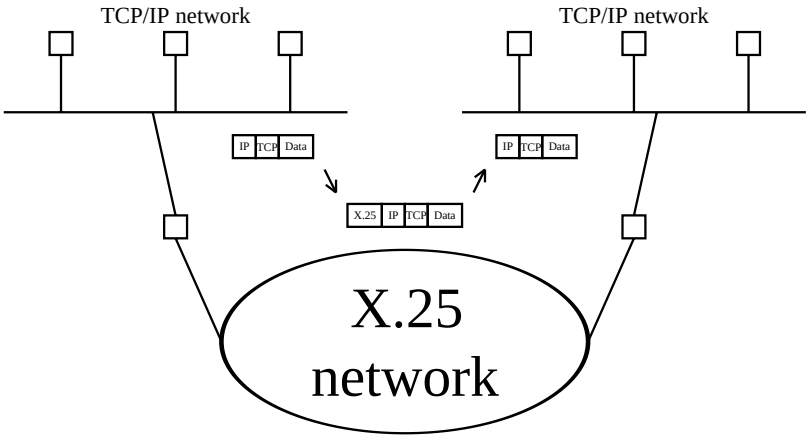

図 1.1: TCP/IP over X.25

歴史的には、組織間を IP で接続する際に、既付設の X.25 等のネットワークをデータ

リンクとみなして IP の送受信に用いるという形で利用されてきた (図 1.1 参照)。IP による広域ネットワークが発達した現在では、IPX、SNA、OSI などのネットワーク・プロトコルを IP ネットワークを用いて送受信するために利用されている (図 1.2 参照)

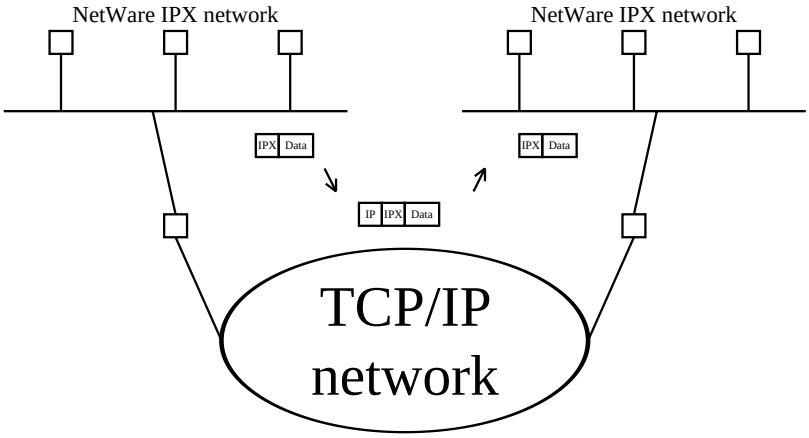

図 1.2: Netware IPX over TCP/IP

ddt WG の研究対象の中心は、トンネルを構成するプロトコルとトンネルで運ばれるプロトコルが同じ場合、特に IP over IP の場合である。IP over IP 利用の主な例として、マルチキャストの経路制御、移動ホストなどがあげられる。

1.2 ddt の基本アイデア

トンネリング技術をインターネットの表舞台に出すための基本アイデアは、トンネリング機能を仮想ネットワークのインタフェースの形で提供するというものである。

トンネルには入口と出口が存在することを考えると、このインタフェースはトンネリングを用いて仮想的に 2 点間リンクを提供していると言うこともできる。さらに 2 点間に限らず複数のホストが接続可能な仮想的なネットワークを提供できるであろう。すなわち、**トンネリング**を用いて構築された**仮想ネットワーク**を、**仮想インタフェース**を通して提供しているわけである。このアイデアを **ddt** と呼ぶ。

1.3 ddt と階層モデル

プロトコル階層モデルにおける仮想インタフェース、仮想ネットワーク、トンネリングの関係を見てみる。トンネルで運ばれるプロトコルを中心に階層構造で示すと表 1.1 になっている。またトンネルを構成するプロトコルを中心に見ると、表 1.2 になっている。

このように注目するプロトコルによって異なる階層構造の図が書ける。そのためトンネルを構成するプロトコルとトンネルで運ばれるプロトコルが同じプロトコル・ファミリ

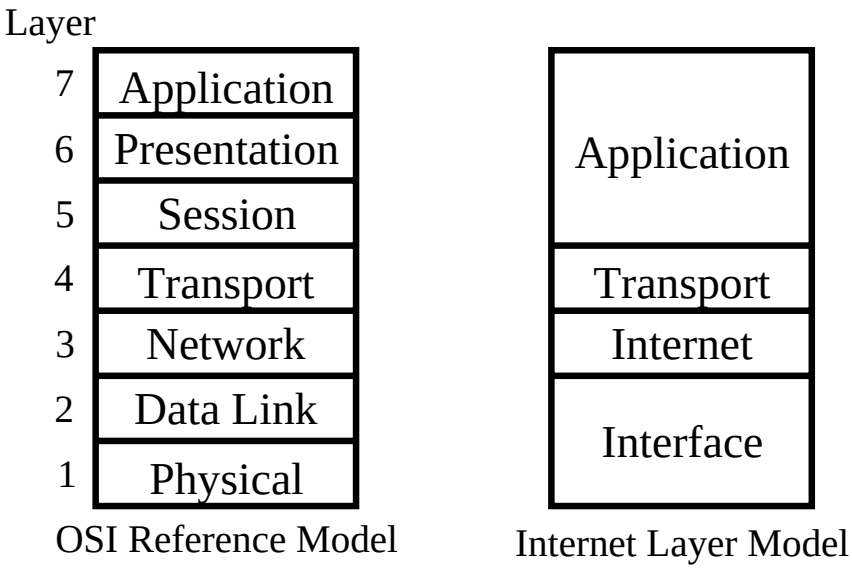

図 1.3: プロトコル階層モデル

仮想インタフェース	3 層と 2 層のインタフェース
仮想ネットワーク	2 層上位
トンネリング	2 層下位 あるいは 1 層

表 1.1: トンネルで運ばれるプロトコルを中心に見た階層構造

仮想インタフェース	7 層より上にあるアプリケーション
仮想ネットワーク	4 層 〜 7 層
トンネリング	3 層 (あるいはさらに上位プロトコル)

表 1.2: トンネルを構成するプロトコルを中心に見た階層構造

に属する場合、例えば IP over IP の場合は複雑である。経路制御の結果、IP データグラムは一度 2 層まで降りてカプセル化されるが、実は 3 層のプロトコルである IP の上にカプセル化される。そのため元のデータグラムはキャリア・プロトコルのアプリケーション層プロトコルとして扱われた状態で再び経路制御部に入る。このとき Layer Violation が起きていると言える。

Layer Violation は経路制御 (Forwarding) を複雑にする。というのは、そのデータグラムが再び同じ仮想インタフェースに投げられると無限ループを起こしてしまうからである。詳細は 2.4 節参照。

1.4 インターネット・アーキテクチャにおける ddt

ここで ddt のアイデアをインターネット・アーキテクチャ・モデルに適合させることを試みる。まずインターネット・アーキテクチャ・モデルを振り返ってみる (図 1.4 参照)。

1. インターネットはゲートウェイ、ホスト、ネットワーク、インタフェースから成る。
2. すべてのホストはネットワークとの接続のためにインタフェースを持つ。複数持っていてもかまわないがパケット転送機能はない。
3. すべてのゲートウェイは複数のネットワークに対してインタフェースを持ち、パケット転送機能を提供している。
4. すべてのネットワークはネットワーク識別子を持つ。
5. すべてのインタフェースはネットワーク識別子＋ホスト識別子という形式のアドレスを持つ。

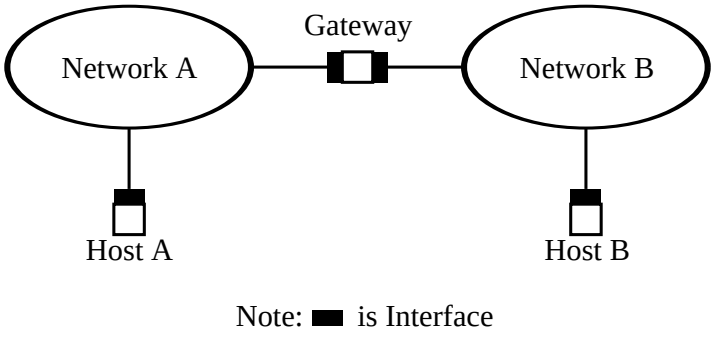

図 1.4: インターネット・アーキテクチャ

ddt をこのモデルに適合させる際考慮すべきは、仮想インタフェースと仮想ネットワークである。トンネリング自体は考慮する必要がない点に注意して欲しい。実際にモデル

に適合させるには、仮想ネットワークにネットワーク識別子を付与し、仮想インタフェースにアドレスを付与すればよい[1](図 1.5 参照)。これにより ddt はインターネット・アーキテクチャのモデルの中で通常のネットワークとなんら変わりなく存在でき、既存の理論を適用できるようになった。

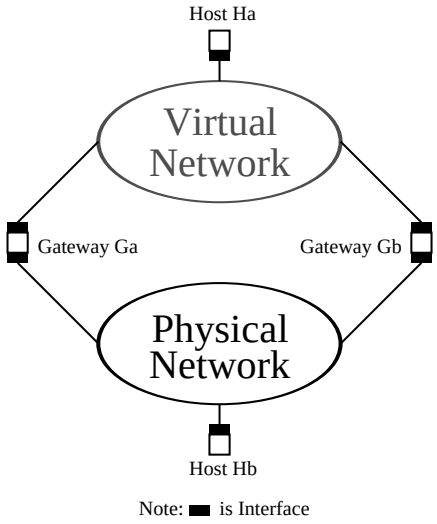

図 1.5: インターネット・アーキテクチャと ddt

ddt の仮想ネットワークは実際には物理的なネットワークを持たないため、その可用性は他のネットワークに強く依存している。そのため経路情報交換や経路情報計算においては注意が必要である。詳細は 2.5節参照。

1.5　仮想アドレスと物理アドレス

トンネリングによる仮想ネットワークは専用の物理ネットワークを持たないため、実際の通信には物理媒体を基盤とするネットワーク (以後物理ネットワークと呼ぶ) を利用する。したがって、物理ネットワークのアドレスがトンネル端点を特定するために重要な意味を持つ。

ddt を利用するホストが複数の物理ネットワークに接続されている可能性がある。この場合、トンネルの端点を特定するためにいずれを使うことも可能であるが、トンネルの識別を容易にするためにそのうちからひとつだけ選んで用いることとする。

このように仮想インタフェースは、表側に仮想ネットワークのアドレスを、裏側に物理ネットワークのアドレスを持つことになる。このふたつのアドレスは異なるプロトコル

[1]ddt では簡単のため、モデル上は numbered link (network) を考察対象とする。

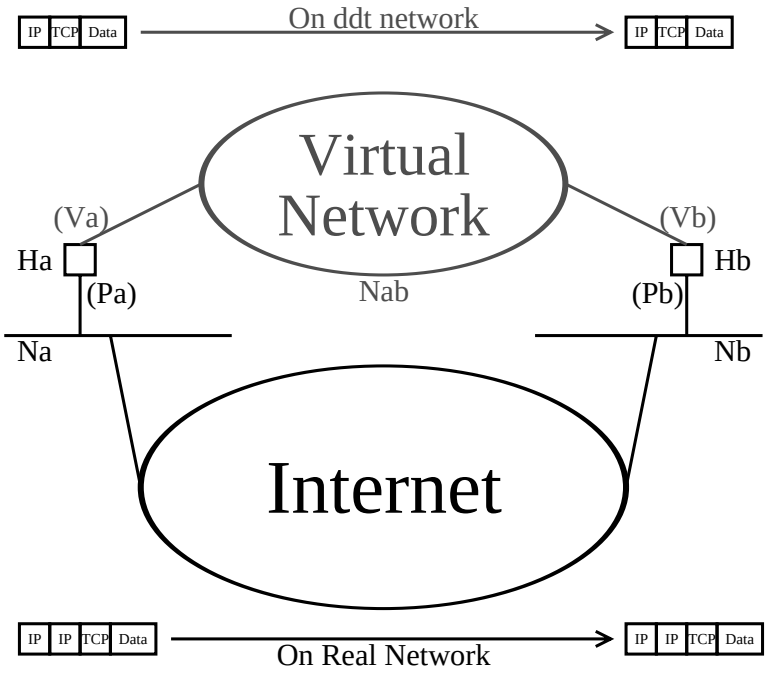

図 1.6: ddt IP over IP の場合のトンネリング

体系に属するものであってもかまわない。

図 1.6 に、ddt による仮想ネットワークが 2 点間リンクである場合の Host H_a 発 H_b 宛 ddt link 経由の TCP パケットの様子を示す。ddt link 上のパケットの IP ヘッダは始点アドレスが V_a、終点アドレスが V_b であり、データ部は TCP データグラムである (図上側)。実際にはパケットは物理的に存在するネットワークを通っている。物理ネットワーク上のパケットの IP ヘッダは始点アドレスが P_a、終点アドレスが P_b であり、データ部は ddt link を通っているはずのパケットである (図下側)。

1.6 まとめ

本章では、トンネリング技術をインターネット・アーキテクチャにすっきりと収めるための技術である ddt を紹介した。ddt をまとめると次のようになる。

1. トンネリングによる仮想ネットワークを仮想インタフェースの形で提供する。

2. 仮想ネットワークにネットワーク識別子を、仮想インタフェースにアドレスを付与する。

また、次の問題が存在する。

1. 各ゲートウェイでの経路制御が難しい
2. インターネット全体での経路情報の扱いが難しい

次章から、ddt の枠組、実装、応用例、実験例、今後の研究計画について述べる。

第 2 章

枠組

ddt とは、トンネリングによる仮想ネットワークを仮想インタフェースの形で提供する技術であった。また、ddt 利用にあったって 2 つの解決すべき問題が存在した。したがって、ddt 技術を分解すると「トンネリング」、「仮想ネットワーク」、「仮想インタフェース」の 3 つの要素技術と、「経路制御」、「経路情報交換」の 2 つの基本技術からなると言える。

本章ではこれらの ddt を構成する個々の技術内容を ddt とは独立に見ていく。その後改めて ddt の視点からそれらを検討し、ddt の全体像と本質を考える。

2.1 要素技術 1 — Virtual Interface

インタフェースは、3 層と 2 層の間を取り持つものである。ゲートウェイの経路制御部から見ると、経路制御の結果データグラムを送り出す先である。物理インタフェースの場合、物理的に存在するネットワークへの出入口であり、純粋に物理ネットワークへのアクセス手段を提供するものであった。仮想インタフェースの場合、次の機能が考えられる。

1. トンネリングによる仮想ネットワークの提供
2. フックとして様々な付加機能の提供
3. 経路制御部の簡便のための機能の提供

以下、それぞれについて見ていく。

2.1.1 トンネリングによる仮想ネットワークの提供

1.2 節で紹介したように、ddt の基本アイデアは、トンネリングによる仮想ネットワークを仮想インタフェースの形で提供することであった。これはまさにそれである。

2.1.2　フックとして様々な付加機能の提供

仮想インタフェースはトンネリング機能に限らず様々な付加機能を提供しうる。そこでフックとして付加機能を提供することだけを目的とした仮想インタフェースを考えてみる。ここではこれをフック・インタフェースと呼ぶことにする。フック・インタフェースにも仮想ネットワークを提供しているとみなせるものがある。

フック・インタフェースは他のインタフェースに張りついてはじめて意味を持つ。このとき、張りついたフック・インタフェースを上位インタフェース、張りつかれたインタフェースを下位インタフェースと呼ぶことにする。上位、下位と呼んではいるが、経路制御部からはどちらも普通のインタフェースとして見えるため階層関係はない。つまり下位インタフェースがまたフック・インタフェースであってもかまわない。その場合下位インタフェースのさらに下位には、ネットワークを持つインタフェースが存在する必要がある。このようにフック・インタフェースは積み重ね可能なので、スタッカブル・インタフェースと呼ぶこともできる。必要に応じてフック・インタフェースを積み重ねることにより、様々な性質のインタフェースを利用できる[1]。

フック・インタフェースによっては下位インタフェースを経路制御部から隠したい場合があるだろう。その場合、インタフェースに hidden 属性を導入することで解決可能と考える。

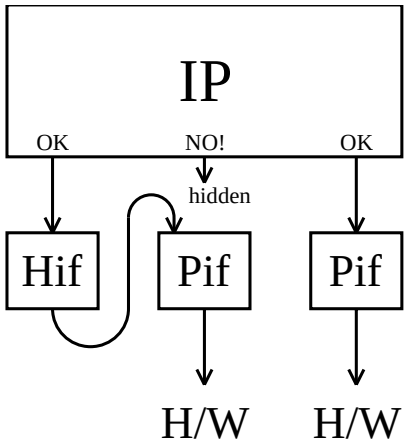

図 2.1: フック・インタフェース

以下に、フック・インタフェースによる付加機能の例を挙げる。

- ひとつのネットワークを複数のインタフェースで共有
  例えば 1 本の Ethernet に論理的に複数のネットワークが同居しているとき、ゲートウェイはすべての論理ネットワークに接続する必要がある。これは、ひとつの論

[1] 概念的には STREAMS と相性が良いかもしれない

理ネットワークにひとつのフック・インタフェースを割りつけることにより実現可能である。

- 細い回線をたばねる
  組織間でバンド幅の細い回線が複数利用可能なとき、トラフィックを各回線に均等割りして有効利用したい場合がある。均等割りの実現を経路制御部で行なう方法とインタフェース内で行なう方法が考えられる。後者の場合、複数の物理インタフェースをたばねて負荷分散を行なうフック・インタフェースにより実現可能である[2]。

- 監視
  ネットワーク管理上、トラフィックの監視が必要な場合がある。しかしトラフィックを監視するにはインタフェースに特別な機能を付加する必要がある場合があるが、そのための改造は容易ではない。ところがこの機能を持つフック・インタフェースがあれば、インタフェースの改造なしに目的を達成できる。

- 流量制限
  仮想インタフェースを通過するデータグラム量を制限する。バースト性のあるトラフィックでもこのインタフェースを通せば流量の平坦化が可能となる。

- パケット・フィルタリング
  ネットワークに送出できるパケットの種類を制限し、それ以外のパケットをエラーにしてしまうことが実現できる。
  これは経路制御技術で考えるべきことかもしえない。

- 途中ルーターによるソース・ルーティング
  あるフック・インターフェースを通過したパケットに、LSRR などのソース・ルーティングをおこなうオプションなどを強制的につけることによって、ポリシー・ルーティングを行なうことができる。

- 移動ホスト支援機能
  移動ホストは、移動のたびに利用可能な物理インタフェースが変わる可能性がある。移動ホスト機能をフック・インターフェースとして提供することにより、利用可能な物理ネットワークに依存せず移動することが可能となる。

2.1.3 経路制御部の簡便のための機能の提供

経路制御の仕事はパケットの宛先等の情報から転送先を決定し送出することであり、経路制御部の動作指定はすべてその形式で記述されなければならない。そのため特殊な動作をさせたい場合は、特別なインタフェースを用意し、そのインタフェースの中で処理をする形態が取られることが多い。以下、そのようなインタフェースの例をあげる。この場合も、仮想ネットワークを提供しているとみなせるものがある。

[2] 文献 [99] の VIF に似ている。

- ループ・バック
  入力パケットをすべて、そのまま経路制御部に返すインタフェース。

- エラー生成
  入ってくるパケットすべてに対し、Network Unreach などのエラーを生成する。

- isolate[100]
  このインタフェースから来たパケットは forward が処理禁止される。仮想インタフェースとしてではなく、インタフェースの属性として扱うことも可能。

2.2　要素技術 2 — Virtual Network

様々な性質の仮想ネットワーク (2 層相当) を提供する部分である。トンネリングやカプセル化により仮想ネットワークを構築するアイデア自体はすでにいくつかの文献に見られる [100][101]。それらは必要なネットワーク・トポロジーを得る手段として提案されている。仮想ネットワーク構築の他の目的として、特別の性質や機能の提供がある。

2.2.1　トポロジーの補助としての仮想ネットワーク

以下の仮想ネットワークにより、様々なトポロジーが実現できる。

- 2 点間リンク
  ふたつのゲートウェイを仮想リンクで直接接続する。ゲートウェイ間に安くリンクを張りたい場面で利用される。最も単純な形態の仮想ネットワークであると考えられる。

- 広域 Ethernet 的ネットワーク
  複数のホストが接続できるネットワーク。複数の物理ネットワークにまたがるホストやゲートウェイが、同じネットワークに接続しているように見せることができる。

2.2.2　特殊機能を実現する仮想ネットワーク

アプリケーション層で行なわれていた非本質的な共通処理を、下位層の機能として実現し、全アプリケーションでその機能を簡単に利用できるようにする技術であると言える。そういう意味では、ネットワーク層に限らず他の層、例えばトランスポート層で実現すべきものもあるかもしれない。

様々な性質の仮想ネットワークをアプリケーションが選択するために、TOS 指定の経路制御が重要になる。

仮想ネットワークが提供する主な性質として以下のものが挙げられる。

- 移動ホストの支援
  ホストがネットワーク間を移動しても、接続したままでい続けられるネットワーク。

- Secure Network
  通過中のデータグラムを暗号化する。盗聴防止に利用できる。現在、セキュリティを高めるためにアプリケーション・プロトコルごとに暗号化方法を規定し、アプリケーションごとに実装するアプローチを取っている。ddt ではネットワーク自体を暗号化するアプローチを取る。そのため個々のアプリケーションを改造する必要はなくなる。経路制御で確実に Secure Network を通すために、TOS に secure bit が必要になるであろう。
  これと似たアプローチとして、トランスポート層プロトコルでの暗号化も考えられる。

- 圧縮
  通過中のデータグラムを圧縮する。物理資源の有効利用ができる。

- バンド幅制限
  通過するデータ量を規制する。例えば telnet トラフィックを ftp のような大きなバンド幅を必要とするアプリケーションから守りたい場合、ftp のトラフィックをバンド幅制限機能のある仮想ネットワークを通すことで実現できるであろう。

2.3 要素技術 3 — Tunneling

トンネリングは仮想ネットワークを実現するものである。
ddt で扱うトンネルとは、次の性質を持つものとする。

1. 入口と出口が存在する。
2. トンネル通過中のパケットには出口・入口を識別する情報が書いてある。
3. その経路のための専用の物理ネットワークを持たない。
   もしくは、物理ネットワークとは完全対応はしていない。

したがって ddt で用いるトンネリング方式には以下の性質が要求される。

1. パケットを見ただけで、自分がトンネル端点としてふるまうべきかどうかわかる。
2. パケットを見ただけで、自分の持つトンネルリンクのうちのどのトンネル端点宛であるかわかる。
3. トンネル入口で加工可能な方式である。出口で元のパケットが取り出せる。

トンネルより緩やかな定義が必要な場合がある。例えばあるホストにとっては物理ネットワークであるものを、別のホストにとっては仮想ネットワークであるようにしたい場合である。そこでトンネルを拡張して次のように定義し、**クラインの壷**と呼ぶことにする。

1. 入口または出口は存在する。明確な出口または入口は存在しないこともある。

2. パケットがクラインの壷の中にあるのか外にあるのか区別できない。

以下本節では、**トンネル**を実現するプロトコルについて考察する。

2.3.1 IP over IP, IP over TCP/IP

トンネルを構成するプロトコルとトンネルで運ばれるプロトコルが同じプロトコル・ファミリに属すること、及び TCP/IP インターネットは世界的規模で利用可能であることから、ddt WG の一番の興味対象である。

トンネルを実現するプロトコルとして次のものが挙げられる。

- IP in IP (ipproto=4)
  キャリアである IP パケットのデータとして、トンネリングされる IP パケットをそのまま載せたもの。

- IP within IP (ipproto=94)
  IP within IP[102] は、Columbia 方式の移動ホスト・サポートのためのプロトコルである。2 種類のフォーマットがある。

  1. 単純カプセル化
     キャリアである IP パケットのデータとして、トンネリングされる IP パケットをそのまま載せたもの。データの最初の 4 ビット[3]の値が 4 であることによって識別する。
  2. 拡張カプセル化
     キャリアである IP パケットのデータに、トンネリングされる IP パケット以外に付加情報を載せられる。フォーマットを図 2.2 に示す。パケットタイプは 0x00 〜 0x3F の範囲になければならない。これは単純カプセル化方式と区別するためである。これにより現在及び将来の IP のバージョンすべてに対応できる。

TYPE	HDLEN	CHECKSUM
more header, if any		
encapsulated packet(s)		

図 2.2: Extended IPIP Packet

- SLIP or PPP over TCP/IP
  TCP/IP による信頼性ある 2 点間リンクの上に、SLIP[7] や CSLIP または PPP[103] などを用いてデータグラムを転送する方法である。PPP を用いれば、IP 以外のプロトコルをトンネルで運ぶことができる。

[3]IP ヘッダの最初の 4 ビットには IP のバージョンが書いてある。現在のバージョンは 4 である [16]。

- PPP over gzip over TCP/IP
  TCP/IP による信頼性ある 2 点間リンクの両端に gzip や compress などを置いてトンネル上のデータを圧縮し、さらにその上を SLIP や CSLIP または PPP などを用いてデータグラムを転送する方法である。物理資源を有効利用することができる。

2.3.2 IP option の応用

IP には option 機能がある。現在のところ、IP option には ddt で使えるトンネリング機能は標準では用意されていない。仮に IP option を利用してトンネリングを実現できた場合の利点・欠点を挙げる。

- 利点
  1. トンネル経路上で発生した ICMP の情報を実際の始点ホストに伝えるのが楽。
  2. IP パケットへの付加バイト数が小さい。
  3. TTL が実際のインターネットに従って減じられる。
- 欠点
  1. IP ヘッダサイズの上限 64 octets を越えた option 追加はできない。
  2. IP 以外のプロトコルでは使えない。
  3. IP option の処理は重いかもしれない。
  4. 通常のリンクよりも TTL の減少が激しい。

IP パケットが最終目的地到着前に、指定したゲートウェイを順番に通るようにするための機能として IP option の SSRR (strict soruce and record route) LSRR (loose source and record route) [16] がある。SSRR は経路をひとつ残らず厳密に指定するが、LSRR は各指定アドレス間はどこを経由しても良い。

LSRR のフォーマットを図 2.3 に示す。LSRR には通過すべきゲートウェイをその順番に書く。現在目指しているゲートウェイが IP ヘッダの終点アドレスに書かれる。つまり、LSRR に書かれたゲートウェイを通過するたびに終点アドレスが変化する。

LSRR オプションを持つパケットは、まず IP ヘッダの終点アドレスのゲートウェイに運ばれる。IP ヘッダ終点アドレスのゲートウェイに到着すると、PTR が示す IP アドレスが IP ヘッダの次の終点アドレスになる。そして、PTR の場所にそのゲートウェイの IP アドレスが入り、PTR をひとつ進める。もし PTR が LSRR option の一番最後をさしていたなら、そのゲートウェイが終点である。このようにしてゲートウェイをたどって行く。

現在インターネット上で実験的に広く使われているマルチキャスト通信の実装では、LSRR option のフォーマットを利用してトンネリングを行なっている [32][4]。マルチキャ

[4]ただし、最新の実装では、IP in IP (ipproto=4) を用いたトンネリングをサポートし始めた

CODE	LEN	PTR
IP ADDRESS OF FIRST HOP		
IP ADDRESS OF SECOND HOP		
...		

図 2.3: LSRR IP option のフォーマット

IP src addr	tunnel src gateway addr
IP dst addr	tunnel dst gateway addr
LSRR ptr	points to LSRR addr2
LSRR addr1	actual src host
LSRR addr2	multicast addr

図 2.4: LSRR を用いたマルチキャスト・トンネルにおける IP ヘッダと LSRR option の状態

スト経路制御でトンネリングを用いる動機は、マルチキャスト経路制御のできないルータがまだ多いことにある。図 2.4 に、マルチキャスト・トンネリング時の IP ヘッダ及び LSRR option の状態を示す。これは通常の LSRR と比べて次の特徴を持つ。

1. IP ヘッダの始点アドレスがトンネルの始点になっている。
2. LSRR ptr がマルチキャスト・アドレスを指している[5]。
3. LSRR ptr のひとつ手前に実際の始点アドレスがある。

これはトンネルの要件を満たしている。

2.3.3　XX over IP, IP over XX

IP 以外のプロトコルを IP で支援したり、その逆を行なったりする。IP をキャリアにする場合の主なカプセル化プロトコルを表 2.1 にあげる [105]。

表中の XNS over IP は 4.3BSD の XNS IDP の実装に含まれている。

Ethernet within IP を用いると、Ethernet を広域で利用することができるかもしれない。

[5] RFC1054[104] 6.2 には次のようにある。
A host group address should not be placed in the source address field or anywhere in a source routing option of an outgoing IP datagram.

番号	キーワード	プロトコル
22	XNS-IDP	XEROX NS IDP
80	ISO-IP	ISO Internet Protocol
97	ETHERIP	Ethernet within IP
98	ENCAP	Encapsulation Header[101]

表 2.1: IP における主なカプセル化プロトコル

2.3.4 IP over UUCP, IP over MAIL

トンネルを構成するプロトコルとして UUCP や MAIL を利用する方法も考えらる。このトンネルが提供する仮想ネットワークは、通常のネットワークとは次の点で大きく異なる。

- RTT (Round Trip Time) が非常に大きい
  通常は高々数秒の RTT しか想定していない。
- リンク上に同時に存在しうるデータ量が大きい
  UUCP や MAIL のキューが許す限りの量が入る。

従って、これを利用するには新しいトランスポート・プロトコルと新しいアプリケーション・プロトコルが必要であろう。これはバッチ指向のプロトコルになるかもしれない。また、間欠リンクを通して経路情報を交換するための新しい経路情報交換プロトコルも必要になるであろう。

インターネットに接続していないゲートウェイにおけるメイル転送、ニュース配布などのサービスはアプリケーション・レベルで行なっている。ところがこれを利用すると、転送・配送機構をネットワーク層で統一的に扱うことができるため基盤となる UUCP や MAIL の機能に依存することがなくなる。

しかしながらゲートウェイ機能をアプリケーション・レベルで行なうのが良いか、ネットワーク層で行なうのが良いかは意見が分かれるところである。

2.3.5 Layer Violation Model

以上、トンネリング実現プロトコルを見てきた。

IP over IP のように、トンネルを構成するプロトコルとトンネルで運ばれるプロトコルが同じプロトコル・ファミリに属する場合、Layer Violation が発生する。Layer Violation をうまくとりこんだ新たなプロトコル階層モデルが必要である。

例えば次のように、2 層〜4 層 (あるいはもっと上) のプロトコルは全部平等であるとみなすモデルも考えられる。

- セッション層からは 2 層〜 4 層のプロトコルが平等に見える
- アプリケーション、あるいはシステムコンフィギュレーションでほんとうに必要なプロトコルスタックを縫い合わせる。
- セッション層の下に直接置けるプロトコルは制限があるかもしれない
- 物理層の直接上のプロトコルは指定されている

モデルについてはまだまだ検討が必要である。

2.4　基本技術 1 — Packet Forwarding Technology

1.2 節にも書いたように、ddt を利用した場合経路制御が複雑になる。IP over IP の場合を例として、実際どのような問題が起きるかを示す。

2.4.1　単純な例

まずうまくいく例を考える。図 2.5 における Host A から Host B への経路制御を考える。Host A における経路制御表は次のようになっているとする。

宛先ネットワーク	ゲートウェイ	インタフェース
190.5	NextHop	イーサネット
190.7	190.7.1.2	ddt

このときの経路制御は次のようになる。

- 190.5.1.1 宛
  190.5 宛のパケットなので、物理ネットワークであるイーサネットを経由して直接 NextHop ゲートウェイに送られる。
- 190.7.1.2 宛
  1. 190.7 宛のパケットなので、まず ddt に送られる。
  2. ddt 内でパケットは 190.5.1.1 宛の IP に包まれる。
  3. 190.5.1.1 は 190.5 のネットワークに属するので NextHop に送られる。

2.4.2　相手ネットワークの他ホストとの通信

次に、図 2.5 における Host B と同じネットワークにつながる Host C(190.5.1.2) との ddt 経由の通信を考える。このとき Host A の経路制御表は次のようになっているとする。

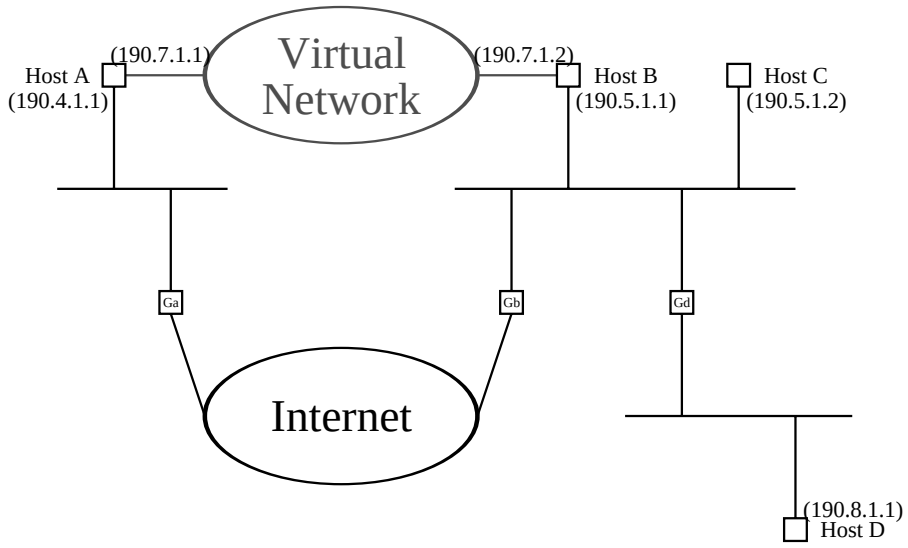

図 2.5: ddt における経路制御

宛先ネットワーク	ゲートウェイ	インタフェース
190.5	190.7.1.2	ddt
190.7	190.7.1.2	ddt

このときの 190.5.1.2 宛の経路制御は次のようになる。

- 190.5.1.2 宛

1. 190.5 宛のパケットなので、まず ddt に送られる。
2. ddt 内でパケットは 190.5.1.1 宛の IP に包まれる。
3. 190.5.1.1 は 190.5 のネットワークに属するので ddt に送られる。
4. ddt 内でパケットは 190.5.1.1 宛の IP に包まれる。
5. 190.5.1.1 は 190.5 のネットワークに属するので ddt に送られる。

無限ループに陥ってしまった。

2.4.3 相手ネットワークの向うにあるネットワークとの通信

今度は、図 2.5 のように Host B のネットワークのさらに奥のネットワークにある Host D(190.8.1.1) との ddt 経由の通信を考える。このとき Host A の経路制御表は次のようになっているとする。

宛先ネットワーク	ゲートウェイ	インタフェース
190.5	NextHop	イーサネット
190.7	190.7.1.2	ddt
190.8	190.7.1.2	ddt

このときの 190.8.1.1 宛の経路制御は次のようになる。

- 190.8.1.1 宛

  1. 190.8 宛のパケットなので、まず ddt に送られる。
  2. ddt 内でパケットは 190.5.1.1 宛の IP に包まれる。
  3. 190.5.1.1 は 190.5 のネットワークに属するので NextHop に送られる。

この場合は目的どうり発信できた。

2.4.4　3 つの問題のまとめ

以上の問題をプロトコルの階層構造的に考えてみる (図 2.6 参照)。

1. アプリケーションで生成されたデータは、OS のネットワーク処理部に渡され、最終的に IP モジュールの経路制御部に渡される。
2. IP の経路制御部は終点アドレスから転送先を決定し、インタフェースとして ddt の仮想インタフェースを選択する。
3. ddt の仮想インタフェースでカプセル化されたデータは IP モジュールに戻される。
4. カプセルによる新しい終点アドレスに合わせて IP の経路制御部は転送先を決定する。

この経路制御の結果が前のものと同じであれば、ループが起きることになる。複数の ddt の仮想インタフェースがある場合にも、複数の ddt をまたがる無限ループが起こりうる。

2.4.5　経路制御問題解決に向けて

このように、ddt 利用時に期待通りの経路制御をさせるにはなんらかの工夫が必要である。解決案がいくつかある。これらは Layer Violation のモデル化と密接に絡むため、同時に考える必要がある。

1. 複数の経路制御表を持つ
   トンネリングの結果同じネットワーク層に戻って来るとは考えない。プロトコル的には同じでも、異なる色のついたパケットと考えてその色専用の経路制御部に処理させる。当然、色ごとに異なる経路制御表を用意する必要がある。

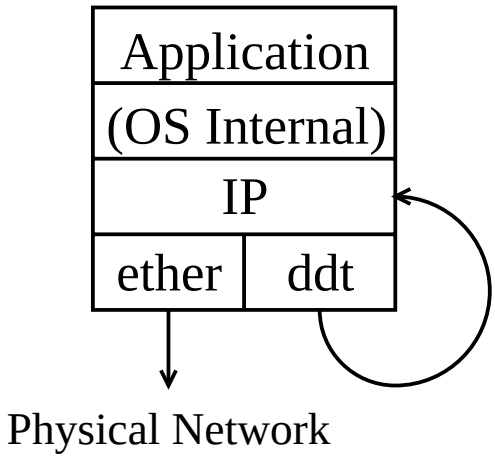

図 2.6: 経路制御における Layer Violation

2. 矛盾の起きない経路制御表を作成する
   トンネリングの結果、同じ経路制御部に返って来ると考える。理論的に無限ループの起きない条件を導き、それを経路制御表作成時の制約条件とする。
   パケットに色をつける必要がないため、アプリケーション層にまで駈け登る ddt をサポートしやすい。

3. 経路制御表のエントリを増やして対応する

4. 合成関数的に経路を計算する
   各ループを経路制御の関数とみなし、その合成関数で最終的な経路を決定する。この方法だと無限ループの検出もできる。アプリケーション層にまで駈け登る ddt がサポートできるかどうかが問題。

5. プログラミング言語の意味論の手法を応用
   スコットのプログラム意味論のようにリカーシブなプログラムの意味を厳密に定める方法がある。各ループをプロトコルからプロトコルへの写像と見て、その浮動点を求めることにより最終的な転送先を決定する。利点としては、ネットワークが破壊される場合も含めて議論できる可能性があること。

6. 他のゲートウェイの経路制御に依存する
   自身の経路制御部から確実に抜け出せるアドレスを持つゲートウェイに転送し、到達性をそのゲートウェイに委ねる。

2.4.6　その他の問題

ddt は様々な性質の仮想ネットワークを提供するため、アプリケーションから明示的に経路として指定したい場合がある。その方法として TOS (Type Of Service) による間接的方法が考えられる。ddt が一般的になると TOS を考慮した経路制御も必要になるであろう。

2.5　基本技術 2 — Routing Control Technology

ここではインターネット全体でのパケット経路制御を考える。

通常のネットワークと、ddt によるネットワークの違いは、物理媒体を基盤とするネットワークを持っているかどうかにある。そのため次の点に気を配った経路制御を行なう必要がある。

1. 仮想ネットワークは物理ネットワークの可用性に依存している
   物理ネットワークがダウンした時、仮想ネットワークも同時に利用不可能になる可能性がある。それを把握せず代替経路として仮想ネットワークを利用すると無意味なトラフィックを発生させる原因になりかねない。

2. 無意味なトンネリングによるネットワーク資源の無駄使いを避けたい
   無意味な遠回りや無意味なカプセル化によるネットワーク資源の無駄使いを避けたい。物理ネットワークと仮想ネットワークでの metric 調整をきちんと行なう必要がある。

3. ddt の相手の物理インタフェースへの経路を絶たれてはならない
   トンネリングが機能しなくなってしまう。

これらを実現するための経路制御の制約条件を理論的に導きたい。その結果、既存の経路情報プロトコルの拡張が必要になるかもしれない。

この技術が確立するまでは、不用意な ddt 経路情報の広告は避けるべきであろう。

2.6　まとめ

以上、ddt を構成する個々の技術内容や問題点を ddt とは独立に見てきた。ここでそれらを ddt を構成する技術として見直してみる。

仮想インタフェースはそれ自身パワーある概念であるが、ddt においては仮想ネットワークを経路制御部に提供するための手段にすぎない。仮想インタフェースの持つパワーを活かして、仮想ネットワークの機能を引き出す実装が望まれる。

仮想ネットワーク利用こそが ddt 構築の目的である。つまり、仮想ネットワーク構築のためにトンネリングを、経路制御部に提供するために仮想インタフェースを利用していると言える。

トンネリングは仮想ネットワークを実現する手段にすぎない。しかし非常に強力な実現手段である。ところがトンネリングの利用にあたっては経路制御・経路情報管理に未解決の大きな問題がある。ddt を実用化する上で重要な技術課題となる。

まとめると次のようになる。ddt 構築の目的は仮想ネットワーク利用である。その実現手段として仮想インタフェースとトンネリングを用いる。トンネリング利用上の大きな課題として経路制御・経路情報管理がある。

以上の考察から、ddt の本質は様々な性質のネットワークを自由なトポロジーで構築できることにあると考える。つまり、必要なネットワーク・トポロジーを実現したり、アプリケーション層以上で実現していた機能を下位層に落すことができたりするところに ddt の本質であると考える。

第 3 章

実装手法

様々な性質の仮想ネットワークの実現方式や、様々なプロトコルでトンネリングを実現するための仮想インタフェースの実装方式について考える。まず、ネットワークの実装例として 4.3BSD Tahoe について述べ、次に ddt の各種実装手法について述べる。

3.1 4.3BSD tahoe におけるネットワークの実装

4.3BSD tahoe におけるネットワーク実装は、図 3.1 に示す構造になっている [106]。

カーネル入口ではユーザプロセスからのシステムコールを適切な関数に振り分けている。ソケット関連のシステムコールではソケット層の関数が呼び出される。

ソケット層ではソケットのセマンティクスを提供している。これは、OSI 7 層モデル (図 1.3 参照) におけるセッション層の一部にあたる。ユーザ・アプリケーションからは、すべてのプロトコルをソケットを通してアクセスする。ソケット層はプロトコル処理のためにプロトコル層を呼び出す。

プロトコル層では、OSI 7 層モデルにおけるネットワーク層・トランスポート層のプロトコルの処理が行なわれる。接続状態を持つプロトコル処理はソケット層と連係して行なわれる。データの送信が必要なときはインタフェース層を呼び出す。受信データはソケット層に引き渡している。

インタフェース層は、物理ネットワークに対するパケットの送出、及び受信パケットのプロトコル層への引き渡しを行なっている。

一般に、出力関数は XX_output()、入力関数は XX_input()、入力キューは XX_intrq という名前を持つことが多い。

≪ カーネル入口 ≫ trap ⇒ ハンドラ呼出 ⇒ 引数チェック ⇒ 下層呼出
≪ ソケット層 ≫ ソケット・セマンティクスの提供
≪ プロトコル層 ≫ プロトコル処理
≪ インタフェース層 ≫ ネットワーク・インタフェース・ドライバ

図 3.1: 4.3BSD におけるネットワーク実装構造

3.2 仮想インタフェースからプロトコル処理部を呼び出す実装

3.2.1 vt 0.9

vt 0.9[107] は、トンネリング機能提供する仮想インタフェースであり、ddt WG の前身で実装された[1]。

vt 0.9 は次のことを目標に設計を行なった。

1. 4.3BSD Tahoe の TCP/IP code をベースとする。
   - 広く利用されている実装をベースとすることにより、vt を広く利用可能とする。
2. 既存の .o の変更を避ける。
   - TCP/IP のソースがなくても利用可能にする。
3. 拡張性のある構造にする。
   - 様々なプロトコルの組合せのトンネリングに対応できる。

vt 0.9 の実装はインタフェース・モジュールとレイヤ・ポンプアップ・モジュールのふたつの部分からなる。インタフェース・モジュールはインタフェース層に位置する。仮想ネットワークのプロトコル・ファミリは、このモジュールでサポートする。レイヤ・ポンプアップ・モジュールはソケット層あるいはプロトコル層に位置する。このモジュールは、キャリア・プロトコルの数だけ必要である。(図 3.1 参照)

[1] “vt” という名称は OSI の Virtual Terminal と紛らわしいため、vt で提唱された概念を ddt と呼ぶことになった。ソフトウェアの名称も変更予定である

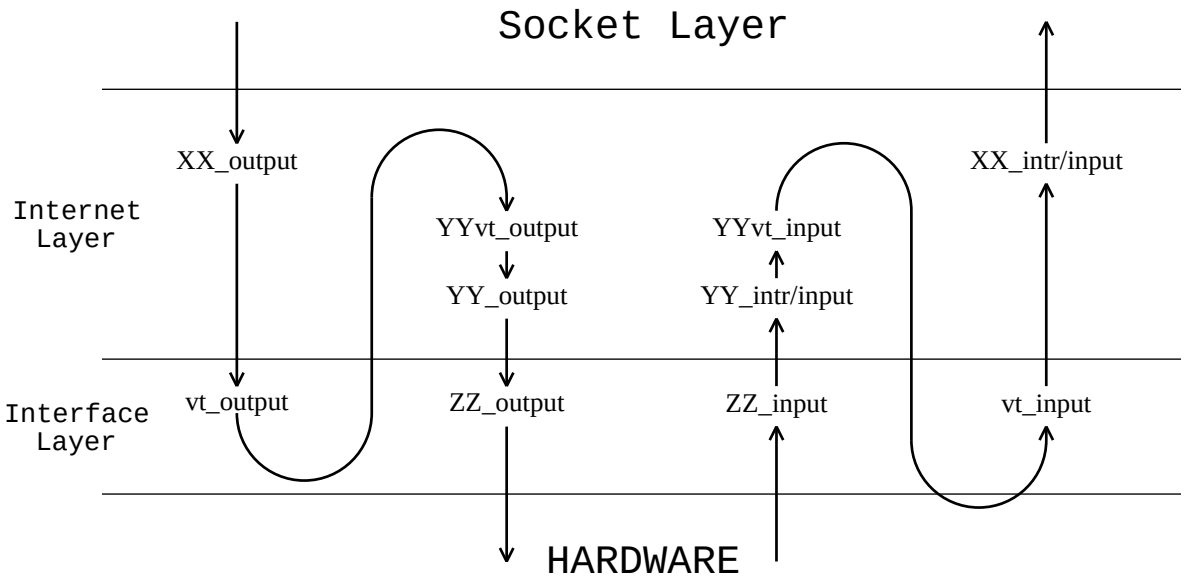

図 3.2: vt の構造

このように vt 0.9 をふたつの部分に分けて実装したことにより拡張性が高くなっている。現在、レイヤ・ポンプアップ・モジュールをプロトコル層に置いた実装が存在するが、ソケット層に置く実装も容易であろう。

現在の実装構造を図 3.2 に示す。この図では、トンネリングされるネットワーク・プロトコルを XX、トンネリングのキャリアとなるプロトコルを YY、実際の物理媒体を ZZ としている。図において、XX, YY, ZZ の処理部分は既存のものを無修正で用いることができる。インタフェース・モジュールは vt_* で示したものであり、レイヤ・ポンプアップ・モジュールは YYvt_* で示したものである。

現在の方式では、トンネルとして IP over IP、XNS IDP over IP を実装している。プロトコルとして、IP over IP では IP within IP を、XNS IDP over IP では XNS-IDP を用いている。

現在の実装にはまだいくつか課題がある。

1. 最適 MTU 選択
   vt による仮想ネットワークの MTU [2]の値の選択である。ふたつの矛盾する要求がある。

   (a) vt に突入時のフラグメント化を避けたい。
   　　vt の MTU を、近隣の 最小 MTU よりも大きくする。

[2]Maximum Transfer Unit の略

(b) vt 間におけるフラグメント化を避けたい。
vt の MTU を、近隣の 最小 MTU よりも大きくする。

現在の実装では、固定値で 576 にしている。将来的には動的最適 MTU 発見を行ないたい [108]。

2. 正しい ICMP を返す
トンネル通過中に発生した障害情報を実際の発信者に伝える方法として３つ考えられる。現実的には 2c のように無視するのが適当と考える。

(a) IP over IP の場合、発生した ICMP 情報を実際の発信者に転送する。
これは難しい。以下に理由を述べる。

- エラーとなったデータグラムのうち ICMP で返ってくるのは IP header + 8 octets だけである。この 8 octets の中には実際の発信者を特定できる情報は入っていない。
- Tahoe の TCP/IP の実装では、ctlinput() からは ICMP データグラムの中を見ることができない。万一 ICMP に発信者情報などを忍び込ませたとしてもそれを取り出すことは難しい。

(b) vt 端点において、障害情報を元にインタフェース up/down を判断する。
問題点としては次のものがある。

- インタフェース up/down 判断を真面目にするためには複雑なメカニズムが必要になる。
- 実際にエラーとなったパケットに対しては障害情報は返らない。

(c) vt 端点ではエラーを無視する。

- リンクが利用可能かどうかの判断は、上位の経路制御情報交換デーモンにまかせる。
- 実際にエラーとなったパケットに対しては障害情報は返らない。

3. 4.3BSD Tahoe 以外への移植
vt を 4.3BSD Tahoe 以外でも利用できるようにしたい。現在考慮しているのは 4.4BSD と System V STREAMS である。

4.4BSD では 4.3BSD Tahoe に比べて多くのプロトコル・ファミリがサポートされているようだ。4.4BSD に移植できると、マイナー・プロトコルも広域で利用できるようになるかもしれない。

移植に当たっての問題点としては次のものが考えられる。

(a) OS の TCP/IP の実装との親和性

- queue 内の data format

- mbuf か pbuf か、はたまた異なるメモリ管理か
- ip_output() の仕様
- STREAMS 対応

(b) OS に如何にして初期化してもらうか

- 誰に vt を attach してもらうか (誰に vt_attach() を呼んでもらうか)
- vt インタフェースの数の指定方法

3.2.2 nsip

4.3BSD Tahoe の XNS IDP の実装に含まれる XNS IDP over IP を提供する仮想インタフェース nsip も、XNS の仮想インタフェースから IP のプロトコル処理部を呼び出す形で実装されている。

3.3 仮想インタフェースからトランスポート・サービスを呼び出す実装

仮想インタフェースからトランスポート・サービスを呼び出す場合、4.3BSD Tahoe では socket 層を呼び出すことになる場合が多い。というのは、実装上 socket 層に依存していないプロトコルは限られており、x over TCP/IP をカーネル内部に実装する場合はほとんどこの方式になってしまうからである。

実装に当たっては vt 0.9 の構造を流用すれば可能であると考える。

3.4 仮想インタフェースとデーモンが協調して処理するよう実装

3.4.1 トンネル・ドライバ Pnet

複雑な仮想ネットワークを開発する場合、すべてをカーネル内に実装することは難しく、またデバッグもたいへんである。仮想ネットワーク実現のための処理をユーザ・プログラムとして実現し、カーネル内には仮想インタフェースとデーモンの間の橋渡しをするモジュールだけを入れる実装方式としてトンネル・ドライバがある。この方式の利点は、カーネル内モジュールが小さいため OS 依存部が少なく、移植性が高いことにある。[100] [109]。

トンネル・ドライバとは、IP (あるいはその他のプロトコル) に対するインターフェイスをデバイス (例えば /dev/tunXX) として提供する仕掛けのことである。すなわち、デバイスに書き込むことにより IP の入力にパケットデータを渡すことができ、ネットワークから出力される IP パケットがデバイスから読み込むことができる仮想デバイスであ

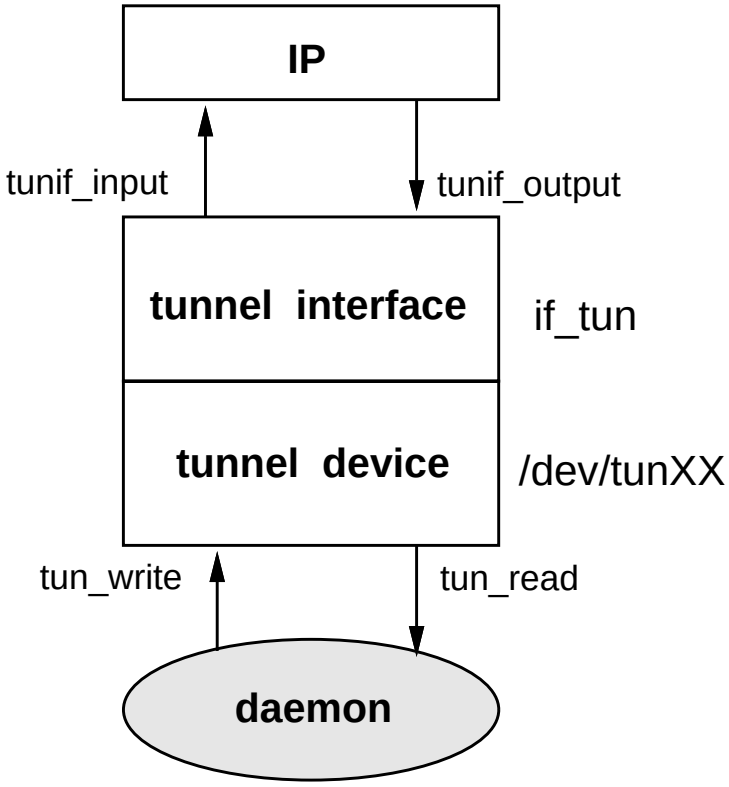

図 3.3: トンネル・ドライバの構造

る。その様子を図 3.3 に示す。仮想端末の裏表が ttypXX と ptyXX の組みで構成されるように、tunXX と仮想ネットワーク・デバイスの組みで IP プロトコルに対するインターフェイスが構築される[3]。

トンネル・デバイスを利用すれば、デーモンがデバイス・ファイルを介してカーネルからのデータ・パケットを読み出して、通信路で必要とされるカプセル化をおこなって、別のデバイスに書き出すことができる。

トンネル・ドライバではリンクレベルのカプセル化処理をおこなうことを目的としていたため、仮想インターフェイスに書き込まれたデータはデーモンよって処理され、別のデバイス (たとえばシリアル回線) に書き出されていた。ここで、別のデバイスにデータを与える代わりに処理されたパケットを再度 IP モジュールに入れてやれば、仮想ネットワークを構築することができる。

MST の PPP はトンネル・ドライバを利用して PPP over TCP/IP を実現している。

[3]ここから疑似ネットワーク・ドライバ (Pnet) という名称がついたのであろう。

3.5　まとめ

以上、ddt の実装手法についてまとめた。残念ながら STREAMS における実装方法については未調査である。

複雑な実装が必要な場合や、仮想ネットワークの研究用にはトンネル・ドライバの手法が、単純で効率の求められる仮想ネットワークの実現には vt の手法が良いと考える。

第 4 章

応用

ここでは ddt を利用することにより広がる世界を紹介する。

4.1 ネットワークの離れ小島

インタネットと各組織の接続においてなんらかの制限が行なわれることがある。このとき、ゲートウェイから遠い部署はインタネットへの直接アクセスは難しくなってしまう。経路の途中での制限が厳しい場合はなおさらである。どうしてもインタネットを利用したい場合は新たに回線を引かなければならなかった。

この問題は、入口ゲートウェイ近辺とその部署とをトンネリングによる仮想リンクで接続することにより解決可能である。当然、仮想リンクの端点周辺においては経路制御情報を慎重に扱わなければならない。これは実際に回線を引いた時も同様である。

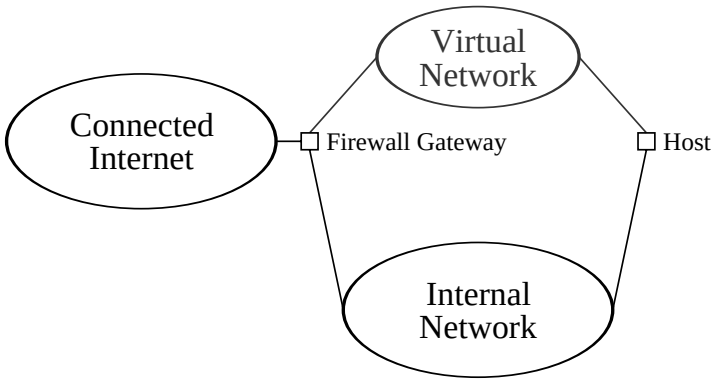

図 4.1: ネットワークの離れ小島を接続

また、ネットワークが分断された時に緊急避難的に仮想ネットワークで分断されたネットワークを接続するという利用方法もある。

4.2　新しいプロトコルの利用

現在大規模インタネットで利用可能なプロトコルは TCP/IP が中心である。それ以外のプロトコル、あるいは IP に新しい機能を付加したプロトコルは、広域での正しいルーティングは期待できない可能性が高い。

それらのプロトコルを扱えるホスト間でトンネリングによる仮想リンクを張ることにより、新しいプロトコルやマイナー・プロトコルの利用が可能となる。

4.3　トポロジーの補助線

あるサイト間の通信を特定の経路を用いて行ないたい場合、IP option のソースルーティングが利用できる。しかし、これをすべてのアプリケーションでもれなく行なうのはたいへんである。

そこで、サイトの出口のゲートウェイと経路途中サイトとの間にトンネルを作り、目的サイト向けのパケットはそのトンネルを通すようにすることで強制的に経路を特定することが可能となる。ただし、想定したトンネル経路の途中のリンクが落ちた場合、想定外の経路を通る可能性があるため注意が必要である。

歴史的には ddt のアイデアは、policy routing 上、この用途で応用するために生まれた。詳細は policy routing WG の報告書を参照されたい。

4.4　実験ネットワーク構築

2 点間リンクや広域 Ethernet 的ネットワークなどさまざまなトポロジーの仮想ネットワークを組み合わせることにより、研究実験環境として Virtual Internet を構築することができる。

それぞれの仮想ネットワークには既存のネットワークと異なるアドレスを与えることにすれば、既存の Internet と全く独立に Virtual Internet という新しい空間を作ることができ、そのなかで既存の Internet に影響を与えずに、研究実験等を行なうことができる。これは今後、実験環境が運用ネットワークと共存できる可能性のひとつを示している。

Virtual Internet 利用の有効な例として、新しい経路制御プロトコルの研究実験が挙げられる。従来は実験を行なうためには、物理的な環境を新たに作るか、ネットワークの一部を細心の注意を払って使うかしか方法がなかったが、Virtual Internet を利用すれば広域にかなり自由なトポロジーの環境を構築することができる。これは、policy routing WG で今後おこなおうとしている IDPR などの試用実験にも使えるのではないかと思われる。

Virtual Internet は、マルチキャストの実験研究上も非常に有用と考える。現在はまだ、マルチキャスト・パケットを処理できるルータ、ゲートウェイは多くない。ところが専用ルータや多くのゲートウェイにマルチキャスト機能を追加することは簡単ではない場合

が多い。そのため従来は、経路制御のしくみ自体にトンネリングを含める形でいびつな経路制御が行なわれてきた。ところが ddt を用いると、そのようなルータ等を隠蔽した理想的なマルチキャスト・ルータだけの Virtual Internet を構築することができ、その上で、現在 Multicast Communication WG が研究中のマルチキャスト経路制御の実験等を容易に行なえるのではないかと思われる。

4.5 Virtual Private Network

企業が社内ネットワーク間相互接続を行なう場合、侵入や盗聴の危険性を避けるために、一般に回線の共有は行なわない。ところが遠隔地を接続する場合はこれでは経済的負担が大きいため、安全にネットワークを共有する技術が望まれる。

ddt による Secure Network を利用すれば、インターネットを通信媒体とした安全な通信路が確保できるため、このような場面で用いることができる可能性がある。

4.6 Secure Network

インターネットで遠隔地と通信を行なう場合、複数の組織を経由して行なわれるため盗聴の危険性がある。現在の盗聴防止へのアプローチはアプリケーション・プロトコル・レベルで行なわれているため、遠隔地との通信を行なう可能性のあるアプリケーションすべての改造が必要となる。

ddt を用いて仮想ネットワーク・レベルで暗号化を行なうことにより、個々のアプリケーションを改造することなく盗聴を防止することができる。

4.7 移動ホスト

移動ホスト・サポート方式の中にはトンネリングを利用しているとみなせるものがある。一般にそれらは次のような方針で研究されている。

1. 移動サポート用に仮想ネットワークを定義する
2. 仮想ネットワーク内ではトンネリングにより経路制御する。
3. 仮想ネットワークは外部からは通常ネットワークと同じに見えるようにする。若干、周辺に協力を依頼することもある。

この観点から移動ホスト・サポート方式を見てみる。

Columbia 方式 [102] では次のようになっている。

1. ある subnet を仮想ネットワークとして使う。subnet 内には MSR (Mobile Support Router) と MH (Mobile Host) が存在する。

2. 移動サポート用 subnet が複数の物理ネットワークをまたがっている場合はトンネリングする。

3. 外から見ると通常のネットワークと同じに見える。

VIP 方式 [110] では次のようにみなすことができる。

1. 移動ホストはホームネットワークを持つ。ホストは移動後もこのホームネットワークに属する。ホームネットワークは物理ネットワークと完全に対応しているわけではないので、仮想ネットワークであると言えるであろう。プロトコル VIP により仮想ネットワークを構成している。

2. ホームネットワークに物理的に接続しているホスト発のパケットは GW が proxy arp してトンネリングする。移動中のホスト発のパケットは、外部初のパケットと同じ扱いになる。

3. 外部発のパケットは、基本的にはまずホームネットワークまで行き、そこから転送される。効率を上げるため、VIP を理解するルータが移動ホストの情報を知っており、トンネリングの途中経路でトンネルの出口を変更する。

これらの方式は視点を変えると、トンネリングを用いて仮想ネットワークを構築していると言うことができる。移動ホストも ddt の枠組で扱うことができるかもしれない。

4.8 Dynamic ddt

様々な性質の仮想ネットワークを動的に作成・破壊することにより、必要な時に必要な性質の仮想ネットワークを利用することが可能になる。TOS を考慮した経路制御と合わせて用いる必要があるかもしれない。また、動的に ddt を張るための調整プロトコルも必要かもしれない。以下に dynamic ddt 利用例を書く。

- secure telnet/smtp/ftp の場合

  1. TCP で接続する前に、接続先と secure ddt を張る
  2. TOS の secure bit を立てて通信する
  3. 通信が終ったら ddt を捨てる

- 大きな ftp をする場合

  1. TCP で接続する前に、接続先と compressing ddt を張る
  2. ftp-data は TOS の throughtput bit を立てて通信する
  3. 通信が終ったら ddt を捨てる

- 切れたネットワークを修復する場合
  1. 経路情報プロトコルでネットワークの接続性を監視
  2. ネットワークが切れた場合、ddt を動的に張りバックアップ
- 専用線を合い乗りする場合
  1. 専用線の物理インタフェースに、ネットワーク共有機能のあるフック・インタフェースを張りつける。
  2. 各フック・インタフェースの上に、バンド幅制限機能付きのフック・インタフェースを張りつける。
  3. 賢いバンド幅制限機能により、いろいろなポリシが実現できる
- ホスト/ネットワーク単位で経路指定をしたい場合
  1. あらかじめソース・ルーティング機能付の ddt を張っておく
  2. 全アプリケーションの通信は必ず ddt を通ってしまう
  3. 必要なくなったら ddt を捨てる
- vip による移動ホストの場合
  1. Stackable I/F 機能で vip0 の下に le0 を張り付ける
  2. vip0 には vip をつける。home net では le0 も同じアドレスにする
  3. le0 のアドレスは vip0 の制御の元で動的に変更する
  4. le0 が利用できない環境では vip0 の下に isdn0 を張り付ける

4.9 まとめ

以上 ddt の応用例を挙げた。利用目的を分類すると、自由なネットワーク・トポロジーの構築、特別な機能を持つネットワーク構築と言える。また、動的に ddt の生成・破壊を行なうことにより、ネットワークの自由度を増すことができる。

第 5 章

実験

ここでは vt 0.9 を用いた実験の報告及びそこでわかった問題点について述べる。vt 0.9 については 3.2.1 節参照のこと。

5.1 実験報告

現在、日本国内には把握している範囲では WIDE 参加組織 6 箇所、6 台で稼働している。以下に vt 利用の世界を紹介する。

図 5.1 は、omrongw.wg.omron.co.jp における “vtconfig vt0” の結果である。“vtconfig” は vt 対応の ifconfig である。

図 5.2 は wnoc-kyo-satgw.wide.ad.jp における “netstat -i” の結果である。vt0 から vt3 までが仮想インタフェース “vt” である。このように通常のインタフェースと同様に見えている。

図 5.3 は wnoc-kyo-satgw.wide.ad.jp における “netstat -M” の結果である。これは、マルチキャスト対応の netstat であり、-M オプションによりマルチキャストの経路制御表の情報が出力される。

図 5.4 は omrongw.wg.omron.co.jp から wnoc-kyo-satgw への traceroute の結果である。133.4.8.9 は物理インタフェースのアドレスであり、133.210.1.2 は vt のアドレスである。133.210.1.2 への tracroute の結果でわかるように、vt によりルータを隠蔽することができる。

図 5.5 は omrongw.wg.omron.co.jp から wnoc-kyo-satgw への ping の結果である。133.4.8.9 は物理インタフェースのアドレスであり、133.210.1.2 は vt のアドレスであ

```
$ vtconfig vt0
vt0: flags=811<UP,POINTOPOINT,MULTICAST>
        inet 133.210.1.1 --> 133.210.1.2 netmask ffffff00
        tunnel inet 133.210.4.4 --> 133.4.8.9
```

図 5.1: vtconfig の結果

```
$ netstat -i
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
am0   1500  133.4.8     wnoc-kyo-satgw   1403917     4   751687     0   147
am1   1500  192.244.23  192.244.23.33     124825     0    22093     0     0
vt0   576   133.18.224  133.18.224.2      463712     0    24706     1     0
vt1   576   133.18.240  133.18.240.2       16663     0   110699     1     0
vt2   576   133.210.1   133.210.1.2        25373     0   370906     1     0
vt3   576   131.206.32  131.206.32.2        1984     0    62133     0     0
```

図 5.2: netstat -i の結果

```
$ mnetstat -M
Virtual Interface Table
 Vif   Threshold   Local-Address   Remote-Address   Groups
  0         1      133.4.8.9
                                                   224.0.0.4
  1         1      192.244.23.33
                                                   224.0.0.4
  2         1      133.18.224.2
                                                   224.0.0.4
  3         1      133.18.240.2
                                                   224.0.0.4
  4         1      133.210.1.2
                                                   224.0.0.4
  5         1      131.206.32.2
                                                   224.0.0.4

Multicast Routing Table
 Hash  Origin-Subnet  In-Vif  Out-Vifs
   1   133.210.1         4    0* 1* 2  3* 5
   2   139.130.204       2    0* 1* 3* 4* 5*
   2   139.130.4         2    0* 1* 3* 4* 5*
   2   133.210.2         4    0* 1* 2  3* 5
   2   192.17.2          2    0* 1* 3* 4  5
(以下省略)
```

図 5.3: mnetstat -M の結果

```
omrongw$ traceroute 133.4.8.9
traceroute to 133.4.8.9 (133.4.8.9), 30 hops max, 38 byte packets
 1  omron-proteon.wg.omron.co.jp (133.210.4.2)  0 ms  10 ms  0 ms
 2  wnoc-kyoto-proteon.wide.ad.jp (133.210.3.1)  40 ms  40 ms  30 ms
 3  wnoc-kyo-satgw.wide.ad.jp (133.4.8.9)  30 ms  120 ms  50 ms

omrongw$ traceroute 133.210.1.2
traceroute to 133.210.1.2 (133.210.1.2), 30 hops max, 38 byte packets
 1  wnoc-kyo-satgw.wide.ad.jp (133.210.1.2)  50 ms  40 ms  40 ms
```

図 5.4: traceroute の結果

る。ping パケットは、行きに 2 回、帰りに 2 回合計 4 回 vt インタフェースを通る。したがって vt による遅延は平均時間の 1/4 の 5ms と考えられる。

5.2　経路制御

vt の実験から、vt 利用ホストにおいて経路制御に問題が発生することが分かった。図 1.6 において H_a から H_b への通信を行なうとき、次のことが分かった。

Na, Nb, Nab がすべて異なるネットワーク　　$\cdots$　問題なし

Nb, Nab が同じネットワークの異なる subnet　$\cdots$　問題あり

ここでネットワークが同じかどうかとは、Class A, B, C のネットワーク部分が同じかどうかを言う。

後者の場合の問題は、H_a において H_b の物理インタフェースに対する経路制御表のエントリがなくなってしまうというものである。ここで N_b、N_{ab} を含むネットワークが N_B であるとする。H_a が N_{ab} と接続すると、H_a は N_B の subnet と接続していることになる。そのため N_a 上のゲートウェイから来る N_B に関する経路情報は必要なくなり無視し始める。ところが実際には H_a は N_B のうち N_{ab} 以外のネットワークに対する情報を持っていないため、N_b とは通信できなくなってしまう。仮に N_b への経路として N_{ab} を選択してしまうと、vt の経路制御が無限ループしてしまう。このように、接続したネットワークの他の subnet に対する経路制御をどう解決するかが問題となる。

原因は、IP ネットワークの経路制御の単位が Class A, B, C などのネットワーク単位か、subnet 単位かの 2 階層しかなく、しかも subnet default (経路を知ってる subnet の補集合) が多くの場合指定できないことにある。つまり、経路制御表の宛先記述能力が低いことが原因である。

この問題は、次のいずれかの方法で回避可能である。

- 方針例

```
omrongw$ ping 133.4.8.9
(省略)
----133.4.8.9 PING Statistics----
101 packets transmitted, 101 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 39/55/190


omrongw$ ping 133.210.1.2
(省略)
----133.210.1.2 PING Statistics----
101 packets transmitted, 101 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 49/75/329
```

図 5.5: ping の結果

1. vt subnet 以外との通信は通常ネットワークを利用する
2. 相手の物理アドレスの属するネットワーク以外は vt を利用する。
3. 相手の物理アドレスの属するネットワークは通常ネットワークを利用する。それ以外は vt と通常ネットワークのうち都合の良い方を利用する。

- 方策例

1. static route を張る
2. default route に頼る
   subnet default が使えると楽
3. CIDR routing にする

5.3 経路情報管理

vt からの経路情報を周囲に広告する場合、vt の相手の物理インタフェースへの経路を絶たることにならないような技術が必要である。
次のいずれかの方法で回避可能である。

- 注意点

1. 物理アドレス宛をどの経路を通すか管理する。
2. 複数の vt の口が存在した時、吸込合いをしないように。

- 方策

1. 経路制御情報が広告される範囲内で static route を張る。
2. 無限大遠の host route info を出す
   この方法で実現可能かどうか確認していない。

5.4 Unnumbered Link

1.4 節で述べたように、ddt はモデル上 numbered link のみを対象としている。しかし、実用上は unnumbered link を実現できる。

ひとつ目の方法は、通常 4.3BSD で unnumbered link する方法と基本的に同じである。1.5 節で、ddt には仮想アドレスと物理アドレスがあることを述べた。unnumbered link を実現するには、このうちの仮想アドレスの付け方を 4.3BSD の unnumbered link と同じ方法にすれば良い。物理アドレスの付け方は unnumbered link 実現と関係ないため通常通りで良い。

物理アドレスが不必要になると便利であるが、経路制御上の問題から簡単にはなくせない。というのは、物理アドレスがなくなると vt の宛先アドレスと、実際にパケットを届ける先のアドレスを同一アドレスにせざるを得なくなってしまうからである。そうなった場合、vt を通るべきパケットを vt に投げることはできるが、いったん vt に入って IP でカプセル化されたパケットを投げる先も vt になってしまう。つまり無限ループが発生してしまうのである。これを避けることができれば物理アドレスをなくすことができ、ひいては unnumbered link 実現につながる。

ふたつ目の方法は、この問題を解決することで実現する方法である。vt 内部から IP パケットを送出する瞬間だけ、vt を DOWN させる。これにより、パケットは vt 以外のしかるべきインタフェースに投げられてネットワークに出ていく。vt を DOWN させるのは便宜的なものであるため、他のプロセスに影響しないようロックなどの措置が必要である。この方法は、vt の相手の属するネットワークへの経路が存在することを前提としているため、5.2 節で述べた問題がある場合には利用できない。

5.5 まとめ

以上、本章では vt 0.9 の実験結果の報告を行なった。

第 6 章

93 年度研究計画

ddt のアイデア自体、生まれてからまだ間がないこと、WIDE ddt WG が生まれて間もないことから研究成果が荒削りで解決すべき問題点も多い。研究対象の枠組自体がまだ流動的である。

この状況を踏まえて、93 年度は次のことを目標として研究活動を行なっていきたい。

- ddt のモデル化、問題点の解決 (93 年度前半予定)
  - ddt の本質を明らかにする。
- モデルに沿った実装と評価 (93 年度後半予定)
  - 実装を公開。BSD への寄付も検討。
  - ゆくゆくは世界中のルータに ddt が載ることを目指す。
- インフラとしての実験環境の提供 (93 年度通年予定)
  - マルチキャスト研究実験環境支援