# 第 8 部

# Mobile and Ubiquitous Computing

# 第 1 章

# Virtual Internet Protocol version 2

## 1.1 はじめに

近年、幾つかの移動体通信用のプロトコルが提案され [58]、実際にインターネットの世界でも実験的に運用される段階となっている。これらの実験により、移動体通信の有用性は再認識されたと言える。しかし、このような実験はあくまで実験として移動透過性を実現することを最重要視して行われてきたため、ほとんどのプロトコルは実際のインターネットでの使用において、セキュリティの面で問題を含んだままとなっている。

現在、インターネットには 100 万を越える計算機が接続されており、中には不正に他人になりすましたり、盗聴したり、通信を妨害するユーザが存在することは否定できない。更に、近年、インターネットの商業利用が進んでおり、インターネットにおけるセキュリティの問題は避けることができない問題となっている [59]。

我々がこれまで提案してきた移動体通信用プロトコルである Virtual Internet Protocol (VIP)[60] も例外ではなく、実際のネットワークで使用する場合、なりすましが簡単にできる等のセキュリティ面での問題を含んでおり、このことは前述の理由により早急に解決しなければならない問題となっている。そこで我々は VIP を現在使用されている IP レベルのセキュリティを確保できるように改良することにした。ここで注意しなければならないのは本章で述べている解決案はあくまで VIP を導入することによってできたセキュリティホールにのみ着目している点で、現行の IP が持つセキュリティホールに対しては無力である。現行の IP におけるセキュリティについては Internet Engineering Task Force (IETF) 等で議論されており、それらとの併用で解決すべきである。

本章では始めに現行の VIP (VIPv1) の概要について述べ、その問題点を 1.3 節で明らかにし、1.4 節で VIP Version 2 (VIPv2) について述べる。更に 1.5 節で実装について述べ、1.6 節で評価を行い、1.7 節で本稿をまとめる。

## 1.2 VIPv1 の概要

本節では仮想ネットワークの概念および現行の VIP (VIPv1) の概要について述べる。VIPv1 には AMT の操作において大きなセキュリティホールがある。

### 1.2.1 仮想ネットワーク

現在の IP では通信相手を指定するために IP アドレスを使用している。全ての通信はこの IP アドレスによってパケットが経路制御され通信相手に配送されることによって行われる。ここで IP アドレスは次の 2 つの役目を持つ。

- 位置識別子
- ホスト識別子

移動計算機環境で IP を使用する場合、IP アドレスが持つこの 2 つの役目のために次のような経緯で「移動できない」という問題が生じる。

1. IP アドレスが位置情報を含んでいるため、移動ホストは移動後に自分の IP アドレスを移動した先のネットワークの IP アドレスに変更しなければならない。
2. IP アドレスがホスト識別子として使用されているため、他のホストが移動ホストを移動前と後とで移動ホストを同じホストとして認識できなくなる。
3. 相手が識別できないために通信を続けることができなくなる。

このような問題を解決するために我々は仮想ネットワークの概念を提案し、それを IP に適用した。仮想ネットワークの概念では Open Systems Interconnection (OSI) の 7 階層モデルのネットワーク層を位置識別子で構成される既存のネットワーク層とホスト識別子で構成される仮想ネットワーク層の 2 層に分け、既存のネットワーク層では位置識別子を使った経路制御のみを行い、仮想ネットワーク層では各ホストに用意された Address Mapping Table (AMT) を用いてホスト識別子と位置識別子との対応付を行うことにより、上位層に対してホスト識別子による位置に依存しない仮想的なネットワークを提供する。仮想ネットワークの概念はあらゆるコネクションレス型のパケット通信プロトコルに適用でき、IP でなくともこの概念を使って移動透過性をトランスポート層より上位の層に提供することができる。

VIP では位置識別子を IP アドレス、ホスト識別子を VIP アドレスと呼ぶ。IP アドレスと VIP アドレスは AMT による対応付けが行えない時に、VIP アドレスを IP アドレスのデフォルト値として使用することができるように同じフォーマット (4 オクテットからなる識別子) となっている。

### 1.2.2 拡散キャッシュ法

前述のようなアドレスの対応付けを行う場合、AMT をどのように管理するかが問題となる。VIP ではこの問題を拡散キャッシュ法によって解決している。

拡散キャッシュ法は AMT の管理に必要なオーバヘッドを最小限にするために考案された方法で、通常のデータパケットを覗見することによって AMT エントリの管理 (作成/更

新/削除) を行っている。実際には VIP ではパケットヘッダの中に送信者の VIP アドレス、IP アドレスおよびその組みのバージョン番号が含まれており、ホストやルータはパケットを受信/中継する際にその情報を使って自分の AMT を更新している。

### 1.2.3　VIP の動作概要

VIP では基本的にホストやルータが AMT を使って VIP アドレスから IP アドレスへの対応付けを行うことによって動作している。まずパケットは送信時に送信ホストによってアドレスの対応付けが試みられる。送信ホスト上に受信ホストに対する AMT エントリが存在していれば受信者 IP アドレスとしてそのエントリにある IP アドレスがパケットヘッダに設定されて送信される。もし、エントリが存在していなければ受信者 IP アドレスとして受信者 VIP アドレスがそのまま利用される。送信されたパケットはルータを通る度にルータによってルータ上の AMT エントリと比較され、ルータが更に新しいバージョン番号を持つ AMT エントリを持っていた場合、ルータは対応付けをやり直し、パケットヘッダを変更するとともに古い対応付を行ったホストまたはルータに対し AMT エントリを削除するように指示する。対応付けをやり直されたパケットは新しい対応付に従ってネットワーク上で経路制御され受信ホストまで配送される。

この機構によって受信者へパケットを確実に届けるため、VIP ではホームネットワークを導入している。ホームネットワークとは移動ホストの VIP アドレスを IP アドレスとして考えたとき、その IP アドレスを含むべきサブネットワークのことで、ホームネットワーク上のルータ (ホームルータ) はその子ホスト (そのサブネットワークをホームルータとする移動ホスト) に対する AMT エントリを必ず持っていなければならない。これによりパケットの送信者が受信者に対する AMT エントリを持っていなくても受信者 VIP アドレスを受信者 IP アドレスとして設定してパケットを送信することによりホームルータまで経路制御され、ホームルータで正しい対応付けが行われて移動ホストにパケットを配送することが可能となる。ホームルータは移動ホストが移動した時にホームネットワークに対して送信するコントロールパケットによって常に最新の対応情報を持つことが保障されている。

## 1.3　問題点

### 1.3.1　移動体通信の性質

広域網での移動体通信では移動ホストの移動範囲の制約はできるだけ少ない方が望ましい。しかし、この性質は逆に他のホストがあるホストになりすます事が簡単にできてしまう環境を作り出すことになる。これまでの IP のネットワークでは IP アドレスと接続位置との間に強い相互関係があったため、他のホストになりすますためには、ホストをなりすまそうとするホストと同じセグメントに接続しなければならなかった。しかし、VIP 環境

下ではホスト識別子 (VIP アドレス) と 位置識別子 (IP アドレス) を分けたために同じセグメントにホストを接続することなく、他のホストになりすますことができる。

同じセグメントに接続しなければ他のホストになりすますことができない、という制限は非常に厳しいもので、実際に他人に気づかれることなく、このような行為を行うのは難しい。しかし、VIP 環境下ではデータパケットやコントロールパケットを送信することによって通信相手の計算機上に AMT エントリを作るだけでなりすましができる。また、なりすましを行わないまでも、途中のルータの AMT エントリを他の方向に向けることによって正当な移動ホストの通信を妨害することができる。

また、移動によってパケットが通る経路が変わるため、通信経路上での盗聴/ 改竄が行われる可能性が高くなる。移動体通信の環境では一般的に通信経路の安全性を保証する事が難しく、この問題を解決するためにはパケットを暗号化する等の何らかの防御策が必要がある。

### 1.3.2 VIP の問題点の考察

以上のように VIP による移動体通信環境における問題点は大きく、

- AMT の変更により、なりすましが簡単にできる。
- AMT の変更により、通信を妨害できる。
- 通信路における盗聴/改竄を防ぐ事ができない。

の 3 つに分けられる。

ここで 1 番目のなりすましの問題と 2 番目の通信妨害の問題は VIP 固有の問題である。しかし、3 番目の盗聴/改竄の問題は VIP 固有の問題ではなく IP 自体の問題であり、VIP で解決すべき問題ではない。よって、VIP では今回は AMT の管理についてのみセキュリティを強化し、通信路上の盗聴/改竄の問題については現在、他の分野で研究されている方法に譲る。

### 1.3.3 AMT の管理

VIP において AMT に関する操作はエントリの作成、更新、削除の 3 つに分けられる。これらの操作はそれぞれ次のような場合に行われる。

- エントリの作成
  - CAMT (Create an AMT entry) コントロールパケットの受信
  - データパケットの受信
- エントリの更新

  - Version の新しい CAMT コントロールパケットの受信
  - Version の新しいデータパケットの受信

- エントリの削除
  - InvAMT (Invalidate an AMT entry) コントロールパケットの受信
  - エントリのタイムアウト

前章で述べたようななりすまし等の問題はこれらの操作の時に操作の原因となったパケットを認証することによって解決できる。

### 1.3.4　認証機構に関する考察

VIP 環境においてなりすまし等を行うためには以下の 2 つのケースが考えられる。

1. 偽った VIP アドレスを設定し、実際にデータパケットやコントロールパケットをパケットをインターネットに対して送信する。
2. 正しいホストが送信したパケットを盗聴・記憶しておき、正しいホストが切断された後、リプレイアタックを行う。

1 の場合には単純に送信者の VIP アドレス、IP アドレス及びそのバージョン番号の組みを認証できれば問題は解決する。このためには IP アドレス、VIP アドレス、バージョン番号、保持時間を使ってデータパケットやコントロールパケットが送信される時に予め認証子を算出し、付加すればよい。

2 の場合には更に複雑な機構が必要になる。リプレイアタックによるパケットは基本的に正当なホストが送信したものであるため、当然、そこに含まれている情報は全て正しいものである。そこで、リプレイアタックを防ぐためには認証されるホストと認証するホストの間にパケットに含まれる情報以外に何らかの信用できる情報が必要である。VIPv2 ではこの情報として時刻情報を利用することにした。時刻情報を共有情報として利用するためには 2 つのホスト間の時計が合っているという条件が必要になる。このため、このレベルで認証をおこなうにはホスト (ルータ) が時計を内蔵している必要があるが、これは現状ではさほど厳い条件ではないと判断した。また、現在のインターネット環境では Netwrok Time Protocol (NTP)[61] 等が使われており、時計を合わせることも簡単にできる。実際にはパケットを受け取った場合、ある一定時間以内のタイムスタンプを持ったものについてだけ処理を行い、その時間を越えたものについては処理を行わない。

このようなことを踏まえて、VIPv2 では 3 つの認証レベルを設けた。3 つの認証レベルを表 1.1 に示す。

AMT エントリの作成/更新、および、InvAMT コントロールパケットによる AMT エントリの削除の場合にはパケットの認証を行った後、VIPv1 と同じ操作をすればよい。タイ

表 1.1: VIPv2 における認証レベル

| レベル | 機能 | 備考 |
|---|---|---|
| 0 | 認証を行わない。 | VIPv1 と同じ |
| 1 | 単純ななりすまし、妨害を防げる。 | 認証子の付加 |
| 2 | リプレイアタックを防げる。 | タイムスタンプの併用 |

ムアウトによる AMT エントリの削除の場合は単にエントリを削除する。ただし、タイマの更新の際、エントリの認証レベルが高いと更新時にタイムスタンプを調べ、ある一定の値より古ければタイマの更新を行わない。これによりリプレイアッタク等も防ぐことができる。

### 1.3.5 認証方式に関する考察

VIPv2 では認証方式として RSA 暗号 [62] と MD5[61] を使用する。RSA は公開鍵と秘密鍵を使い暗号化、署名ができる暗号系で素因数分解の難しさを利用している。近年、RSA では鍵の予測に対して十分な鍵の強度を得るために 512 または 1024 ビット程度の鍵がよく使用される。VIPv2 では 512 ビットの鍵を用い RSA を暗号化ではなく電子署名として利用する。

RSA は鍵を公開できる、鍵を予測しにくい等の優れた性質を持つが、認証のために多くの計算量が必要である。従って全てのパケットに RSA の認証子を付加して認証を行う事は現実的ではない。そこで VIPv2 ではより少ない計算量で認証が行えるように MD5 を併用している。MD5 を使う場合、鍵として 16 バイトからなるバイト列を用意しておき、その鍵と、送信者の VIP アドレス、IP アドレス、バージョン番号、保持時間、タイムスタンプについてメッセージダイジェストを計算する。MD5 では計算した結果は 16 バイトのバイト列になるが VIPv2 ではデータパケットにおける IP パケットヘッダ長の制限のために更に 8 バイト毎に排他的論理和をとり 8 バイトに折り畳んで使用する。

## 1.4 VIP Version 2

これまで述べて来たことをもとに、VIP Version 2 (VIPv2) を設計した。本節では VIP に AMT 管理時のセキュリティ機構を導入した (VIPv2) について説明する。

### 1.4.1 パケットフォーマット

VIPv2 におけるパケットフォーマットを図 1.1 に示す。

VIPv2 ではコントロールパケットを IP データとして定義した。これは RSA を使用した場合、認証子の部分が IP ヘッダの長さの制限より長くなってしまうためである。また、

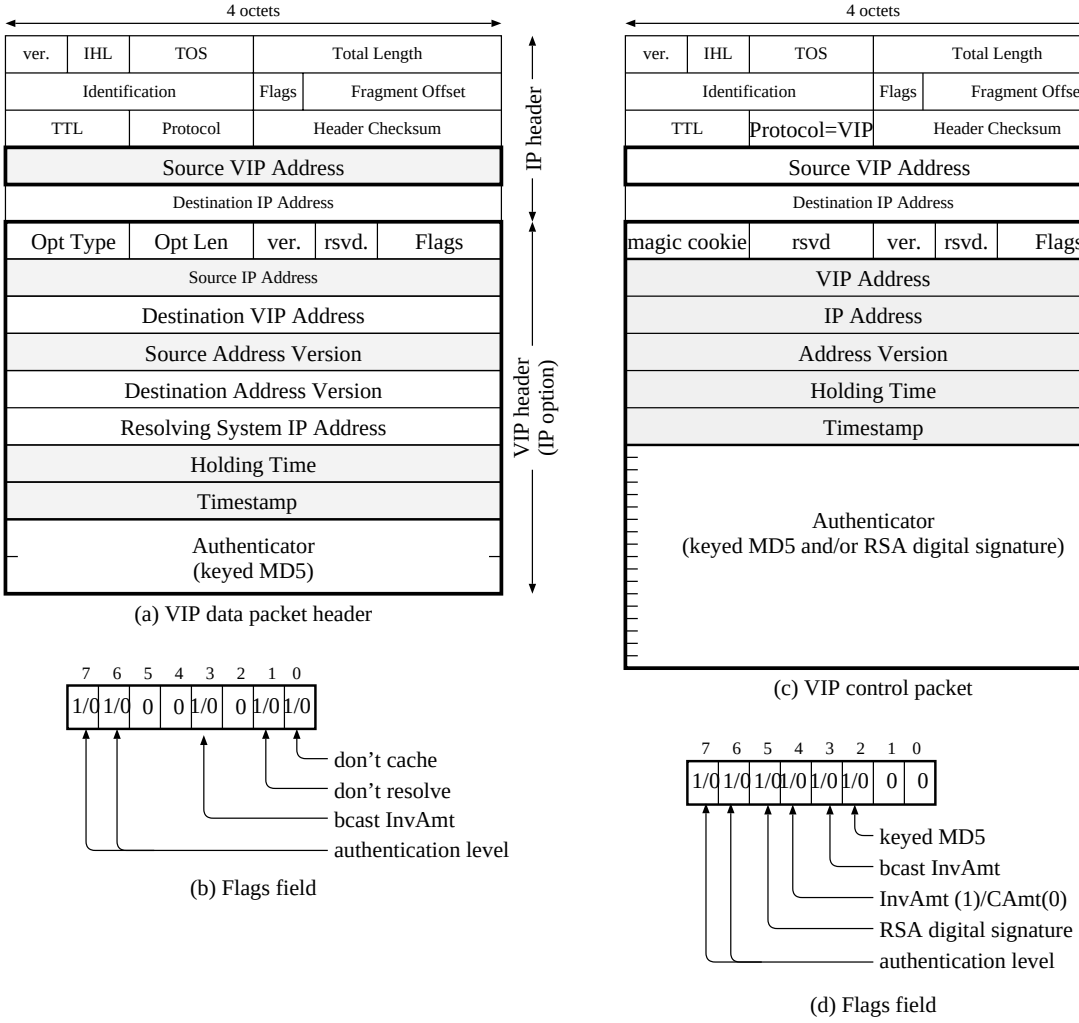



図 1.1: VIP のパケットフォーマット

前述のようにデータパケットの方でも MD5 の計算結果の 16 バイトをを全て使用するのではなく、8 バイト毎に排他的論理和をとり、その結果を認証子として使用している。

データパケットでは図 1.1 のハッチングの部分と 16 バイトの鍵を使って鍵付 MD5 を計算し、更に上位 8 バイトと下位 8 バイトの排他的論理和をとって認証子として使用している。

コントロールパケットではデータパケットと同じ鍵付 MD5 の結果を排他的論理和を使って 8 バイトにした認証子か、または RSA を使った電子署名を用いて認証を行う。

### 1.4.2　AMT エントリフォーマット

図 1.2 に AMT エントリの例を示す。AMT エントリにある Authenticator フィールドは AMT エントリを作成/更新した時の Authenticator を保存しておくためのフィールドで

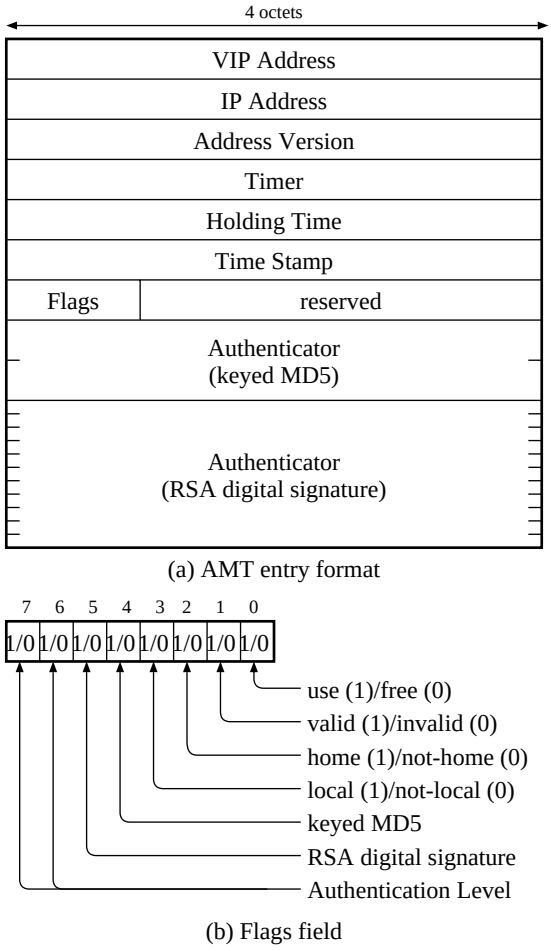



図 1.2: AMT エントリのフォーマット

2 度目以降の更新の負荷を軽減する役目を果たす。

また、`Flags` フィールドに Authentication Level, RSA digital signature, keyed MD5 の 3 つのビットフィールドが追加されている。これらはそれぞれ、認証レベルを 2 進数で示したもの、RSA digital signature が設定されているかどうか、keyed MD5 が設定されているかどうかを示す。

### 1.4.3 認証

VIP ホストはデータパケットを送信する時、全てのフィールドを設定した後、鍵付き MD5 によりメッセージダイジェストを計算する。計算された 16 バイトの計算結果は更に上位 8 バイトと下位 8 バイトの排他的論理和をとり、ヘッダに格納されて送信される。このパケットを受信/中継するホスト (ルータ) は自分が持つ AMT エントリのバージョンと比較して自分が持つ AMT エントリのバージョンが古く、また、そのホストに対の鍵を持っていれば MD5 を計算、比較し、実際に AMT エントリを更新する。もし、MD5 が正当なも

のでなければ AMT の変更は行わない。また、受信/中継するパケットが自分の AMT エントリと同じバージョンである場合、タイムスタンプも同じなら AMT エントリに保存してある認証子 (Authenticator) を使って単に認証子の比較を、異る場合は MD5 を計算しなおしパケットが正当なものかどうか判断し、AMT エントリのタイマの更新を行う。

また、対応付けの際、古い AMT エントリを発見した場合、AMT エントリに保存してある AMT エントリが使える場合はそれを使って、使えない場合でその AMT エントリのホストの MD5 の鍵を知っている場合はそれを使って古い対応付けを行ったホストに対して InvAMT コントロールパケットを送る。

一方、コントロールパケットの方は VIPv2 と機能的には認証機構が付いた以外はとくに変化はない。実際にコントロールパケットを受信/中継するホストまたはルータは処理を行う前に認証子の確認を行い、正当なパケットだと判断されたら実際の処理を行う。

以下に AMT 操作の手順を示す。

- エントリの作成
  パケットの forward (if necessary)
  $if$(鍵を持っていない && $CAMT$)
    鍵の取得
  $if$(鍵を持っている)
    $if(AL(\underline{PKT}) == 2)$
      TS による有効性のチェック
    $if(AL(\underline{PKT}) >= 1)$
      RSA/MD5 の計算、チェック
    AMT エントリの作成

- エントリの更新
  パケットの forward (if necessary)
  $if$(鍵を持っていない && $CAMT$)
    鍵の取得
  $if$(鍵を持っている)
    $if(AL(\underline{AMT}) == 2)$
      TS による有効性のチェック
    $if(AL(\underline{AMT}) >= 1)$
      RSA/MD5 の計算、チェック
    AMT エントリの更新
    CAMT を broadcast

- タイマの Update
  パケットの forward。(if necessary)
  $if$(鍵を持っていない && $CAMT$)

鍵の取得
$if$(鍵を持っている)
$if(AL(\underline{AMT}) == 2)$
TS による有効性のチェック
$if(AL(\underline{AMT}) >= 1)$
RSA/MD5 の計算、チェック
タイマの Update

- エントリの削除 (InvAMT)
パケットの forward (if necessary)
$if$(鍵を持っていない && $CAMT$)
鍵の取得
$if$(鍵を持っている)
$if(AL(\underline{AMT}) == 2)$
TS による有効性のチェック
$if(AL(\underline{AMT}) >= 1)$
RSA/MD5 の計算、チェック
AMT エントリの削除
InvAMT の伝播

## 1.5 実装

VIPv2 の実装は BSD 系の UNIX オペレーティングシステムである SONY NEWS-OS4.2 および BSDI BSD/386 1.1 上で行った。VIPv2 の実装は RSA の計算など計算量の大きい処理を必要とするため VIPv1 の実装と異り多くの部分をユーザ空間におけるデーモンプロセスとして実装した。これは UNIX カーネルが non-preemptable であるために認証子の計算等に不適切である、ユーザ空間で実装する事によりプロトコルの変更などが簡単に行え実験に向いている、等の理由のためである。

VIPv2 ではデータパケットを送信する際に MD5 を計算しなければならないためカーネルのなかに MD5 の処理ルーチンを持っている。しかし、AMT の操作をカーネル内で行わない実装にしたため、VIPv1 の実装に比べ、カーネルへの実装は簡単なものとなった。また、コントロールパケットを IP データにしたことによって VIP オプションの型が一つになり、更に簡単なものになっている。

AMT の操作部は全てデーモンプログラムとして実装されている。デーモンプログラムはコントロールパケットやデータパケットをカーネルから受取り、認証子の計算を行ってパケットの正当性を評価し、AMT の操作を行っている。また、AMT エントリのタイムアウトについても同様にデーモンで管理している。実際のカーネル内部の AMT エントリの操作は `ioctl()` によって行なっている。

表 1.2: MD5 の計算に必要な時間

| 測定機種 | 時間 (uSec) |
|---|---|
| NWS-3470(RISC NEWS) | 216 |
| NWS-1750(CISC NEWS) | 716 |
| 486DX2-66 | 91 |

## 1.6　評価

VIPv2 では主に AMT 操作時の認証について改良を行った。現在、まだ性能評価は行っていないが、定性的には十分、当初の目的を達成する事ができた。シミュレーションの結果では MD5 の計算におおよそ表 1.2で示した時間が必要であることが分かった。

認証デーモンに認証時の情報を報告する機能を付加して監視しながら実験運用を行った際、なりすましを試みたホストなどを確実に検出できた。また、UNIX における実装も VIPv1 のころに比べてカーネルの変更部が少なくなった分、簡単になった。

## 1.7　まとめ

本稿では VIP の AMT の管理に認証機構を導入した VIPv2 について述べた。VIPv2 では VIP を導入する事によってできたセキュリティホールの問題を解決し、実際に広域網で使用してもなりすまし、経路妨害等を防止する事ができるようになった。これにより VIP はほぼ実用の域に達したと言うことができる。今後、実際に性能評価などを行い、更に改良が必要かどうか検討する必要がある。

また、VIPv2 では鍵の配送機構について言及していない。現状ではオフラインで鍵を予め配っておき、それを使用している。しかし、より快適に運用するためには鍵の配送をオンラインで行うための機構を用意すべきである。

現在、VIPv1 は WIDE Internet 内で広域実験を行っている。今後、VIPv2 についても広域実験を行い、実際のインターネットでの動作を確認したいと考えている。

# 第 2 章

# サービス名による通信相手の指定

## 2.1 はじめに

近年、ネットワークアプリケーションの分野では通信相手の変化が起こっている。初期の頃の計算機ネットワークでは通信は計算機システム間において行なわれていた。しかし、近年の計算機ネットワークの利用体系を調べると、この様な体系の通信はむしろ稀で、殆どのアプリケーションが予めネットワーク上で提供されている「サービス」の利用、という形をとっていることがわかる。また、その他にも、人と人のコミュニケーションに利用されたり、蓄えられたデータへのアクセス手段として利用される例も少なくない。

しかし、現在使われているプロトコル群のネットワークアーキテクチャを見ると、相手のホストを指定しなければ経路を制御できないものが殆どである。本研究では、この様な点に注目し、サービス名をホスト名に対応づけることにより、サービス名から自動的にサーバホストを選択し、相手のホストの明示的な指定を不必要とするための機構を提案し、実装する。

## 2.2 用語解説

本文に入る前に、ここでは本稿で広く使用される幾つかの用語について説明/ 定義しておく。

**ホスト**
: 計算機ネットワーク上のある計算機。

**ノード**
: 計算機ネットワーク上のあるネットワーク機器。但し、ブリッジ等、リピータ等の電気的な橋渡しをする機器は除く。ホストに加えて専用ルータ等も含む。

**リモートホスト**
: 計算機ネットワークを介して通信を行う相手の計算機。

**ローカルホスト**
リモートホストの対語で自分の計算機。

**サーバ**
サービスを行うプログラム。サービスについては本文参照。

**エンティティ**
情報交換実体。通信を行うプロセスやネットワークソフトウェア、またはそれらを操作するものを指す。

## 2.3　計算機ネットワークにおけるサービス

### 2.3.1　サービスとは

計算機ネットワークの分野においてサービスと言う言葉は次のような 2 つの意味で使われている。

- 後述の階層モデルの$< N >$層が$< N + 1 >$層に対して提供する機能
- あるホスト (アプリケーション) が他のホスト (アプリケーション) に対して提供している機能

以後、本稿ではサービスと言う言葉は後者の意味で使う。

サービスとはあるプロセスが、他のプロセスが自分では解決できない問題等について、情報または処理能力を提供することである。ここで、2 つのサービスがある場合、サービスを受けるホストがその 2 つのサービスを利用した時に得る情報または処理の種類が同じである場合、2 つのサービスは同じサービスであると定義する。ここで、サービスには次の様な 2 つのものが存在することに注意する必要がある。

- どのサーバを利用しても同じ情報または処理結果が得られるサービス
- 利用したサーバによって情報または処理結果が異なるサービス

一般に分散データベースの検索などのサービスは前者の性質を、`finger` 等のサービスは後者の性質を持っている。

### 2.3.2　移動計算機環境とサービス

WIDE Project では、これまで、移動型計算機を支援する為のプロトコルである **Virtaul Internet Protocol** (**VIP**) [60]–[64] を提案してきた。VIP は、実際に BSD 系の UNIX オペレーティングシステムや MS-DOS 上に実装され、現在、広域網での運用実験に入っている。この様な中で幾つかの問題が明らかにされた。

現在提案されている移動計算機のためのネットワーク層プロトコルが提供している機能は、全てのプロトコルにおいてほぼ等価であり、相手の計算機が何処へ移動しているかに関わらず、それを意識することなく、パケットを通信相手に届ける機能を提供している。これによりユーザは、相手の計算機が、ネットワークのどの位置に接続されているかを意識することなく、パケットを送受信すること (相手の計算機を指定すること) ができるようになる。

これらの、プロトコルが実際に広く利用されはじめるのは近い未来であり、それによって計算機のネットワーク移動透過性をユーザに提供することができるようになるであろう。しかし、前述の様な移動体通信用のプロトコルを使用した場合でも、ユーザは快適にネットワーク上を移動することはできるようになるが、快適にネットワークアプリケーションを利用することは実際にはできない。

現在の一般的なサーバクライアントモデルを採用しているアプリケーションでは、クライアントは設定ファイルなどによってサーバを予め指定しておく必要がある。これは、自動でサーバを発見するプロトコルを持っているアプリケーションでも、起動した時にサーバを発見し、アプリケーションの終了まで、そのサーバが利用できる限りそれを使うという意味では同じである。つまり、通常のサーバクライアント型のアプリケーションはサーバを指定すると最後まで、そのサーバを利用する。

現在の広域ネットワークではネットワーク的に遠くなれば Round Trip Time (RTT) も大きくなり、パケットの紛失等も多く起きるようになる。つまり、現状のこの様な計算機ネットワークを利用する限り、ネットワーク的に遠くにあるサーバを利用するのは効率が悪い。しかし、実際に現在提案されている移動型計算機用のプロトコルを利用した場合、クライアントからサーバへの到達性が保証される為、移動後も遠くにあるサーバを利用することになり、快適なアプリケーションの動作は望めない。

このような問題を解決し、快適なアプリケーションの動作を実現する為には、移動した先でサーバを近くにあるものに変更すればよい。しかし、現在提案されているような移動通信用のプロトコルを利用した場合、アプリケーションにとって移動を隠すことを目的としている為、逆にこれが原因となってクライアント (移動ホスト) は移動を検知し、サーバを切替えることは難しい。

また、何らかの方法でクライアントアプリケーションが自ホストあるいはサーバホストの移動を検知できた場合でも、サーバ切替えを行なう場合、当然、アプリケーションは切替えに先立ってサーバの存在位置を発見しなければならない。しかも、発見したサーバは自分からみた場合、あるいはシステム全体からみた場合に、一番効率よく働く物でなければならない。この様なサーバ発見は次の様な問題を含んでいる為、非常に難しい問題である。

- ネットワーク的な位置をはかることが難しいため、どのサーバが一番近いのかわからない。
- サーバをサーバ発見サーバの様なものを介して検索する場合、移動先ではじめにどのサーバに接続してサーバ発見を行なえば良いのかわからない。

- 高速な移動ではサーバを発見するための時間さえ与えられないことがある。

### 2.3.3　サーバクライアントモデルの耐故障性

サーバクライアントモデルを採用しているネットワークアプリケーションでは、サーバが故障を起こすと、全てのクライアントホストも利用できなくなる。そこで、この様な場面に対応するため、通常複数のサーバを用意しておくなどの対策が取られている。しかし、実際にこの様な代替サーバをうまく使うのは難しい。

サーバクライアントモデルの一つの欠点としてサーバの故障がシステム全体に大きな影響を与えるという問題がある。サーバクライアント型のアプリケーションは一般的にクライアントだけでは動作せず、サーバホストが故障した場合、全てのクライアントホストは正常に動作しているにも関わらず、システム全体が動作不可能となる。

また、現在の計算機ネットワークは、信頼性が高いとは言いがたい。サーバホストが故障せずとも、他のルータなどで故障が起こった為にサーバとクライアント間の通信ができなくなるという事態も考えられる。この様な場合、他の経路を使って到達できるサーバを探す必要がある。

サーバクライアントモデルに基づいて設計されたアプリケーションを使用している場合、サーバが故障すると一般的にクライアントは代替サーバを利用しようとするか諦めるかのどちらかである。代替サーバを利用する場合、現在のアプリケーションの実装では次の 2 つの方式のどちらかによってサーバを切替えているのが普通である。

- タイムアウト、再送回数の超過等により、クライアントアプリケーションソフトウェアがサーバの故障を知り、予め設定されている代替サーバに自動的に切替える。

- ユーザがサーバの故障に気付き、明示的にサーバを変更する。

自動的な切替えでは、切替え時に予め設定されている代替サーバを使うのではなくサーバ発見をもう一度行なうものもある。しかし、このような機能を実現する為には、サーバ、クライアント共に、この様な機能を持つものに変更する必要があり、アプリケーションプログラマの負担を増加させる結果となる。

## 2.4　サーバ発見/切替え

2.3.2 節や 2.3.3 節で述べたような問題は、

> **全てのサーバが等価なサービスを行っているようなサービスを使う場合でも、サービスを受ける為には特定のサーバホストを指定して接続を行う必要がある。**

という現状が影響している。

現在のネットワークアプリケーションには上記の様な問題を解決する為、独自にサーバホストの発見や切替え機構を持っているものがある。また、Dynamic Host Configuration Protocol (DHCP)[65] や Boot Strap Protocol (BOOTP)[66] 等のホスト設定用のプロトコルを利用してサーバホストのアドレスを得ることもできる。

ここでは、この様な現在のネットワークアプリケーションのサーバ発見、サーバ切替えについて考察する。

### 2.4.1 サーバ発見

現在のアプリケーションのサーバ発見方法には大きく 2 種類ある。

- アプリケーション依存のサーバ発見 (NIS[1]等)。
- DHCP や BOOTP を使ったサーバの発見 (DNS 等)。

前者の方法の利点は、サーバとの接続が切れた時にサーバ発見を行なうので動的な設定ができる点である。欠点はアプリケーション自身にサーバ発見機能を実装する必要があり、アプリケーションプログラムの設計/実装が難しくなることである。

逆に、後者の方法の利点は、サーバ発見の為のソフトウェアと実際に利用するアプリケーションをわけることができるため、アプリケーションプログラマの負担を減らすことができること、欠点は、動的なサーバ発見が難しいことである。

また、サーバ発見の手法はそのアルゴリズムにより、次のような 2 つにわけることもできる。

- ブロードキャスト等を使ったサーバの発見。
- サーバクライアントモデルを用いたデータベース検索によるサーバの発見。

前者は、NIS、DHCP 等で利用されている方法で、初期設定を必要としない。また、動的な発見も行ないやすい。後者は予め、データベースを持っているサーバをクライアントに登録しておく必要があり、更に、静的な情報検索にしか利用できないことが多い。

現在のアプリケーションの多くは自動サーバ発見は行なっておらず、ユーザが一つ一つ設定ファイルに記述しているのが現状である。機構的には、現在、DHCP 等により、ネームサーバ等のリストを取得し、それを設定した上でアプリケーションを使うのが一番現実的である。

---

[1] Sun Microsystems 社によって開発されたシステムで `/etc/hosts`、`/etc/passwd` 等の設定ファイルを一括管理するためのシステム

### 2.4.2　サーバ切替え

通常、クライアントはサーバ切替えを次のいずれかの方法によって行なう。

- 予め設定されている代替サーバに自動的に切替える。
- ユーザ自身が明示的にサーバを変更する。

前者の方法は予め幾つかのサーバを設定ファイルに指定しておき、はじめのサーバへの接続がうまくいかないと次のサーバを試す、という動作をするのが普通である。後者はユーザがアプリケーションを使用していて (使用しようとして)、必要に応じてサーバを変更する。

前者の方式の利点は一度指定してしまえばユーザはサーバの切替えに関与しなくて済むこと、後者の利点は必要に応じてサーバを変更することができることである。

### 2.4.3　現在の方法の問題点のまとめ

現在のサーバ発見/切替え機構は事実上、静的な情報によって行なわれている。静的な情報を使ってサーバ切替えを行なえなくなると、人の手を介してサーバの切替えを行なうことになる。

また、一番近いサーバを使うことなどは考えられておらず、この様な使い方をする場合はユーザの手を介さなければならない。移動体通信等の場合も殆どの場面においてサーバ発見を自動で行なうことはできず、ユーザが行った先の管理者に聞いて、クライアントの設定ファイルを変更する作業が必要になる。

## 2.5　現在のネットワークアーキテクチャに関する考察

本節では実際のネットワークアーキテクチャを踏まえながら現在の一般的なネットワークアーキテクチャについて考察を加える。

### 2.5.1　階層モデルの利点

階層型の計算機ネットワークアーキテクチャは、現在、殆どの計算機ネットワークで採用されている。これは、一般に次のような利点があるためである。

- 機能別の各層を個別に設計する事が可能である
- 各層を個別に実装することが可能である
- テストが容易である
- ある一つの層 (機能) だけを取り換えることが、全ての層を一つとして設計した時より、容易である

- 設計、実装の技術が確立しており、容易に採用できる

これらの長所を見た時に気づくことは、これらは、ネットワークアーキテクチャを設計する者にとっての利点であり、計算機ネットワークの利用者である一般ユーザにとっての利点とはなり得ないということである。

### 2.5.2 Peer-to-Peer 通信に関する考察

階層型のネットワークアーキテクチャでは Peer-to-Peer の通信を提供している。そのため、通信相手が誰なのか、また、その相手はどこにいるのかをアプリケーションが知らなければ通信を開始することができない。

しかし、クライアントを利用しているユーザから見れば、サーバがどのホストかは一般的に関係なく、目的とするサービスさえ手にはいれば良い。また、移動環境などにおいても、今、相手はどのホストを利用しているのか、そのホストはどこに存在するか、等ということに気をとられることなく相手とまたは相手のホストと通信できるような環境が理想である。しかし、現在のネットワークアーキテクチャではこの様なアプリケーションの構築を許しておらず、予め相手との接続を行う前に、手動または自動で相手のホストに対する情報を取得し、その上で相手との接続を確立しなければならない。

この様な機能を実現するために、XNS プロトコル群では Clearinghouse [67, 68, 69] を Internet プロトコル群では Domain Name System (DNS)[70, 71] を用いて相手のホストのアドレス等の検索を行っている。

相手に関する情報を得た後にどれくらいの量の通信を行うかにもよるが、この様な検索作業は、通信量が少ない場合は一般に大きなオーバヘッドとなる。また、接続が確立した後も、相手が場所を移ったり、利用していたサーバが故障を起こした場合等はもう一度始めからやり直さなければならないため、ここでもオーバヘッドを生じることになる。

さらに、検索を行ってサーバのアドレスを得る場合には、全ての検索について同じサーバのアドレスを値として返すことが多いため、一つのホストにアクセスが集中することが多く負荷分散が難しい、故障に弱い等の問題点がある。

### 2.5.3 アドレッシングに関する考察

現在のネットワークプロトコル群では、ネットワークアプリケーションが使う (相手や自分を識別するために使う) アドレスは、おおよそ次の様な形になっている。

ネットワーク識別子 + ホスト識別子 + ポート (ソケット) 識別子

これらは、それぞれ各層における識別子にほかならない。

このような識別子がアプリケーションの接続点を指定するものとして利用されているため、ネットワークアプリケーションは相手を特定するために、全ての識別子を予め知らなければならない。しかし、計算機ネットワークが一般に広く利用されるようになってきた

現在、ユーザ層も厚くなり、このようなネットワークアーキテクチャ全体を理解した上で全てを指定することをユーザに求めるのは難しくなってきている。

現在は、この問題をディレクトリサービスを利用することによって解決しているが、このままでは動的な指定は望めない。

### 2.5.4　経路制御に関する考察

現在使用されているネットワークプロトコル群では、本章で見てきたように、ほぼ、ある一つの層で経路制御を行っている。また、更にその層のほとんどはパケット交換を行っている層である。

パケット交換の利点は、

- ルータが経路制御のためにコネクション毎の状態を持たなくても良い。
- 多重化が簡単である (自然な多重化ができる)。
- 一つのパケットを経路制御するために必要な資源が既知である。

等である。これらの多くは、大規模なネットワークを構築するにあったって非常に重要な要素となる。しかし、ネットワーク層による経路制御は、経路制御にホストではない「ネットワーク上の何か」を指定したい時に、その実現が非常に難しいものとなる。

現在の経路制御機構を変更することなく、このような機能を実現するための方法として、現在、利用されているのはアプリケーションゲートウェイやアプリケーションルータである。実際に電子メールなどでは、電子メールアドレスを相手を指定する識別子として用いる為に、この様な実装がなされており、実用化されている。しかし、設定が複雑である、経路の最適化ができないなど多くの問題を抱えている。

## 2.6　名前空間と経路制御

本節では、計算機ネットワークにおける「名前」の持つ意味及びその扱いについて議論する。ここで言う名前はある実体 (ホストなど) や終点 (ソケットなど) を示す印の事である。すなわち、ホスト名は計算機自体を示す名前、ホストアドレスは計算機が接続されている位置を示す名前となる。

### 2.6.1　現在の計算機ネットワークアーキテクチャに関する考察

現在のネットワークアーキテクチャでは各層に一つ以上の名前を持っている。ここで、複数の名前を持っている層は実は二つ以上の機能を持っている層であると言うことができる。これを図で表すと図 2.1 の様になる。

ここでそれぞれの名前 (ID) は以下のような目的で用いられている。

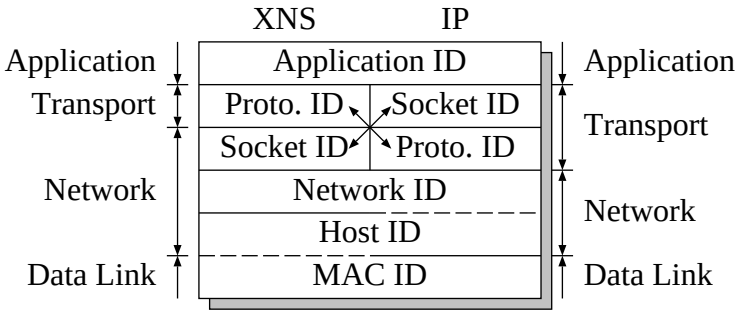



図 2.1: 現在の階層構造と名前

**Application ID**
アプリケーションプロトコルを指定する

**Socket ID**
アプリケーションが使ってるソケットを指定する

**Protocol ID**
トランスポート層プロトコルを指定する

**Network ID**
相手のホストが接続されている物理ネットワークまでパケットを経路制御する

**Host ID**
一つの物理ネットワーク上でホストを識別する

**MAC ID**
物理ネットワークに依存するインターフェイスを識別する

ここで、各層においては各層が持つ機能 (ストリーム通信の実現等) の他にこれらの名前を用いて対応付けを行っている事に注意しなければならない。例えば、Internet プロトコル群においてはインターフェイス層で Host ID から MAC ID に対応付けを行っているし、IP 層では Network ID や Host ID を用いて次にパケットを渡すべき相手を特定している。この時、各層における対応付けは必ず上下関係にある。つまり、各層を通る度に ID は一つずつしたの層のものに対応付けされていき、最終的に MAC アドレスが求められるのである。この様な関係を図 2.2に示す。

図 2.2を見てもわかるように現在のネットワークアーキテクチャにおける層間の対応付けは必ず、別の名前空間に対して行われる。つまり、各層に独立した名前空間を持っているという事ができる。

この図を見てわかるように現在のネットワークアーキテクチャでは上の層から間の層を経ずに下の層が利用している名前へ対応を行う事はできない。これは各層における対応表

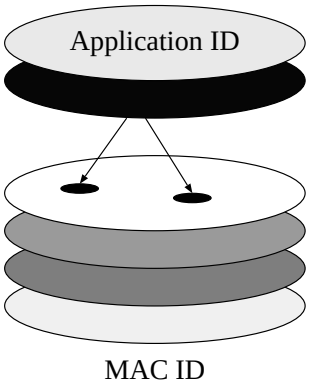



図 2.2: 各層における対応付け

が自分が利用している名前を一つ下の層に対応付ける為にしか使えないものだからである。この様な表を利用している理由はネットワークアーキテクチャの階層構造にある。

また、対応表が用意されていない層に対しては、予めユーザが指定しておかなければならない。このため、ユーザは相手のアプリケーションを指定する為に、相手の計算機のホスト ID 等を予め知る必要がある。

この様な一連の動作により、相手に届いたパケットには全ての層における ID が含まれており、各階層においてスムーズに処理を行う事ができる。

ここで、ネットワーク層での対応表、すなわち経路制御表に注目する。経路制御表では基本的に、

$$Network\ ID \rightarrow Host\ ID$$

という対応を行っている。これを繰り返す事により、パケットは最後の物理ネットワークまで到達する。

このことは次の様な二つの意味を持つ。

1. 現在の計算機ネットワークでは通信相手のネットワーク ID が予めわからないとパケットを中継するべき次のホストを特定する事ができない。

2. ネットワーク ID 以外のものを使った経路制御ができない。

また、最後の物理ネットワークに到達した後、ルータはホスト ID を使って相手のホストを特定する。実際には、

$$Host\ ID \rightarrow MAC\ ID$$

という対応付けを行う事により、最終到達ホストのインターフェイスを特定している。

ここでもホスト ID がわからない場合は、MAC ID を知る事はできず、パケットにホスト ID が含まれている必要がある。

### 2.6.2 名前空間の共有

前節で述べたような各階層での名前が必要である、という問題を解決する一つの方法として、上位に位置する層では、その層で使用している名前から下位に位置する全ての層で使われている名前への対応表を持つ方法が考えられる (図 2.3参照)。

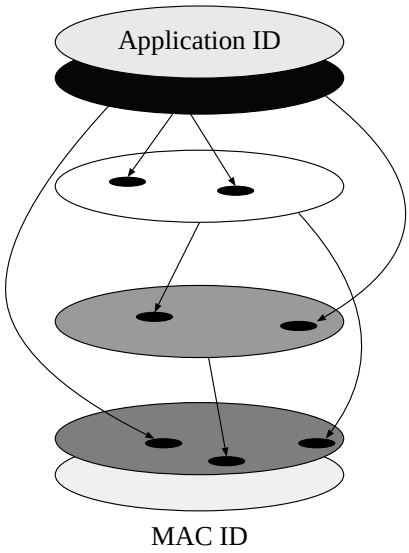



図 2.3: 層を越えた対応付け

この方法では対応表が、これまでの

$$< N > ID \rightarrow < N-1 > ID \tag{2.1}$$

から、

$$< N > ID \rightarrow \{< N-1 > ID, < N-2 > ID, < N-3 > ID...\} \tag{2.2}$$

となる。

もう一つの方法として 2.2式の左辺も限定せずに、

$$\{< N1 > ID, < N2 > ID, < N3 > ID...\} \rightarrow \{< N1 > ID, < N2 > ID, < N3 > ID...\} \tag{2.3}$$

の様に階層をなくしてしまう方法がある。この方法ではある階層における名前を規定していない、という特徴がある。この方法を図で表すと図 2.4の様になる。

この図では名前空間が重なることができる事に注意が必要である。つまり、XNS プロトコル群におけるホストアドレスと MAC アドレスの様な関係を自然に実現する事ができる。

ここまでに前節で述べた 1 の様な問題点を解決する為に 2 つの解決案を示した。しかし、まだ 2 の問題については議論していない。ここでは名前空間の経路制御における意味を考えながら 2 の問題について議論する。

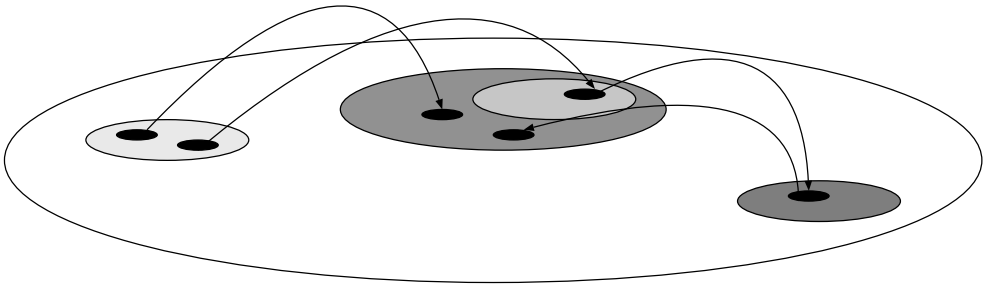


図 2.4: 層をなくした名前空間における対応付け

現在のネットワークアーキテクチャでは経路制御は経路制御表による対応付けの繰り返しによって実現されている事は既に述べた。ここで、現在の経路制御表が 1 対 1 の対応付けしかできない事も既に述べた。更に、経路制御表において左辺は常にネットワーク ID である。この為、経路制御はネットワーク ID だけで行われている。

ここで、ネットワーク層以下の層が最終的に求めなければならないものはこれまで述べてきた事より、MAC アドレスである事がわかる。また、現在の計算機ネットワークアーキテクチャではネットワーク層においてのみ経路制御を行っているがこれに関する理由は、ネットワーク層における経路制御だけで相手まで到達する経路を得る為には十分だったからである。

しかし、近年の状況からこの様な経路制御では現在のネットワークアプリケーションを支援するのは難しくなってきた。つまり、ネットワーク ID のみによる経路制御では不十分になってきた。この様な状況を 2.6.2節の様な方法で支援する為には、経路制御を全ての層で行う必要がある。これにより、様々な名前を使って経路制御を行う事ができるようになる。しかし、この方法では全ての層に経路制御を行うような機構を組み込まなければならない、一度上位層までパケットを挙げなければならず効率が悪いなどの問題がある。

この様な理由により、これからの計算機ネットワークアーキテクチャでは 2.1式や 2.2式の様な経路制御表では不十分な事がわかる。よって 2.3式の様な名前空間を用いるべきである。

## 2.7　サービス名による経路制御機構

### 2.7.1　サービス名前による経路制御の概念

これまで述べて来たように、現在のサーバ発見/切替え機構はアプリケーションとして実装されている。このため、クライアントはホストの移動等に対応することができなかった。

このような現状を考慮して、ここでは経路制御部でのサーバ発見/切替え機構を提案する。

現在の、ネットワークアーキテクチャでは一般にネットワーク番号を基に経路制御を行なっている。ここでネットワーク番号の代わりに、サービス名による経路制御を行なおうというのが基本的な考え方である。

サービス名による経路制御を導入する場合、本章の最初で述べたような、現在の経路制御を特徴付けている以下のような点について議論しなければならない。

- 交換方式
- アドレッシング
- 経路制御方式と経路制御表
- 経路情報交換プロトコル

このうち、交換方式については基本的にパケット型の交換を前提とする。これは現在の交換方式がパケット交換を用いたものが殆どで、また、ネットワークメディアもパケット (フレーム) 型の通信を提供しているものが殆どだからである。但し、他の交換方式を用いた場合にも以降に述べることを、接続要求時等に当てはめる事ができる。

また、アドレッシング (名前空間)、経路制御方式と経路制御表、経路情報交換プロトコルについては個別に節を改めて議論することにし、ここでは動作概要について述べる。

この方式では図 2.5で示すように、各ルータやホストはサービス名を鍵とした経路制御表を持っており、これによってパケットの経路制御を行なう。この例では、Host 2 が service B のサービスを受けようとする時、パケットの行き先として service B を指定してパケットを送信する。すると、Network B、Router A、Netowrk C、Router B、Network D を介して Host 3 からサービスを受けることができる。また、Default Route の概念もあり、Host 1 が service A のサービスを受けようとする時、Host 1 は Default である Router A に service A を行き先として指定したパケットを送信する。これにより、Router A はこのパケットを Host 2 に中継し、Host 1 は Host 2 からサービスを受けることができる。

この方式の特徴は、次の様なものである。

- あるサービスを受けようとしているホストがパケットを送信する時点では、相手がどのホストであるか知らなくてもよい。これによって、サーバ発見の手続きを予め踏むことなく、サーバとの通信が可能となり、それにかかるオーバヘッドがなくなる。
- 経路制御表のエントリを変更することにより、無理無くサーバを変更することができる。すなわち、クライアントがサーバの故障などに依存してサーバを切替える必要が無い。

これらの特徴が前章で述べた様な問題点を解決していることは明らかである。

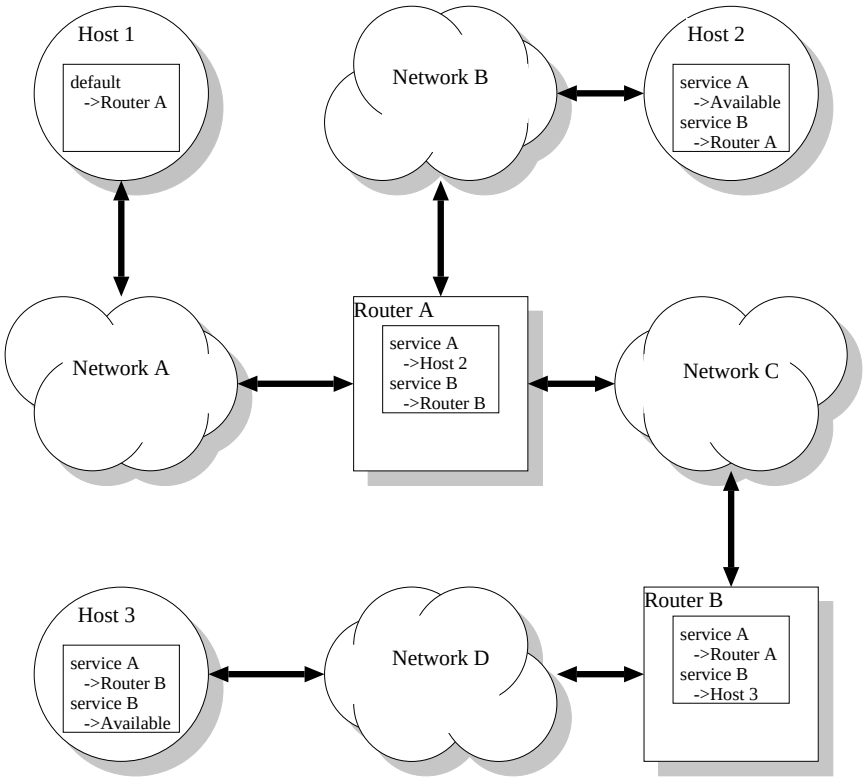



図 2.5: サービス名による経路制御の概念

### 2.7.2 名前空間

本方式ではサービスと言うこれまでアプリケーション層にしかなかった概念をネットワーク層に導入している。ネットワーク層は指定された相手によって、その相手までのパケットの到達性を実現する層である。ここで、ネットワーク層における「相手」というものがこれまでは「ホスト (計算機)」に限定されていた。しかし、前に見て来た様に実際にアプリケーションが指定する相手は「ホスト (計算機)」であることは少なくなってきている。この様な状況で「ホスト (計算機)」だけに相手を限定して経路制御を行うことは不自然である。

今回、ネットワーク層に新たに「相手」としてサービス名を指定することができるようにネットワーク層を拡張した。これによってネットワーク層で使われる名前空間は「ホスト名 + サービス名」になった。しかし、これは 2 つの名前空間がネットワーク層で使われる様になったのではなく、新たに「ホスト名 + サービス名」という名前空間が作られ、それがネットワーク層の新しい名前空間として採用された、とみなした方が良い。何故ならば、ネットワーク層において経路制御表は 2 つあるわけではなく、あくまでも一つであり、どちらを使っても経路制御ができるからである。言い方を変えると、パケットの「受信者アドレス」としてこれまで使用されていた部分が新たに他のものに変る、もしくは、「受信者アドレス」の他に「サービス名」のフィールドが新たに追加されるのではなく、「受信者アドレス」の意味が拡張される、と考えているからである。

また、ここではこの 2 つの名前群を一つの名前空間としているが更にその空間を広げることもできる。実際に 2.7.3節で更にこの空間を広げる例を示す。

但し、上で述べていることは全ての名前を必ずしも同列の識別子として実装しなければならないということではない。名前空間に構造を持たせることと名前空間を共有することは別の問題であることに気をつけなければならない。この議論は、IP アドレスが一つの名前空間にユニキャストアドレスとマルチキャストアドレスを設けているのと同じである。この様な意味では IP アドレスも二つの名前群を一つの名前空間に置いていると言うことができる。

### 2.7.3 経路制御方式と経路制御表

2.7.1節で述べたようにサービス名における経路制御機構においては対応表エントリとして以下のようなものを持つ。

{ サービス名、ホスト名 } → パケットを次に渡すべきルータ

ここで、右辺と左辺は同一名前空間とする。

ルータはこの様な経路制御に基づいてパケットを中継する。この経路制御表の検索は 2.7.2節で述べたように、鍵は結局、一つの名前空間なので、サービス名かホスト名かによって二度検索を行う必要はない。もちろん相手を表す為の名前空間がサービス名かホスト名か

によって構造化されている場合、この構造を利用して検索を高速化することもできる。むしろ、この様な高速化は行うことができるのであれば進んで行うべきである。

この経路制御表で「パケットを次に渡すべきルータ」はルータを表す名前であればなんでも良い。現在のネットワークアーキテクチャでは一般的にホストアドレスを用いているが、本経路制御表では例えば、ホストの物理ネットワークにおける MAC アドレスをそのまま使っても問題はない。この場合、2.7.2節で述べた名前空間は更に広がり、同一空間に MAC アドレスを持つことになる。ここで、述べていることは IP プロトコル群に当てはめれば IP の経路制御表と ARP テーブル[2]を同じ表の中に入れたことにあたる。IP で ARP テーブルと IP 経路制御表をまとめなかったのはこの二つが階層が異なるものであり、経路制御表に IP アドレス以外の異質なものを入れなければならなかったからである。(歴史的な理由もある。) しかし、これは本方式との考え方の違いで、本方式では全ての名前を同じ名前空間にあるものとして扱うため、一般性は失われない。

### 2.7.4　本方式における問題点

この方式を導入する場合、幾つかの問題点がある。これらはネットワークアーキテクチャにおける致命的な問題ではないが、実際に現在のネットワークで利用しようとした時に問題になる。

本方式を既存の計算機ネットワークに導入しようとした場合、一度に全てのホストのネットワークソフトウェアを変更することは不可能である。しかし、ルータやホストがこの機能を実現していない場合、経路制御が途中で途絶えてしまうことになる。

図 2.6に示したのは Router B が本方式を実装していない場合の図である。このようなネットワークの場合、Host 1、Host 2 は service B のサービスを受けようとしてパケットを送信するが、Router 2 がこのような経路制御を実装していない為、パケットをそれ以上経路制御できなくなり、結果として Host 1、Host 2 は service B のサービスを受けることができなくなる。これは Host 3 が service A のサービスを受けようとする場合も同じである。

この様な問題は経路制御を行っているプロトコルに新しい機能を持ち込もうとした時には常に起こる問題であり、通常は現在の M-Bone[3]の実装の様に、トンネリング等により解決される。

他の問題点として経路情報の管理をどのようにして行なうか、という問題がある。経路情報の交換は経路の最適化を行なう上で重要であり、エンドホストでもこのようなプロトコルを実装する必要がある。

これまでのネットワークアーキテクチャにおいて経路制御を行なう場合、エンドホストでは同じサブネット上のホストにパケットを送信する時はそのアドレスから同じサブネッ

[2]IP アドレスから Ethernet の MAC アドレスに対応付けを行う表。ARP: Address Resolution Protocol

[3]現在の Internet におけるマルチキャストバックボーンの名前。トンネリングを利用して仮想バックボーンを形成している。

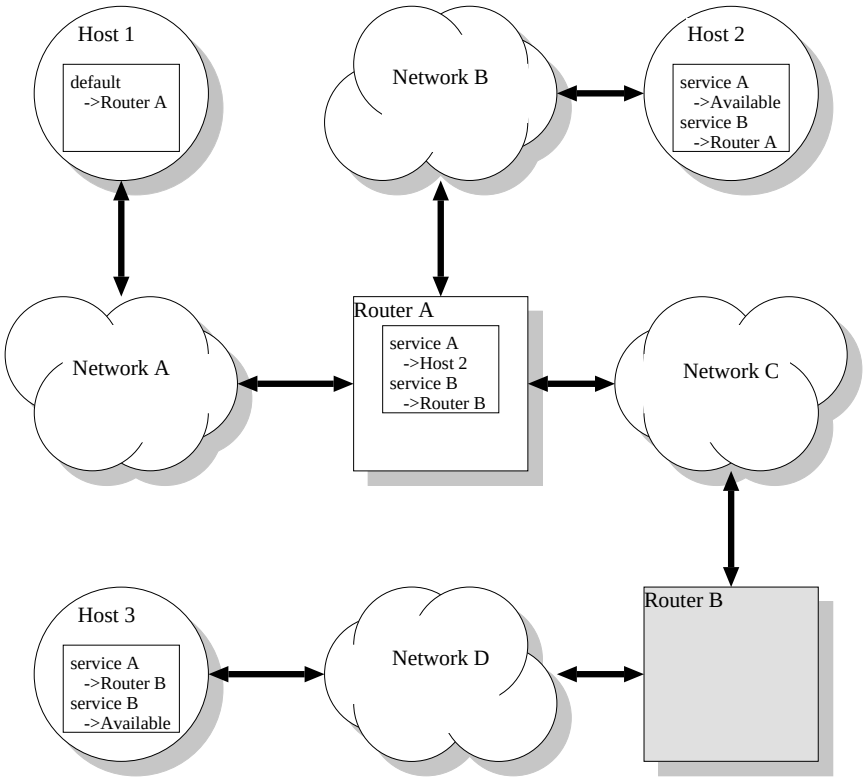



図 2.6: 混在したシステム

ト上に相手のホストが存在していることを知ることができ、直接相手のホストにパケットを送信することができた。しかし、本方式を使った場合、サーバが同じサブネット上にあるかどうかをそのサービス名から知ることはできない為、経路情報交換プロトコルのエンドホスト上での動作が不可欠になる。

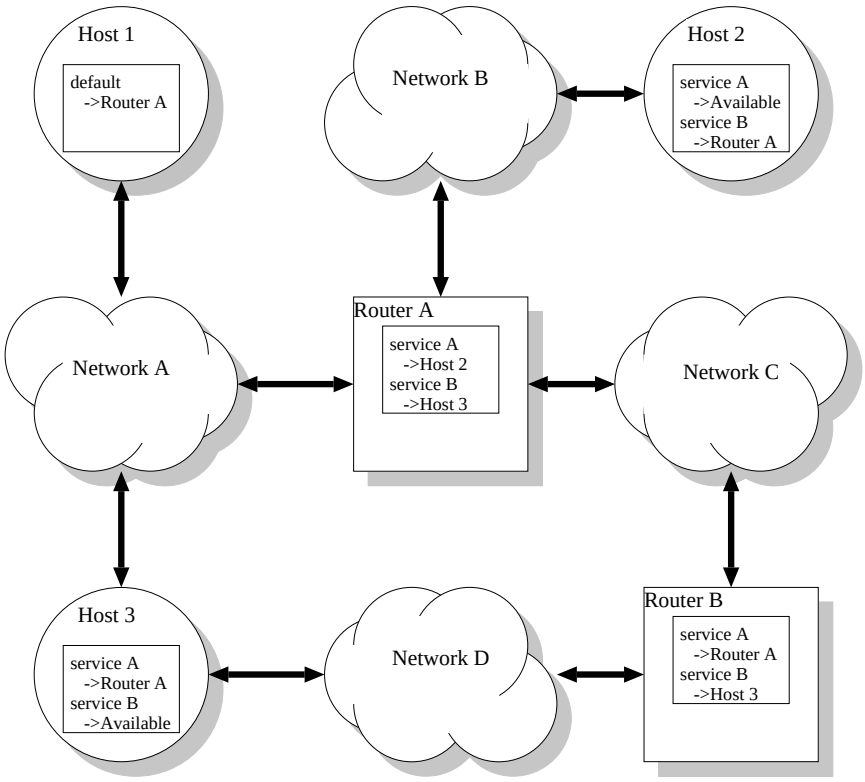



図 2.7: 経路の最適化ができていない例

経路情報の交換をせず、デフォルトルートに頼った場合の経路の最適化ができていない例を図 2.7に示す。この例では、Host 1 がデフォルトの経路に頼っている為、service B のサービスを受けようとした場合、service B のサーバである Host 3 に直接到達できるにも関わらず、はじめに Router A にパケットを送信する。ここで、Router A はパケットを経路情報に基づいて Host 3 に中継する。Host 3 からの返答は直接、Host 1 に到達し、結果として Host 1 が service B のサービスを受ける場合には三角形の経路を通ることになる。

### 2.7.5　経路情報交換プロトコル

サービス名による経路制御を行う場合、サーバがサービスを行う範囲において経路情報を伝播させなければならない。この場合には、通常の経路情報交換プロトコルで使用されているアルゴリズムである、ディスタンスベクタ型アルゴリズム、リンクステート型アルゴリズムの二つのアルゴリズムをそのまま適用することができる。本章ではそれぞれのア

ルゴリズムに付いて個別に議論する。

**●ディスタンスベクタ型アルゴリズム**

サービス名による経路制御においてディスタンスベクタ型のアルゴリズムを用いて経路情報を交換する場合、これまでの経路制御と同じように動作する。Host A が service A の Host B が service B のサーバである図を図 2.8に示す。

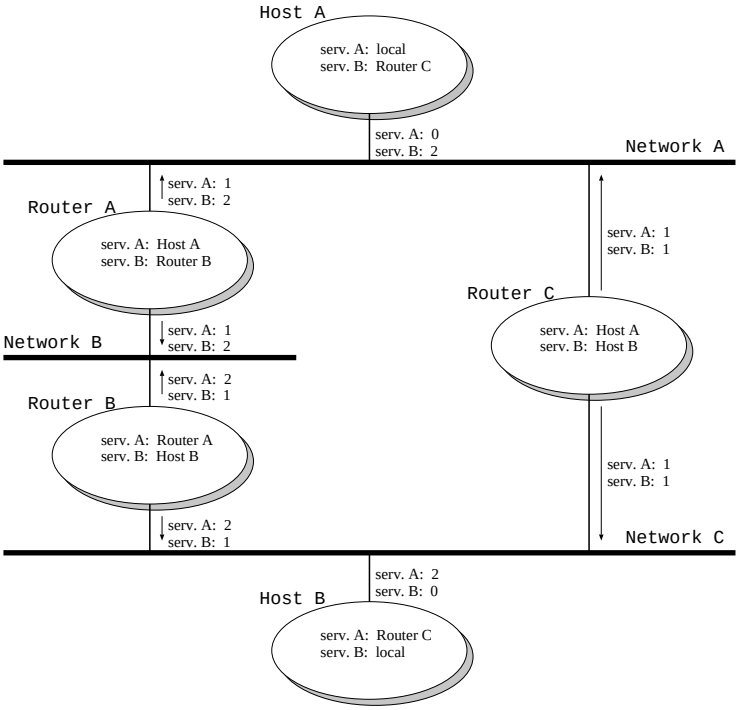


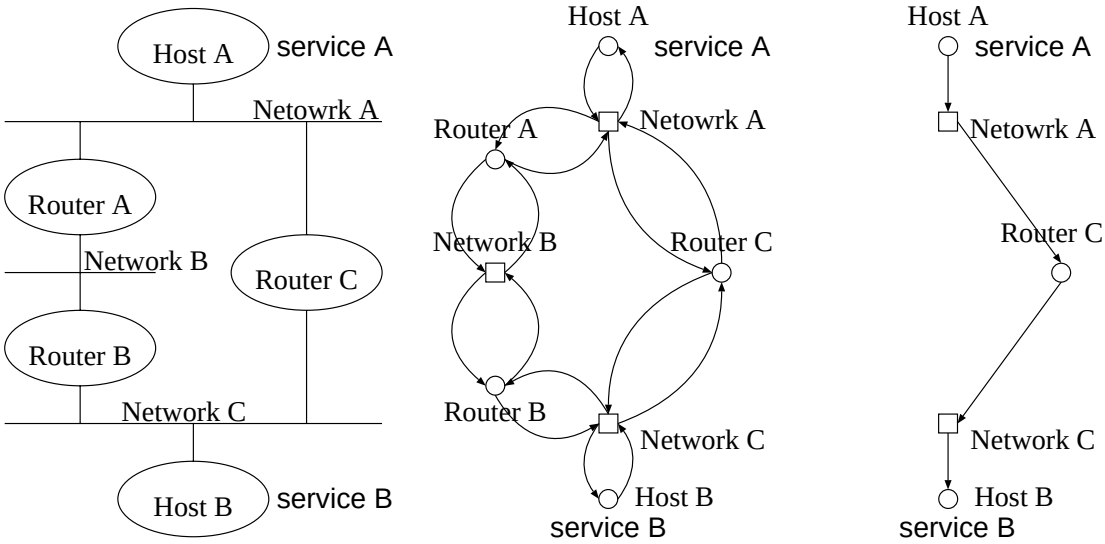

図 2.8: ディスタンスベクタ型アルゴリズム

ここで、気付くことは通常の経路制御ではルータのみが経路をアナウンスしていたのに対して図 2.8ではホストも経路をアナウンスしている点である。ホストアドレスによる経路制御の場合は実はネットワークに対する経路をアナウンスしていたのに対して、サービス名による経路制御ではサービスをしているホストからアナウンスが必要な為、このような違いが見られる。

また、この様な違いの他に、サービス名による経路制御の場合、通常は一つのサーバがサービスする範囲はある程度限定されている。このようなことを考慮してサーバがサービスする範囲にだけ経路が広がるような工夫をしなければならない。このためにはエリア識別子を用いて計算機ネットワークを幾つかのエリアに分けたり、ルータによって経路制御を伝播しない様に経路情報を止めたりしなければならない。

**●リンクステート型アルゴリズム**

ディスタンスベクタ型のアルゴリズムを用いて経路情報を交換する場合と同様にリンクステート型アルゴリズムを用いた場合も、これまでの経路制御と同じように動作する。Host A が service A の Host B が service B のサーバである図を図 2.9に示す。




(a) Network　　(b) Directed Graph　　(c) SPF tree for Host A

図 2.9: リンクステート型アルゴリズム

通常の経路制御におけるリンクステート型アルゴリズムの動作と図 2.9を比べた場合に気付くことはそれぞれの図の (c) だけが異なることである。これは、(b) から (c) を作る時にサービスだけに注目した結果である。サービス名による経路制御ではサーバ以外への到達性は無視することができ、その結果最短木を作る時に、サーバまでのパス上にないホストやネットワークは意味のないものとして省略することができる。

また、上記のことは (b) までの処理をアドレスによるこれまでの経路制御と共有することができると言うもう一つ意味を持つ。つまり、リンクステート型のアルゴリズムを使った経路情報交換プロトコルを使った経路情報管理をしている既存の計算機ネットワークにサービス名による経路制御を導入した場合、経路情報交換プロトコルの一部を変更するだけでサービス名による経路制御表も同時に管理することができる。

## 2.8　対応付けと経路制御

ここでは対応付けを使って、現在のネットワークアーキテクチャと前章で導入したサービス名による経路制御機構の混在に起因する問題を解決する。この方法の基本的な考え方は VIP のホスト識別子から位置識別子への対応付けにヒントを得たもので、ここではサービス名から位置識別子への対応付けを行なっている。

実際には、サービス名から位置識別子への対応付けを行うということは、対応付けの回数が一度多い為、これまで述べてきたようなサービス名による経路制御以上に多くのことができる。しかし、ここではサービス名による経路制御を現在に計算機ネットワーク上で実現する為の手段としてだけ対応付けを用いている。

### 2.8.1 サービス名から位置識別子への対応付け

現在のネットワークではそのアドレス (位置識別子) によって経路制御が行なわれている。よって、この様な環境の中でサービス指向の経路制御を行なう為には、サービス名を位置識別子に対応付けすれば良い。ホストはパケットを送信する時、そのサービス名を現在使用されているホストアドレスに対応付けを行なってパケットを送信する。これにより、サービス名による経路制御を実装していないホストではこれまで通りアドレスを使った経路制御を行なうことができ、事実上のトンネリング[4]が可能になる。

対応付けによる問題解決では、サービス名による経路制御を実装している各ホストやルータに次のようなエントリを持つ対応表を用意しておく。

サービス名 → サーバまたは次にパケットを渡すべきルータのアドレス

各ホストやルータはパケットを送信/中継する際、サービス名をサーバアドレスへ対応付けし、パケットを送信する。これにより、実際のネットワーク上に出て行くパケットは全てこれまで使われていたようなアドレスを持つことになる。ここで、このパケットがルータに到達した際、ルータが別の対応付けを持っていた場合は、そこでもう一度対応付けがやり直され、ルータは新しいアドレスをパケットに設定して中継する。

図 2.10は図 2.6の例に対応付けを導入した例である。ここで、Host 1 が service B のサービスを利用することを考える。Host 1 は、まず、service B を利用する為にサービス名-アドレス対応表を検索する。しかし、service B に対するエントリが見つからないのでデフォルト値である Router A への対応付けを行なう。ここで、Host 1 から送信されるパケットは、

| | |
|---|---|
| サービス名: | service B |
| 相手のホストのアドレス: | Router A |

となる。このパケットは通常の経路制御表 (図の太い枠の中の表) によって Router A に送信される。Router A はこのパケットを受け取るとルーティングを行なう前にもう一度対応付けを行なう。ここで、Router A のサービス名-アドレス対応表では service B のサービスを行なっているホストは Host 3 になっているので対応付けがやり直され、パケットは、

| | |
|---|---|
| サービス名: | service B |
| 相手のホストのアドレス: | Host 3 |

[4] 他のプロトコルを利用してパケットを相手まで到達させる技術

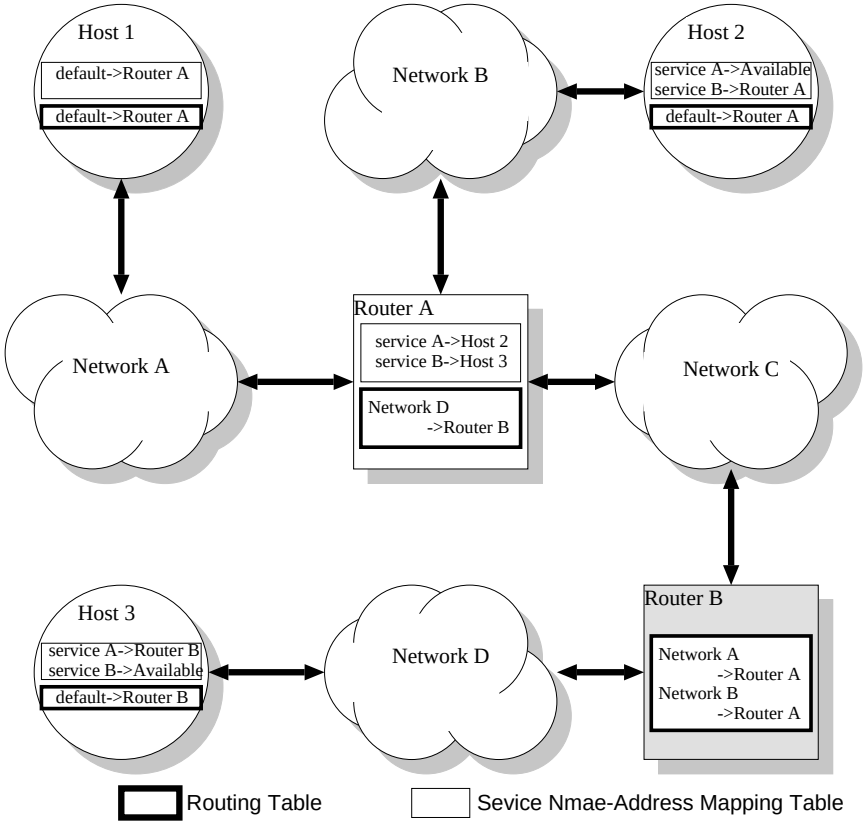



図 2.10: 対応付けを用いた解決

となる。その後、Router A は経路制御表に従い、パケットを Router B に中継する。Router B ではパケットの中に含まれる相手のホストのアドレスを見てパケットを Host 3 に中継し、Host 1 が送信したパケットは Host 3 に到達することになる。Host 3 からの返答は通常通り、Host 1 のアドレスを指定されるので Router B、Router A を通って Host 1 に届けられる。

このように対応付けを用いることにより、サービス名による経路制御を実装していないホストが混在する環境でもサービス名によって相手を指定することができるようになる。

### 2.8.2 考察

本方式では、ルータ毎に対応付けを行なうことにより、サービス名による経路制御を実装していないホストが混在する環境に対応した。これは実際には経路表を 2 度参照していることになるため、サービス名による経路制御を直接実装するのに比べいくらかのオーバヘッドが生まれる。また、対応付け導入することにより、対応表が一つ余分に必要になりその分のメモリ消費も負荷になる。

しかし、この方法を用いることによる利点もある。経路制御に使われるホストアドレスとサービス名とが分離されており、実際の経路制御に使われる識別子はホストアドレスだけなのでネットワークソフトウェアの経路制御部分が簡略化できる。また、その分、高速化も期待できる。更に、エンドホストにおいてこれまでのソフトウェアをそのまま相手のホストを指定することによって、利用することも可能である。

## 2.9 対応情報交換プロトコル

対応表を管理する為には大きく次の様な 2 つの手法が考えられる。

- 人手による管理
- 管理プログラムによる自動管理

事実上、前者による管理ではデフォルトを設定して、他の幾つかの重要なものに関して管理を行なうのが限界であり、今日の Internet の中でこの手法による管理を行なうのは非現実的である。そこで、普通は後者の手法をとることになる。本節ではこのような管理プログラムによる対応表の自動管理に対して考察を加える。

現在、経路制御情報交換プロトコルで使われているアルゴリズムは大きく 2 つある。これらのアルゴリズムは、サービス名による経路制御方式におけるサービス名に基づく経路制御表の管理にもそのまま用いることができる。しかし、対応付けを行なった場合はこの限りではない。ここではそれぞれのアルゴリズムを用いた場合に関して、どのように対応表を管理しなければならないかについて議論する。

また、サービス名による経路制御の場合、サービス範囲を狭い範囲に限定して運用されることが予想される。この様な場合、データベースを用いた経路情報のアナウンスも現実的である。

### 2.9.1　ディスタンスベクタ型アルゴリズム

ディスタンスベクタ型のアルゴリズムでは通常、定期的に自分が接続されているネットワークに対して自分が到達できるネットワークに関する情報をブロードキャストすることによって経路表を更新している。しかし、対応付けを用いたサービス名による経路制御方式においてはサービス名による経路制御機構を実装していないホストが存在する為、通常通り、ネットワークデバイスが提供するブロードキャスト機能を用いて経路情報を交換したのでは情報が途中で途絶えてしまい、経路表 (対応表) を正しく管理することができない。

このような問題点を解決する方法として次のような 2 つの方法が考えられる。

- アプリケーションによる経路情報 (対応情報) の中継
- サービス名による経路制御機構を実装していないホストのトンネリング

この 2 つの方法はディスタンスベクタ型アルゴリズムをそのまま、用いた方法であり、ブロードキャストをする為に途中で情報が途絶えるという問題を解決したものである。

前者の方法はサービス名による経路制御機構を実装していないホストでも経路制御 (対応表制御) プロトコルだけは動作させ、実際の経路制御 (対応付け) は行なわないまでも経路情報 (対応情報) の中継だけは行なう方法である。この方法の利点は全てのルータでこのプロトコルを動かすことにより、メトリック値などを正しく伝えることができること、予め設定する事項が何もなく、管理が楽であることである。逆に欠点は、全てのルータがこのプロトコルを実装しなければならないことである。

後者の方法はブロードキャストをやめ、予め設定されているルータに対してユニキャストをすることにより、間にあるサービス名による経路制御機構を実装していないホストをトンネリングしてしまう方法である。ディスタンスベクタ型アルゴリズムではブロードキャストを用いることは本質ではなく、当然、ユニキャストを用いて経路情報を交換することができる。通常ブロードキャストを用いるのは一度の送信で全てのホストが受信できるため効率が良いこと、ルータ発見をする必要が無いこと等に起因する。

これらの 2 つの方法が排他的でないことは明らかである。よって、実際にディスタンスベクタ型アルゴリズムを使って対応表の管理をする場合、はじめのうちはサービス名による経路制御機構を実装しているホストが少ない為、後者の方法を用い、徐々に前者の方法に移っていくのが望ましい。また、後者の方法は間にサービス名による経路制御機構を実装していないホストが多く存在する場合に良く動作し、前者の方法は限られた範囲で良く動作するので、Wide Area Network (WAN) では後者を、Local Area Network (LAN) では前者を用いると良い。

### 2.9.2 リンクステート型アルゴリズム

リンクステート型アルゴリズムではディスタンスベクタ型アルゴリズムを用いた場合と比較して、より良く対応表を管理することができる。

実際にリンクステート型アルゴリズムを用いて対応表を管理する為には、現在使われている様なルータ発見プロトコルがブロードキャストまたはマルチキャストを利用しておりうまく動作しない為、ディスタンスベクタ型アルゴリズムの時と同様の手法によって間のサービス名による経路制御機構を実装していないホストをトンネリングしなければならない。

しかし、これまでの経路制御表の管理の場合と同じく、サーバの故障などにより短い時間で対応することができる等の利点がある。

### 2.9.3 その他の方法 1: サーバデータベースによる方法

サービス名に経路制御ではサービスを行う範囲をある程度限定することが考えられるため、これまで述べて来たような 2 つのアルゴリズム加えてデータベースによる管理をすることができる。図 2.11にその概念を示す。もちろん、ここでアナウンスサーバはルータである必要はない。

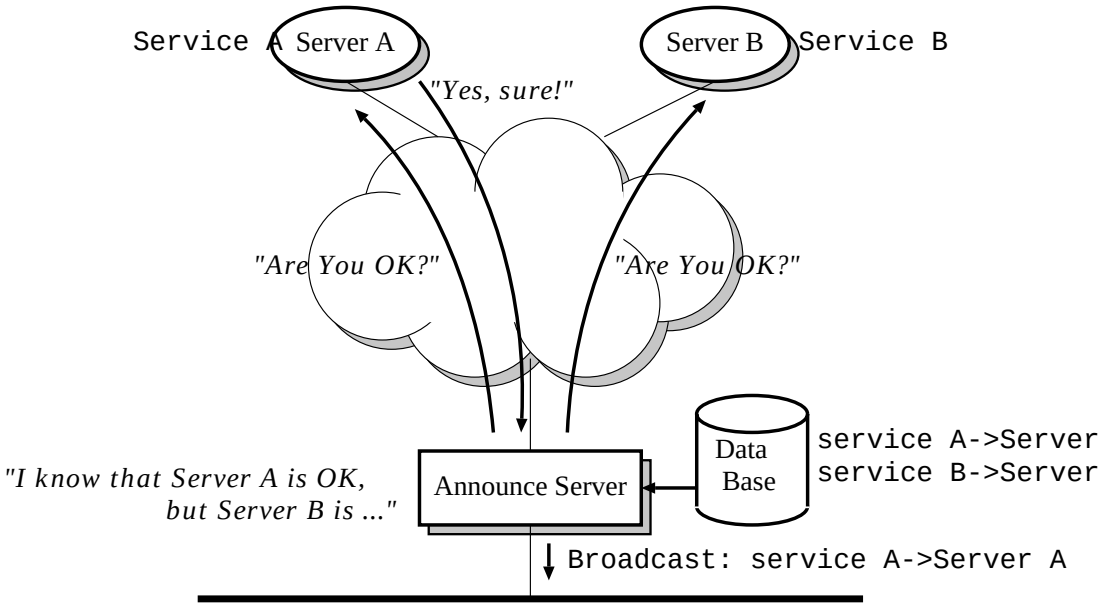



図 2.11: データベースによる対応情報のアナウンス

通常、ネットワーク上にサーバを配置する作業は静的である。そこでサーバを立ち上げた時にデータベースに登録しておきその情報をサービス名による経路制御を行なうネットワークにブロードキャストによってアナウンスする。これにより、ある限られたネットワーク上においてはサービス名による経路制御機構を利用することができるようになる。この

方法では管理者による緻密な管理が必要になるが、管理する範囲が十分に小さい時には実用的である。この方法ではアナウンスサーバを立ち上げる管理者がそのネットワークにおいてどのサーバを利用するかを決定することができる。

しかし、この方式をそのまま用いたのではサーバの故障に対応できない。そこで、実際にはデータベースを管理するホスト、もしくはアナウンスを実際に行なっているホストがアナウンスを行なう前にサーバが故障していないかどうか確認する必要がある。もし、サーバが故障しているような場合にはそのサーバに関するアナウンスを取り止める。

### 2.9.4　その他の方法 2: サーバからのアナウンスリクエストによる方式

図 2.12に示したのはディスタンスベクタ型アルゴルズムを使ったプロトコルの変形である。この方法ではサーバがアナウンスサーバの位置をデータベースとして持ちそこに対して自分が行なっているサービスを知らせることにより、アナウンスサーバが管理しているネットワークに自分の存在をアナウンスすることができる。この方式の特徴はサーバを準備した管理者がアナウンスしたいネットワークを選べることである[5]。

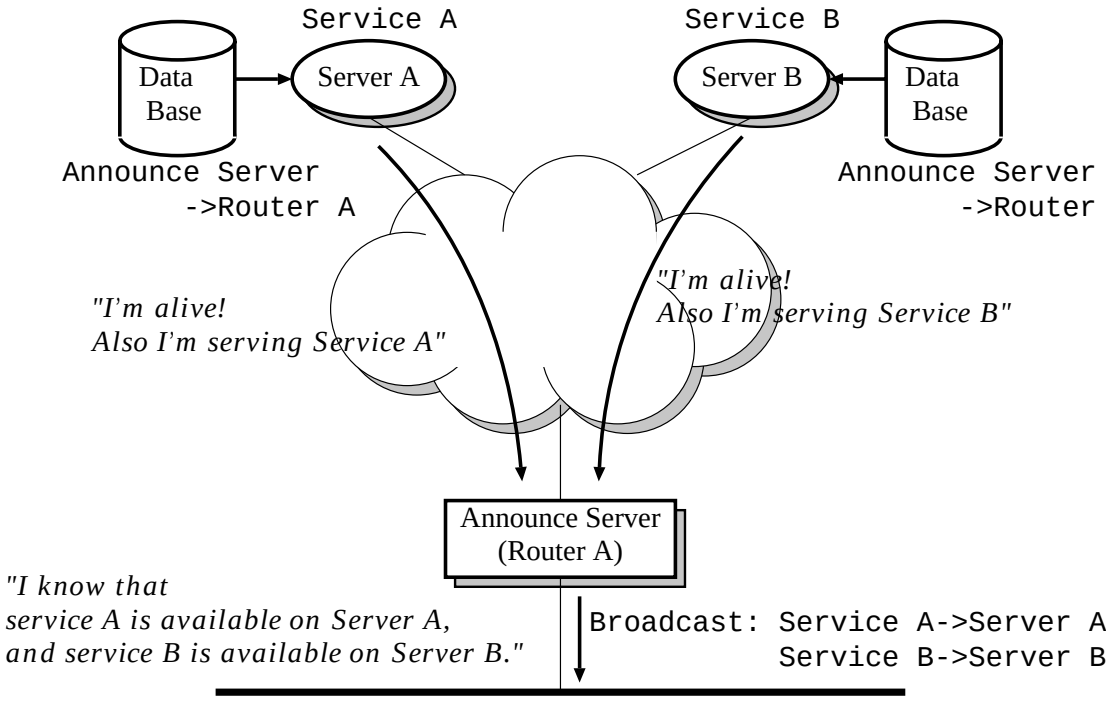



図 2.12: アナウンスサーバによるアナウンス

この方法も十分に小さい範囲を管理する為には有向な手段である。実際に小さいサイトで運用する為には対応付けを用いている限り、このアルゴリズムでも十分である。

[5]ただし、セキュリティ的な意味合いは無いので注意が必要。

### 2.9.5 その他の方法 3: アナウンスサーバによる方法 (1,2 の統合)

その他の方法 1 および 2 で述べた 2 つの方法はお互いに排他的ではなく、また、両方ともアナウンスサーバと言うものを用いている。つまり、この 2 つは同時に実装することができる。つまり、アナウンスサーバは独自のデータベースによってサーバ情報をアナウンスすると同時に、サーバからの要求によってサーバ情報をアナウンスする。この時、当然、アナウンスサーバにはサーバ選択の権利が与えられており、どのサーバをアナウンスするかをここでフィルタリングすることができる。また、全てのサーバをアナウンスし、決定をクライアントに任せることもできる。

## 2.10 移動計算機環境での利用

前節までに、対応付けを用いたサーバ名による経路制御について述べた。これまでの議論により、サービス名による経路制御が実際に実現可能であることが分かった。本節ではこのような経路制御機構を移動計算機環境でどのように利用するかについて議論する。

2.3.2節で述べた様に移動計算機環境において、サーバクライアントモデルに基づくアプリケーションでは次の様な問題点があった。

> 計算機が移動してもサーバの切り替えが起きないので、ネットワーク的に遠距離にあるサーバを応答性が悪いにもかかわらず、そのまま利用することになる。

しかし、これまで見て来たように、対応付けを用いたサービス名による経路制御機構を用いれば、この様な問題を解決することができる。

移動型計算機用のプロトコルの多くは移動時に移動したネットワーク上で一時アドレスを取得すると同時にデフォルトルータに関する情報も取得する。このルータが仮に対応付けを用いたサービス名による経路制御機構を実装しているとすれば、問題は自然と解決する。移動計算機から送信されたパケット (このパケットには適当な受信者アドレスをつけておく) はデフォルトルータに到達し、このルータによってサービス名-アドレスの対応付けが行われる。これにより、パケットはサーバに到達することができる。しかし、もし、デフォルトルータが対応付けを用いたサービス名による経路制御機構を実装していなかった場合、問題は少し複雑になる。この場合は、移動先のネットワーク上にアナウンスサーバが存在しており、更に、移動型計算機がこのアナウンスを受信して自分の対応表を更新する必要がある。この場合は、アプリケーションが指定したサービスは移動ホストからパケットが送信される時点で対応付けがされていることになり、そのまま、サーバに到達することができる。また、快適な動作を望まないまでも、サーバへの到達を保証するためには移動計算機が対応表にエントリを持っていない時のデフォルトのサーバを自分が通常利用しているサーバにしておけば良い。これにより、最悪の場合でも、近くのサーバを利用できない時は自分が通常利用しているサーバを利用できる。

この様に移動計算機環境においても移動型計算機に対応表管理プログラムを動作させておくことによりある限られたネットワーク上、つまりアナウンスサーバが存在するネットワーク上ならアプリケーションを快適に利用することができる。また、これまで通り、最悪の場合は通常自分が利用しているサーバに接続することができる。

## 2.11　設計・仕様

本節では、前節で述べた対応付けを用いたサービス名による経路制御機構を実際に Internet プロトコルスーツ上に実装する場合の細かい仕様について述べる。ここで Internet プロトコルスーツへの実装を採用したのは Internet プロトコルスーツが現在、一番広く利用されているプロトコルスーツだからである。

### 2.11.1　サービス名

対応付けを用いたサービス名による経路制御では、まず、「サービス名」をどのように表すかを定義しなければならない。Internet プロトコルスーツにおいてサーバのサービス提供の端点は、

サーバのアドレス + プロトコル番号 + ポート番号

で表される。ここで、一般的なサービスはプロトコル番号とポート番号はサービスによって予め決まっていることに注意する必要がある。つまり、事実上はプロトコル番号とポート番号の組みがサービスを識別する番号として利用できる。

もちろん、サービス名として文字列による名前を利用することもできるが効率の面からあまり、望ましくなく、今回はサービス名として

プロトコル番号 + ポート番号

採用することにする。

ただし、この方式には次の様な欠点がある。

**同じプロトコルとポートで提供されているサービスを区別できない。**

Firewall 等を用いて、内向き、外向き等のネームサーバを用意している様な環境では同じプロトコルとポートを使って異なった内容のサービスをしている (サービス内容が等価ではない)。このような環境では「プロトコル番号 + ポート番号 」をサービス名として使うことはできない。

**ポート番号が一定していない様なサービスには利用できない。**

サーバのポート番号が時々刻々と変化するようなサービスでは「プロトコル番号 + ポート番号 」をサービス名として使うことができないのは自明である。しかし、実際には Internet 上ではこのようなサービスはなく、現在のネットワークでうまく動作することを目標とする限り問題にはならない。

**TCP/UDP 以外では利用できない。**

TCP または UDP 以外のプロトコルを使用しているネットワークアプリケーションではこの方法を使うことができない。しかし、実際には殆んどのアプリケーションは TCP または UDP を利用して実装されているため、このことについてはあまり問題にならない。

逆に利点としては次のような点を挙げることができる。

**アプリケーションを変更する必要がない。**

現在のアプリケーションはサーバに接続する際に、サーバのアドレス、プロトコル、サーバのポートを指定している。このため、現在のアプリケーションが指定しているものの中からサービス名を取り出すことができ、結果として現在のアプリケーションをそのまま利用できる。

**実装が簡単である。**

実装する場合、文字列による指定と比べて固定長であるため、実装が簡単である。また、これによる高速化も期待できる。

### 2.11.2 対応表およびその管理

対応付けを用いたサービス名による経路制御では各ホスト上に対応表を必要とする。対応表は VIP における Address Mapping Table (AMT) と異なり、デーモン等の管理プログラムによって管理されるため、とくに制御用のフィールドは必要ない。よって、基本的には図 2.13-(a) で十分である。しかし、2.14章で述べる様な将来性のために今回は図 2.13-(b) の様なフォーマットを対応付けエントリとして採用する。

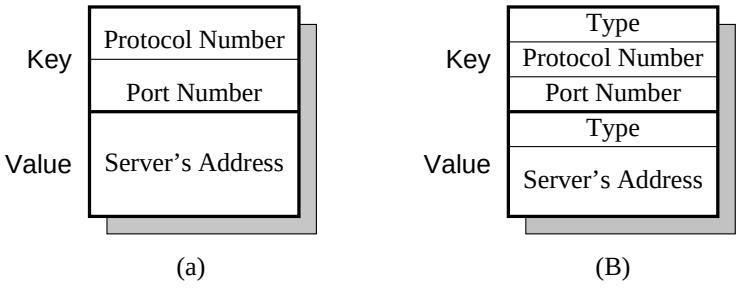



図 2.13: サービス名–アドレス対応付けエントリ

ここで、*Type* はエントリのタイプを示す値であり、現在、ID_INETSERV=1、ID_INETADDR=2 が定義されている。前者はプロトコル番号とポート番号の組み、後者は IP アドレスである。これらの型は、鍵、値どちらにも利用できることができるが、今回の実装では ID_INETSERV を鍵、ID_INETADDR を値とするエントリしか使っていない。

また、今回の仕様ではエントリの値が “0.0.0.0” という値を持つエントリ特別視することにした。このエントリは「透過」の意味を持ち、送信時のみに意味を持つ。これについては 2.11.4節において詳しく説明する。

このような、フォーマットを用いた場合、対応付けを行う部分はパケット毎に表を検索し、エントリを得ることができる。ここで、同一の鍵を持つエントリは複数存在しない、と言う制限を設けておくと、対応付け部では高々一つの対応付けしか存在し得ないので、操作が簡単になり対応付けの高速化をはかることができる。また、経路制御の時に、複数ある同じ鍵を持つエントリの内、今回はどのエントリを対応付けに使うか、といったポリシ的なこともここでは意味を持たなくなる。

しかし、上記の様な制限をつけた場合、2 つ以上の同じメトリックを持つサーバが存在する場合の扱いが問題になる。このようなサーバが存在する場合、どちらを使うかは対応表管理プログラムによって決定する。対応表管理プログラムはこの様な 2 つ以上のサーバを発見すると、自分が持つポリシに従い、どちらを実際の対応表に設定するかを決定する。この時、プログラムの内部情報として 2 つのエントリに対する情報を保持しておくことはとくに制限しない。対応表管理プログラムは 2 つ以上のエントリを保持しておくことによって、

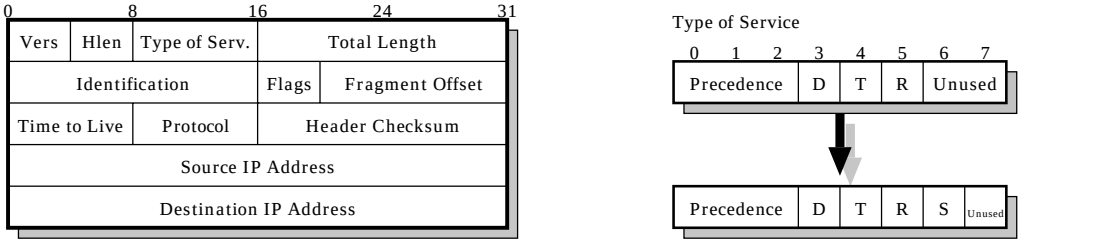
- 定期的にサーバを変更して負荷を分散したり
- サーバ故障時の対応を早めたり

することができる。

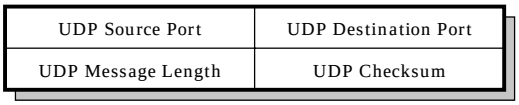
### 2.11.3　パケットフォーマット

対応付けを用いたサービス名による経路制御ではパケット毎にサービス名を付加しなければならない。しかし、今回の実装ではサービス名としてプロトコル番号とポート番号の組みを用いたことにより、サービス名を付加する必要はない。IP パケットヘッダは図 2.14-(a) で示すように、プロトコル番号を含んでいる。また、TCP、UDP ヘッダにはポート番号が含まれている。これらの番号をパケットから取り出し、サービス名として利用することにより、サービス名による対応付けを行うことができる。

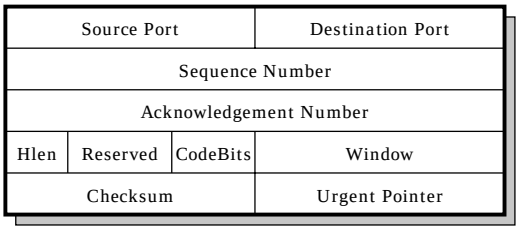
ここで、全てのパケットについてサービス名に基づいて対応付けによるアドレス変換を行うこともできるが 2.12章で述べるような種々の問題があるため、アドレス変換を行うパケットには印をつけておく必要がある。現在の IP パケットのヘッダでは 2 ビットの使われていない空間がある。これは Type of Service フィールドの下位 2 ビットであり、今回の実装ではこの内の上位 1 ビットを変換許可フラグとして使用することにした。




(a) IP Packet Header Format




(b) UDP Packet Header Format




(c) TCP Packet Header Format

図 2.14: パケットフォーマット

### 2.11.4 エンドホストの動作

エンドホストではパケットを送信する時に、対応付けを行わなければならない。エンドホスト上のアプリケーション (クライアント) はこれまでの様に、パケットを送信する際、相手のホストアドレス、使用するプロトコル、ポート番号を指定したパケットを IP 層まで伝える。ここで、TCP 層または UDP 層においてはとくに変更は必要ない。IP 層においては受け取ったパケットのプロトコル番号およびポート番号からサービス ID (エントリタイプ + プロトコル番号 + ポート番号) を作成し、これによって対応表を検索する。エントリが存在すれば、そのエントリの値であるアドレスにアプリケーションから受け取った相手のアドレスを変更し、更に変換許可フラグをセットしてパケットを送信する。ただし、ここで先に述べた様にエントリは存在するが、その値が “0.0.0.0” である場合、アプリケーションから受け取った相手のアドレスを変更せずに変換許可フラグだけをセットしてパケットを送信する。また、アドレスを変更した場合、TCP、UDP 層におけるチェックサムも計算しなおさなければならない。この時、UDP/TCP のデータグラムはフラグメンテーションがおきているかも知れないことに注意する必要がある。図 2.15に送信手順を示す。

一方、受信側では変更点はない。このため、現在運用されているサーバを変更することなくクライアント側を変更するだけで導入することができ、導入の容易性を向上させることができる。

### 2.11.5 ルータの動作

```
ip_output ()
{
    プロトコル番号とポート番号を使って鍵を作成
    if (検索 && エントリがある) {
        if (値が "0.0.0.0" ではない) {
            受信者アドレスを書き換える
            TCP/UDP のチェックサムを再計算
        }
        変換許可フラグを立てる
    }
    通常の IP の処理
}
```

図 2.15: エンドホストにおける送信時の処理

ルータにおいてはパケット送信時の処理に加え、パケット中継時の対応付けが必要になる。ルータはパケットを受信した際にそれが自分宛のパケットではない場合、通常の IP の機構によりパケットを中継する。中継する場合、経路制御表を引くが、対応付けを用いたサービス名による経路制御機構を実装したホストでは経路制御表の検索に先立って、対応表を検索する。中継するパケットに関するエントリが見付かったらアドレスの変換とチェックサムの再計算を行う。この時もエンドホストの送信時同様、UDP/TCP のデータグラムはフラグメンテーションがおきているかも知れないことに注意する必要がある。アドレス変換が終ったパケットは通常の中継の手順を踏んで他のホストに中継される。

また、受信したパケットが自分宛だった場合も少々動作が異なる。ルータは自分宛のパケットを受け取ると上位層に渡す前に対応表を検索する。検索した結果受け取ったパケットに関するエントリが存在し、更に、その値がパケット中の受信者アドレスと異なる場合、もう一度アドレス変換を行い、経路制御しなおす必要がある。

図 2.16にルータのパケット処理アルゴリズムを示す。

### 2.11.6　セッション層

前節までに述べて来たことは全て IP 層での実現であった。UDP を使ったアプリケーションではここまで述べて来たことでサーバ指定、サーバ切り替え双方とも動作する。しかし、TCP を用いた場合、サーバ指定に関しては動作するが切替えは TCP が状態を持つプロトコルであるため、IP における経路制御の方向を変えただけで、そのまま呼が切替わるという分けにはいかない。そこでアプリケーションに対して透過的に呼を張り直す必要がある。

今回はこの機能を実現するために OSI の 7 階層モデルのセッション層にあたる部分を実装することにした。この部分では、TCP の接続を張ったままパケットの経路を切り替える

```
ip_input (tcp or upd header)
{
    if (自ホスト宛) {
        if (変更許可ビットが立ってない ||
                検索の結果、エントリがない ||
                パケットの受信者アドレスとエントリの値が同じ) {
            パケットを上位層に渡す
        }
    }
    ip_forward();
}

ip_forward()
{
    通常の中継動作を行う。
        ただし、対応付けを行う場合は redirect の抑制を行う。
}

ip_output()
{
    if (変更許可ビットが立っている) {
        プロトコル番号とポート番号を使って鍵を作成
        if (検索の結果、エントリがある) {
            if (値が "0.0.0.0" ではない) {
                受信者アドレスを書き換える
                TCP/UDP のチェックサムを再計算
            }
        }
    }
    通常の IP の処理
}
```

図 2.16: エンドホストにおける送信時の処理

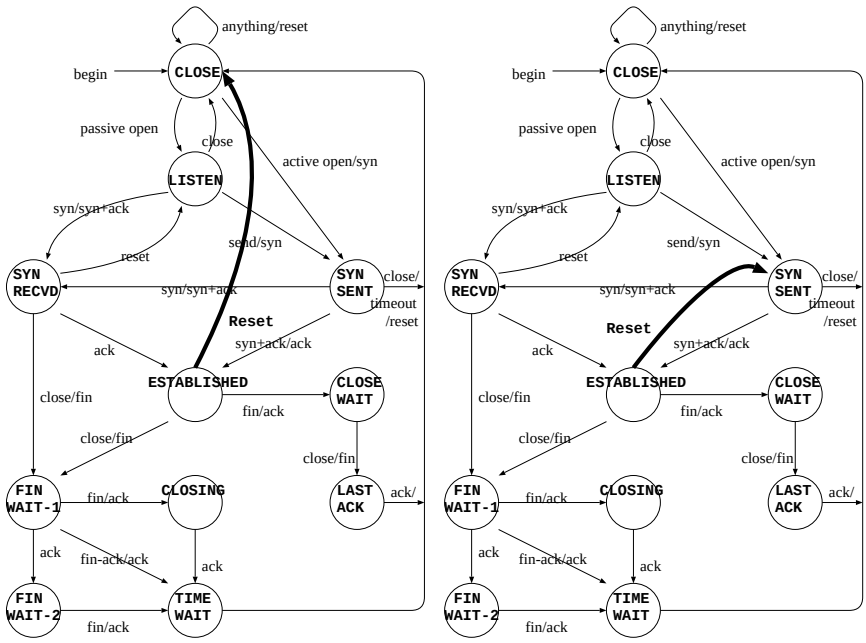



図 2.17: TCP の状態遷移図

と、切り替わった先のホストから TCP Reset が送られることを利用して、TCP にリセットがかかったのをこの部分で受けて、TCP を張り直すことによって経路制御部分によって通信相手のホストが変更されてもアプリケーションから見た場合、TCP によって実現されていた仮想通信路の先が入れ替わっただけの様に見える機能を実現した。

これを TCP の状態遷移図で表すと図 2.17の様になる。実際には太線の矢印は TCP の状態遷移ではなく、一度、アプリケーションに制御を戻して、アプリケーションが指示する部分であるが、概念的にはこの様な矢印が存在するように見える。

### 2.11.7　対応情報交換プロトコル

今回の実装の目的は主に対応付けを用いたサービス名による経路制御が動作することを確認することにある。そこで、対応情報交換プロトコルについてはできるだけ単純で実験に向いているものを選択する。

今回の対応表管理プロトコルでは実装では、その他の方法 2 として前に述べた方法の更に単純化を狙った変形を使う。この方法ではサーバの故障の検出をせず、アナウンスサーバのデータベースだけで運用する。つまり図 2.18の様な構造を持つ。

実際にサーバが故障しているかどうかを調べる方法は幾つかの解決すべき問題点がある。考慮点は次の様なものである。

- サーバホスト自体が何らかの原因によって動作していないことは、ある程度、ICMP Echo 等を用いることによって調べることができるが、経路制御の問題で自分とサー

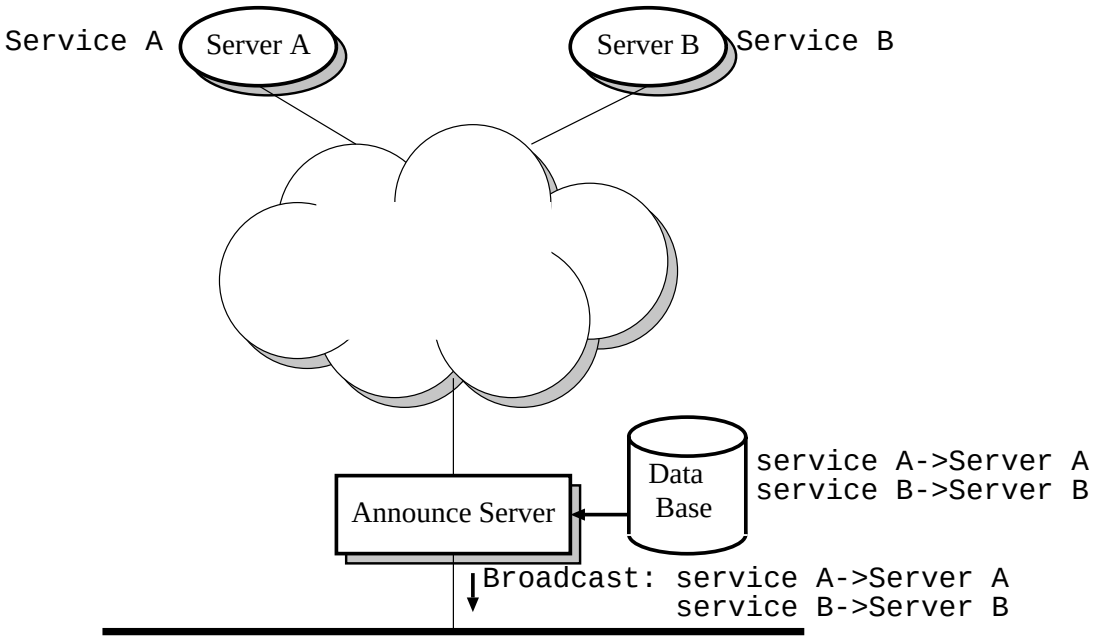



図 2.18: 今回の実験に使用する対応表管理プロトコルの動作アルゴリズム

バの間では通信できないが、クライアントとサーバとの間では通信ができるような状態を発見できない。

- サーバホストは動作しているがサーバプログラムが動作していない状態を認識するのは難しい。

今回の実験ではこの部分を省略していることをここに明記しておく。

この実装では、実験のためにサーバの故障をエミュレートするためにはアナウンスサーバのデータベースから故障発生を仮定するサーバのエントリを削除するば良い。これにより、アナウンスサーバからサーバの動作を確認する部分を省略することができる。

アナウンスサーバは自分が知っている (データベースに登録されている) サーバに関する情報を 1 パケットにしてブロードキャストする。ブロードキャストするデータグラムは図 2.19のようなものである。

## 2.12 実装

本章では、今回の実験で作成したプログラム群の実装について述べる。実装は基本的にカルフォルニア大学バークレー校が開発/配布した BSD 系の UNIX オペレーティングシステムの一部である Net/2 をもとに BSDI 社が開発している BSD/386 上で行ったが、対応表管理用のアナウンスサーバだけは Sony 社のやはり BSD 系の UNIX オペレーティングシステムである NEWS-OS 4.2.1R 上で実装した。

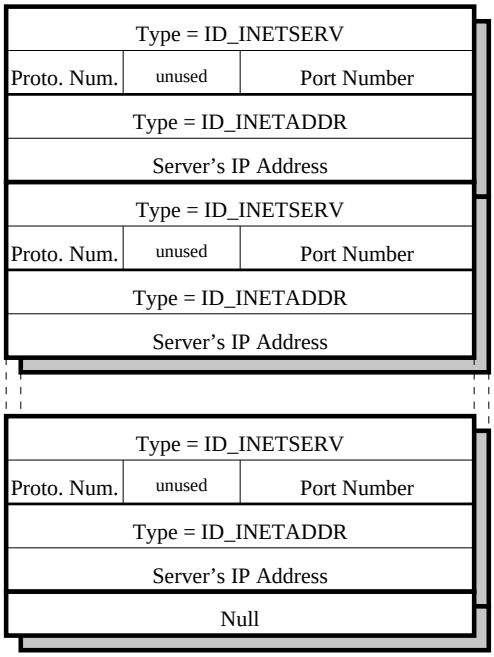



図 2.19: アナウンスデータグラムフォーマット

### 2.12.1　対応付けを用いたサービス名による経路制御

今回の実装では対応付け部の実装はカーネルの中でおこなった。現在の BSD 系の UNIX では IP や TCP、UDP 等のネットワーク機能は全てカーネルの中で実現されている。この部分に変更を加えることにより、対応付けを用いたサービス名による経路制御機構を実装した。この機能を実現するためには次の様な実装が必要である。

- 対応表およびその管理機構の実装
- アドレス変換部の実装

本節ではこの 2 つの部分の実装について個別に述べる。また、Protocol Control Block (PCB) についても送信した時の相手のアドレスと受信したパケットの相手のアドレスが異なるとパケットを受信できない様な実装になっており (TCP のプロトコル的にも本来はこの実装は正しい)、この部分についても変更が必要であったため変更を加えた。これについても以下に述べる。

今回の実装では、対応表はカーネルの中に静的変数領域として実装した。この表は図 2.20 の様な構造を持つ。

ここで、各フィールドは次の様な意味を持つ。

**Flags**

このフィールドはエントリを管理するために使われる。今回の実装では最下位 1 ビッ

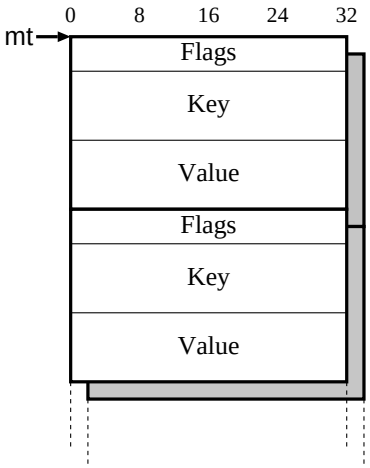



図 2.20: 対応表の構造

トをそのエントリが使われているかどうかを示すためのフラグ (INUSE) として使用している。このフラグが 1 なら現在使用中、0 ならそのエントリは無効であることを示している。

**Key**

このフィールドは対応表の鍵である。鍵の中の細かいフォーマットについては前章で述べた。現在の実装ではこのフィールドに入るのは Type=ID_INETSERV の型のサービス識別子だけである。

**Value**

このフィールドは対応表の値である。鍵の中の細かいフォーマットについては前章で述べた。現在の実装ではこのフィールドに入るのは Type=ID_INETADDR の型の IP アドレスだけである。

また、このような表を管理するために次の様な関数を実装した。

`mt_init()`

初期化のための関数で、表の全てのエントリをゼロクリアし、INUSE フラグを落す。

`mt_add()`

引数に鍵と値をとり、それらを使って新しいエントリを追加する。ここでの鍵の重複チェックは行っていない。これは将来、2.14章で述べるような複数のサーバにパケットのコピーを中継する機能を実現するためである。

`mt_del()`

引数に鍵をとり、その鍵を持つエントリを全て削除する。実際には検索を行い、INUSE フラグを落すことによって呼の機能を実現している。

`mt_lookup()`
　鍵を引数にとり表を検索する。検索後、エントリが見付かればそのエントリへのポインタを返す。検索は線形探索によって実現している。

この部分の実装に C 言語のプログラムで約 150 行を要した。

また、現在の BSD 系の UNIX では IP の実装は基本的に図 2.21の細線で表した部分の様な構造になっている。ここで太線で表されているのが今回新たに追加した部分である。

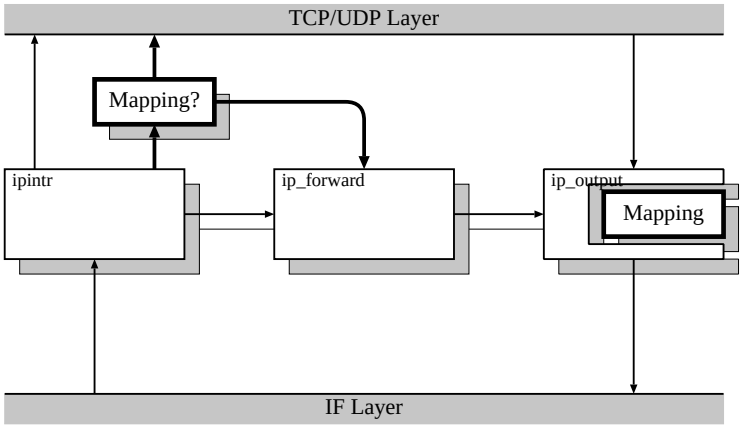



図 2.21: アドレス変換部 (IP 部) の構造

今回実装したうちの「Mapping?」の部分ではパケットに変換許可ビットが立っている場合、単に表検索を行い、そのパケットが要求しているサービスに対するエントリがあるかどうかを調べている。もし、存在し、パケットの受信者アドレスとエントリの値が異なる場合は `ip_forward()` にパケットを渡す。そうでない場合は上位層にパケットを上げる。

「Mapping」の部分では中継の場合はパケットに変換許可ビットが立っている場合に限り、送信の場合には全てのパケットに対して表を検索し、エントリがあればそれにしたがって受信者アドレスを書き換える。この時、TCP、UDP ヘッダにあるチェックサムについても再計算しなければならない。ここで注意しなければならないのはこの部分を通るパケットが既にフラグメンテーションされている可能性があることである。したがって、チェックサムの再計算は UDP/TCP パケットの先頭に位置する IP パケットに限って、変更前のアドレスと変更後のアドレスだけを使って行う必要がある。チェックサムの計算は次の様なアルゴリズムで行うことができる。

$$sum = sum - x_H - x_L \tag{2.4}$$

$$sum = sum_L - (\mathrm{0x10000} - sum_H)_L \tag{2.5}$$

$$sum = sum_L - (\mathrm{0x10000} - sum_H)_L \tag{2.6}$$

$$sum = sum + y_H + y_L \tag{2.7}$$

$$sum = sum_L + sum_H \tag{2.8}$$

$$sum = sum_L + sum_H \tag{2.9}$$

ここで、$sum$ はパケットに含まれるチェックサムの値、$x$ は変更前の IP アドレス、$y$ は変更後の IP アドレスである。$sum$、$x$、$y$ は全て 32 ビットの符号なし数である。また、下付き文字の $L$ や $H$ はそれぞれその数の下位 16 ビット、上位 16 ビットを符号なし数として扱うことを示している。

上式で前半部分は変更前の IP アドレスを引いている部分、後半部分は新しいアドレスに対するチェックサムを計算している部分である。ここで、後半部分はチェックサムの計算式そのものである。また、(2.5) と (2.6)、(2.8) と (2.9) の様に 2 回ずつ同じことをしているのは、1 回目の演算で桁あふれが起きた場合に対処している為である。

アドレス変換を行ったパケットは通常の経路制御機構によって次のホストに渡される。

また、BSD 系 UNIX における Internet プロトコル群の実装ではコネクションを管理するために Protocol Control Block (PCB) と呼ばれる管理エントリをコネクション毎に持つ。このエントリは、自分および相手のアドレスとポート番号、プロトコルを含んでいる。カーネルは IP 層から上がって来たデータグラムをこの PCB と比較して各プロセス (ソケット) に振り分ける。この時、PCB 中の相手のアドレスとデータグラムの送信者のアドレスが一致しないと、たとえ受信側のプロトコル番号 + ポート番号がデータグラムのそれと一致しても上位層にデータグラムが渡されることはない。この実装は、TCP のコネクションが「相手のアドレス + 相手のポート + 自分のアドレス + 自分のポート」で識別されることを考えれば当然である。

しかし、対応付けを用いたサービス名による経路制御方式ではパケット中のアドレスが途中で変換されるため、応答を返すホストはクライアント側から送信した相手と異なったホストになる可能性がある。また、予め最終的にどのホストに到達するかを知る方法もない。このため、カーネルが受け取った応答パケットと PCB の値とに食い違いが生じ、上位層に渡されず、パケットは破棄される。

そこで、今回の実装では、送信時に対応表を調べて変換許可ビットを立てる時に、同時に PCB にも同じ様なフラグを立てることで解決した。パケットを受信した時にこのフラグが PCB に立っていた場合、受信したパケットの送信者アドレスと PCB に設定されている相手のアドレスの比較を省略している。これにより、ある一つのポートでパケットを受けることができるソケットは一つに制限されてしまうが、UNIX における TCP/UDP – ソケット間のインターフェイスの実装は始めからこの様になっているため問題はない。

### 2.12.2 セッション層

前節で述べたように、TCP においては IP パケットの経路を変えるだけではサーバ切替えがうまくいかない。そこでサーバが切り替わったことをアプリケーションに知らせることなしに階層でもう一度コネクションを張り直す機構が必要である。この様な機構を実装

する方法としては次のような 3 つの方法が考えられる。

- TCP の状態遷移を変更する。
- TCP 層と UNIX のソケットインターフェイスの間に再接続の機構を組み込む。
- UNIX ソケットインターフェイスとアプリケーションの間に再接続の機構を組み込む。

はじめの方法は、カーネルを変更する必要がある代わりにアプリケーションを変更する必要がない。二番目の方法も同じである。三番目の方法はアプリケーションを変更する必要があるがカーネルの変更は必要ない。今回は実験実装ということで変更が容易な三番目の方法をとることにした。

UNIX ソケットインターフェイスとアプリケーションの間に再接続の機構を組み込む場合、更に次の 2 つの方法が考えられる。

- アプリケーションを直接変更する。
- ライブラリを新たに作り、socket、send、write 等のシステムコールや関数を新たなライブラリで置き換える。

今回は、アプリケーションの変更を最小限にするために後者の方法をとることにした。

ライブラリでは socket、connect、write、read、send、recv の 6 つについて実装した。ライブラリの中ではこれらのシステムコールをそれぞれ s_socket、s_connect、s_write、s_read、s_send、s_recv という関数名でインターフェイスをもとのシステムコールとそろえて実装している。再接続機構を使うアプリケーションは socket を s_socket、connect を s_connect 等と define してコンパイルしなおすことにより、アプリケーションプログラムを変更することなく再接続機構を使用することができるようなる。

それぞれの関数では本来の動作の他に次の様なことを行っている。

s_socket
: ソケットに関する情報の保存。domain、type、protocol 等を保存。

s_connect
: 接続先のアドレス、ポート番号等の保存。

s_write, s_send
: TCP リセット時の再接続。だたし、TCP リセットがかかった時にそのまま再接続をしたのでは TCP リセットがかかった時に前回送った内容が相手に届いていない可能性がある。そこで TCP レベルでアクノレッジが返って来ていないパケットについては TCP の再接続をした時にもう一度送信する必要がある。そこで、今回は新たに、アクノレッジが返って来ていない分のデータをカーネル空間からユーザ空間に読み出すためのシステムコールを一つ追加してこの機能を実現した。

s_read, s_recv
TCP リセット時の再接続。

### 2.12.3 制御コマンド

今回の実装においては対応付けを用いた経路制御機能をコマンドによってオン・オフする機能をつけた。この機能は AF_INET のソケットに対する ioctl を発行していてカーネル内のスイッチ変数を制御することによって実現されている。

また、対応表を直接制御する (エントリの追加、削除、表示) ためにコマンドを作成した。このコマンドは追加、削除に関しては AF_INET のソケットに ioctl を発行することで、表示に関しては /dev/kmem を読むことで実現した。

### 2.12.4 対応表管理プロトコル

今回の実装では UDP パケットを使い、対応情報をアナウンスサーバがブロードキャストすることによってクライアント (リスナ) が対応表を知る方式を採用した。アナウンスサーバ、クライアント共にデーモンとして実装されており、それぞれ次の様な動作をする。

**●アナウンスサーバ**

今回の実験ではアナウンスサーバはできるだけ簡単な実装を行うようにした。アナウンスサーバは起動されると始めにコンフィグレーションを読み込み、それによってアナウンスリスト (対応表) を作成する。以後、ソケットを開いておき、一定間隔でアナウンスリストを自分が接続されているネットワークに UDP データグラムとしてブロードキャストする。ブロードキャストする時のポートは今回は 5432 番を使用した。

**●リスナ**

リスナの動作はアナウンスサーバに比べて少々複雑である。リスナは起動されると始めにソケットを開きそのソケットを UDP、ポート番号 5432 番、相手のホスト INADDR_ANY で bind システムコールを発行する。その後、自分が持つマッピングテーブル (カーネルではなくデーモンの中に独自のマッピングテーブルを保存している。) を初期化する。ここで、リスナが独自のマッピングテーブルを持っているのは時間情報などを管理するためである。

初期化が終るとリスナはタイマを一定時間にセットしてソケットを受信状態にする。アナウンスサーバからのパケットが受信されるとそれを読み、自分のマッピングテーブルと比べ、新しいものは自分のマッピングテーブルに加えると同時にカーネルにも設定する。この時、自分のマッピングテーブルにはタイムスタンプをつけておく。新しいものではない場合はタイムスタンプの更新のみ行う。また、設定したタイマが動作すると自分のマッピングテーブルを検索し、ある一定時間以上更新されていないエントリに対しては自分およびカーネルのマッピングテーブルから削除する。

## 2.13　実験及び評価

今回実装したソフトウェア群を使って実際に対応付けを用いたサービス名による経路制御方式について実験および評価を行った。今回の評価は主に移動環境における利用に留まっているがここで得られた結果はそのまま固定計算機環境、更には広域のインターネットでも利用できるものである。

### 2.13.1　実験環境

今回の実験は図 2.22のような実験環境を用いて行った。ここで、“Internet” とあるのは実際には WIDE Internet であり、その上部のサイトが電気通信大学の一部、下部のサイトが (株) ソニーコンピュータサイエンス研究所の一部である。

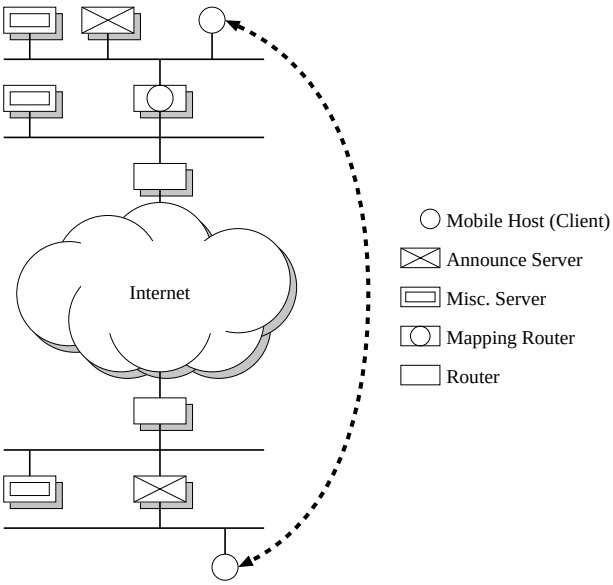



図 2.22: 実験環境

実験で使用したのは移動ホスト (クライアント) として東芝社の DynaBook V486E に BSDI 社の BSD/386 1.1 を OS として載せたもの、アナウンスサーバとしてはソニー社の NWS-3720 及び NWS-3420 に NEWS-OS4.2.1R、NEWS-OS4.2R を OS として載せたもの、マッピングルータとしては Gateway 社の Gateway2000 に BSDI 社の BSD/386 1.1 を載せたものを使った。

このような環境で移動ホストは 2 つのサイト間でオフライン移動を行う。また、マッピンルータの対応表のエントリを変えたり、アナウンスサーバがアナウンスする情報を変更したりした。

### 2.13.2 移動環境での動作に関する評価

**● UDP を使ったアプリケーション**

UDP を使ったアプリケーションの移動環境における動作の評価用として今回はネームサーバを使った。ネームサーバは通常は UDP モードで動作している。また、(Firewall の中などを除いて) 全てのサーバが同じ機能を提供している。

実験は、電気通信大学内のネームサーバを設定した移動計算機 (クライアント) を電気通信大学内で使用しており、その後、電源を切り、ソニーコンピュータサイエンス研究所に持っていき、もう一度移動計算機を立ち上げて通常の利用をするという実験を行った。測定結果を表 2.1に示す。これは `nslookup` コマンドを用いて 100 回名前を検索するのに要する時間を`/usr/bin/time` コマンドを用いて計ったものである。時間をずらして 2 度測定した。検索する名前は測定の前に一度検索してキャッシュにいれておいた。

表 2.1: ネームサーバ検索に要した時間

| | 名前検索に要した時間 (Sec) | |
|---|---|---|
| | 電気通信大学 | ソニー CSL |
| 通常の状態 | 77.36 | 27.12 |
| 負荷をかけた状態 | 222.11 | 36.78 |

ここで、通常の状態とは極普通のネットワーク状態においての測定結果であり、それに対して負荷をかけた状態とはソニー CSL 側のルータから電通大に FTP をしながら測定した状態である。

また、ソニー CSL では Firewall を用いているために、電気通信大学のネームサーバを用いている場合には外側から見えるように設定されているホストしか検索できなかったのに対し、ソニー CSL の中のネームサーバを用いた場合には内部の名前も同時に引けるという利点も得られた。

**● TCP を使ったアプリケーション**

TCP を使った評価用のアプリケーションとしてはかんなかな漢字変換システムを選んだ。かんなかな漢字変換システムは TCP を用いて動作している。また、提供する機能は、かな漢字変換でサーバを変えてもほぼ同じサービスを提供する。

ネームサーバと同じように電気通信大学からソニーコンピュータサイエンス研究所へ移動する実験を行ったがネームサーバの時とは異なり、一つ問題が起きた。かんなサーバを使った時の問題点はサーバが個人辞書や頻度辞書を管理している点である。このため、まず、移動後に個人辞書が存在しない、というエラーがでた。個人辞書をつくってもう一度つなぎ直したところ、移動前と移動後とで変換結果 (変換効率) に差がでた。しかし、これ以外では快適に動作した。

これについては時間の測定方法がなく、電気通信大学をサーバとした場合とソニー CSL をサーバとした場合の差を測定することはできなかったが、使用している感じでは、電気通信大学のサーバを利用していると突然変換が止まることがあるが、ソニー CSL に接続している場合は安定して使うことができた。

### 2.13.3　故障への対応に関する評価

アナウンスサーバの設定を変えることにより、サーバの故障をエミュレートしそれに対する評価を行った。

**● UDP を使ったアプリケーション**

この実験も前節と同じくネームサーバによって評価を行った。実際にアナウンスサーバを立ち上げておき、クライアント側でそれによってネームサーバのポート (UDP の 53 番) のアドレスとの対応を変更するようにした。

結果としては、移動計算機はアナウンスサーバがアナウンスするネームサーバに接続することができ、問題なく動作した。また、アナウンスサーバ自体が動いていない時でも、もともと `/etc/resolv.conf` に設定されているサーバを利用することができた。

**● TCP を使ったアプリケーション**

TCP に関するアプリケーションも UDP に関するアプリケーション同様、前節と同じかんなかな漢字変換システムを用いて実験を行った。

結果は、かんなかな漢字変換システムがサーバとクライアントの間でコネクションを張った時に初期化が必要なため、サーバの切替え後は全然使用することができなかった。

### 2.13.4　アプリケーションの実装に関する考察

今回の実験で、サーバ側に状態を持つ実装がなされているアプリケーションについては故障時の切替えができないことが分かった。しかし、これは実装依存なので、アプリケーションの実装方法を変更することで問題を解決することができる。

サーバが状態を持つようなアプリケーションは現在使われているアプリケーションの内のかなりの比率を占める。このような実装になっている理由には次のようなものがある。

- 幾つかのモード (オプション) があり、そのネゴシエーションのために初期化が必要である。
- 通信路の上を通るデータの情報量を減らす (データを減らす) ために両方に状態を持つ。

ここで、前者については再接続される度に自動的にネゴシエーションを行うことで解決することができる。実際には、アプリケーションがソケットを開いた時に初期化関数を登録しておき、接続した時にライブラリからアップコールされる様な構造にしておけば良い。

これにより、アプリケーションは初期化部と実際のアプリケーションの機能部を分けて設計できる。また、コネクションを開いてしまえばそれを閉じるまで途中でサーバが替わってもそれを意識することなく通信を続けることができるようになる。

また、後者については次のようにして解決することができる。X プロトコルの様にサーバの状態を変えることを目的としているのでなければ、通信に、ある区切りがあることが普通である。例えばかんなサーバなら資源を獲得してからそれを開放するまでが一つの区切りとなる。そこで、アプリケーションが次の一区切りを開始する前にサーバとの同期をとっておき、その状態を記憶しておく。ここで、区切りの途中でサーバの切替えが起こったら、クライアントはその前の同期点からもう一度その区切りをやり直す必要があることを知ることができる。もし、サーバが同じ回答を返してくることが期待できるようなサーバの時はこの最試行をセッション層ライブラリがアプリケーションに替わって行うこともできる。

今回の実装ではアプリケーションを変更せずに再コンパイルだけでセッション層を使えるアプリケーションを作れるようにしようとしたことが失敗を招いた。やはり、これからアプリケーションを設計する時には始めからセッション層を使うことを意識して、本節で述べたような実装にすることが望ましい。

### 2.13.5 まとめ

この実験でオフライン移動時には今回提案し、実装した対応付けを用いた経路制御機構が非常に望み通りの動作をすることが分かった。実際に移動した先でアナウンスされている近いサーバに接続することができ、しかも、移動した先のポリシにあったサーバを利用することができる。

しかし、オンライン移動や故障時のサーバ切替えに関しては UDP によるサーバが状態を持たないサービスでは望ましい動作をするが、サーバが状態を持つようなアプリケーションでは一度クライアントを起動しなおす、または、初期かからもう一度やり直す必要があることが分かった。

## 2.14 応用及び発展性

これまで、対応付けを用いたサービス名による経路制御のみについて述べて来た。しかし、この機構は他の様々な面への応用の可能性を含んでいる。本章では対応付けを用いたサービス名による経路制御に用いた技術の応用、発展性について幾つかの点について議論する。

### 2.14.1　サービス名以外の識別子への応用

今回の研究ではサービス名による経路制御を対応付けを用いて実現する技術について議論してきた。このような技術は十分実用価値があり、近年のアプリケーションに良く利用できることがわかった。ところで、近年のアプリケーションという観点から見た場合、これまで使われてきたホスト名 (アドレス) や今回議論してきたサービス名以外にも指定すべき名前 (ID) が多く存在する。本節ではその中でも重要な 3 つの事項について個別に考察する。

**●人の名前**

まず、ホスト名やサービス名の他に指定する識別子としては人の名前を挙げることができる。通信は本来、人と人との間の情報伝達から始まった。当然、現在でも通信の基礎をなしているものは人と人の間の情報の交換である。

現在の計算機ネットワークで利用されているネットワークアプリケーションにも、この様な、人と人との間の通信を支援するものが幾つか存在する。代表的なものとしては `talk`、`phone` 等を挙げることができる。これらのアプリケーションを利用するためには、`finger` 等の別のアプリケーションを使って相手がどのホストを利用しているか予め調べ、そのホストとユーザを指定して接続を要求する必要がある。

しかし、この様な利用形態は利用者から見た場合、あまり便利なものとは言いがたい。そこで、本論文で提案したサービス名を用いた経路制御機構の応用を考える。基本的なアイディアはサービス名の代わりに相手を指定し、相手が現在、何処にログインしているかという情報をもとに経路制御を行う、というものである。これによって、ユーザは相手がログインしているホストを予め調べることなく、相手が何処にログインしていても名前を指定するだけで相手に接続できる様になる。

実際にこの方式を採用する時には以下のような問題を解決する必要がある。

**空間的スケーラビリティ**

相手は世界中を移動している可能性があり、広範囲にわたる対応表の情報伝達手段が必要である。

**数的スケーラビリティ**

一つのルータに世界中の人間に対する対応表エントリを持たせることはできない。何らかの方法を用いて一つのルータが持つエントリの数を少なく押えなければならない。

**個人識別子**

個人を識別する識別子として何を用いるかについて、適当な識別子を選択または作成しなければならない。

**(高速) 移動**

人間が対応表情報の伝播より高速に移動している場合、どのように処理するのが望ま

しいかを検討する必要がある。

### ●ファイル名

現在の匿名 FTP でも、人名の時と同じことが言える。あるファイルが必要な時、通常は、そのファイルが存在しているホストが分かっていればそこへ接続するが、分かっていない場合は archie 等のアプリケーションを使ってファイルが存在しているホストを調べ、そこへ接続してファイルを取得する。

ここで、本来なら ftp コマンドの引数として指定したいものはファイル自身であることに注意する必要がある。

ファイルが存在するホストに関するデータベースが対応表として存在していれば ftp コマンドに引数として指定するものとしてファイル名を指定することができる。しかも、対応表が経路が最適になるように管理してあれば、ミラーサイト[6]などがある場合、ネットワーク的に一番近いサイトに自動的に接続することを可能にすることができる。

このような機構を実現するためには次の様な問題を解決しなければならない。

#### 空間的スケーラビリティ

匿名 FTP のサイトは Internet の広範囲にわたってに存在している。このような環境に対応するためには広範囲にわたる対応表の情報伝達手段が必要である。

#### 数的スケーラビリティ

匿名 FTP 上には無数のファイルが存在しており、また、匿名 FTP サイト自体が様々な規模で多く存在する。これら全てをどのようにして管理するかを考えなければならない。

#### セキュリティ

匿名 FTP の中には、ある特定の人にのみしか公開していないものがあり、この様なファイルについてどのように取り扱うかについて考えなければならない。

#### 曖昧な指定

archie 等を使う時は、名前がはっきり分かっていなくても大体の名前で検索を行うことができる。このような機能をどのようにして実現するかを考えなければならない。

#### ファイル名とデータの関係

同じ名前で異なるデータが入っているファイルが存在する[7]。このような場合、ファイル名だけでファイルを識別することができない。

---

[6] あるサイトがサービスしているファイル木と同じものをコピーしておきサービスしているサイト。これにより計算機ネットワーク全体での通信量を減らすことができる。

[7] バージョンが変ったなど。

**● Uniform Resource Location (URL)**
ファイル名に似ているが、現在の注目を浴びている識別子として URL がある。URL は Location と言う名前が示すようにその中にホスト名も含んでいる。しかし、実際にはやはりあるファイル (情報) が欲しいのにはかわりはなく、実体が示せればホスト名などは含まれていない方が (位置情報が含まれていない方が) 好ましい。現にこのような要求を受けて現在、Uniform Resource Identifire (URI)、Uniform Resource Name (URN) 等が次々に出て来ている。この様な位置に依存しない識別子を採用することにより Hypertext Markup Language (HTML)[8] の複製等ができるようになる。

また、同じ情報を幾つかのキーワードで検索したい場合が多々ある。このような要求を満たすためにファイル名のところで述べたような問題点の他に別名の定義などをできるようにしなければならないだろう。

前述したようなファイル名による経路制御が実現できれば、現在の URL に代わるような識別子を導入することができる。これにより、リンク先のファイルがある場所が突然かわった、等と言う問題も解決することができる。

### 2.14.2　1 対多の対応表

ここでは今回提案した方法をマルチキャストへ拡張することを考える。これまで述べてきた対応付けは全て 1 対 1 の対応付けであった。これを 1 対多にしてみようと言うのが基本的な考えである。

また、全てのエントリに対してパケットを複製して中継するのがマルチキャストへの拡張なのに対し、複数のエントリの中から一つを選択することによる負荷分散を行なうこともできる。

**●マルチキャストへの適用: データグラム型**
現在のデータグラムを用いたアプリケーションは殆どが通知型のものか、要求/応答型のものである。これらのアプリケーションに 1 対多の対応付けを導入する案について個別に吟味する。

**○通知型のアプリケーション**
通知型のアプリケーションにはこの方式はうまく適用できる。この方式を用いることにより、通知する側から同じイベントを要求している通知される側のホストに対して一度にパケットを送信することができる。例えば、放送開始の通知をこの様な方式で行なうことを考える。その情報が欲しいユーザはあらかじめ自分のホストにも放送開始の通知が来るように経路情報 (対応情報) を流しておき、放送局ではその内容等に応じた識別子で放送開始

[8] ハイパーテキストの記述言語

を通知する。これにより、あらかじめ経路情報 (対応情報) を流しておいたユーザは、通知を受け取ることができる。

この方式は、マルチキャストで良く送信者主体型、受信者主体型の 2 つに分類するが、この内の受信者主体型に当たる。受信者主体型のマルチキャストでは一般的に信頼性を確保するのが難しい。送信者は受信者が何人居るか、等の情報を保持しない為にアクノレッジによる受信確認ができない為である。

この方式の概念図を図 2.23に示す。

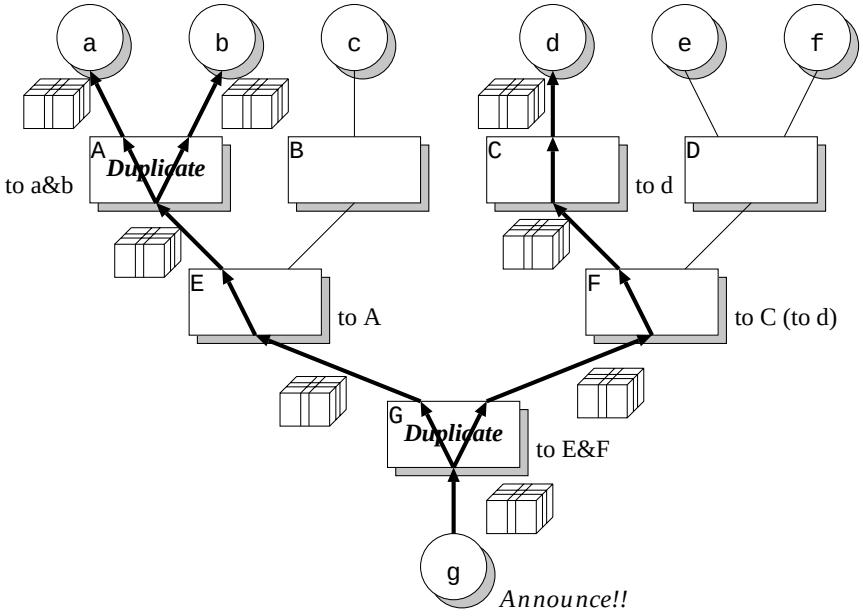



図 2.23: 対応付けを使ったマルチキャストによる通知

ここで、“E & F” 等と書かれている部分は例えば “a & d” でも基本的に問題はない。つまり、はじめにホスト a が加盟し、次にホスト b が加盟する場合は、はじめはホスト a の為にルータ A、E、G を設置しておき、b が加盟した時点でルータ A の表だけを更新すれば良い。この辺りのことに対しては、今後、もっと考察が必要である。

**○要求/応答型のアプリケーション**

要求型のアプリケーションでも通知型のアプリケーションとほぼ同じことが言える。要求者は一つの要求で複数のサーバに対して要求を出すことができ、また、その応答をもらうことができる。しかし、このような使い方ができるアプリケーションはある程度限られてくる。

ネームサーバの様なものではこの方法は余り意味を持たない。要求を複数のサーバに送って同じ返答を複数のサーバから受け取っても誤り訂正以外のことには使えないからである。

もう一つの利用方法としてサーバが異なる結果を返すようなサービスに対して利用する

ことが考えられる。前出の例と異なり、こちらの場合はうまく動作することが期待できる。複数のサーバ上のデータを検索する場合、複数の返答を得ることができる。例えば、archie サーバが提供しているような検索サービスを考える。それぞれの FTP サイト上に archie サーバを挙げておき、一度に全てのサーバに対して要求を発行する。これを受け取ったサーバはそれぞれ、自分が持っているファイル群に対する検索を行ない、もし要求されたものを持っていれば要求元に返答を返す。また、TTL を制限しておくことにより、半径 n ホップ以内等の制限をかけることもできる。このようにマルチキャストを用いることにより、archie 情報について並列検索をすることができる。但し、要求を出した方のホストは複数のホストから返答があることを予想したプログラムを作らなければならない。

このように、要求/応答型のアプリケーションでは同じ結果を返すものでなければマルチキャストによる要求は有効であると言える。

**●マルチキャストへの適用: ストリーム型**

マルチキャストにおけるストリーム型の通信はまだ研究段階であり、実際には運用されていない。しかし、これまでの研究で、主に以下のような分類がなされている。

- 1 対多のストリーム
- 多対多のストリーム

これとは直交する概念で次のような分類もある。

- 信頼性の有るストリーム
- 信頼性の無いストリーム

つまり、多対多の信頼性の有るストリーム通信、1 対多の信頼性の無いストリーム通信等が存在することになる。

ここで提案している方式で、多対多のストリーム通信を実現するのにはその解釈の違い以外には技術的な面での興味深い事項は無い。多対多の通信を行う場合、その通信路自体に名前をつけることになる。ここで名前が通信路につくことが重要である。実はこのことは現在の IP 等におけるマルチキャストと同じ結果になる。つまり、やらなければならないことは、現在、マルチキャストの分野において研究されていることと同じである。言い方を変えれば、「現在、IP 等で使われているマルチキャストは通信路に名前をつけた時の、名前と経路情報の対応付けである。」と言うことができる。

また、信頼性のある通信を行う場合は、現在の IP 上におけるマルチキャスト通信と同じ問題がある。つまり、どのようにして誤りを起こしたパケットを検出するのか、どのようにして誤りを訂正するのか等の問題がある。

**●負荷分散**

これまで、1 対多の対応表を用いてパケットを複製し、マルチキャストを行う場合の可能

性について考察してきた。ここでは、逆に 1 対多の対応表を用いた負荷分散について考察する。この方法の概念図を図 2.24に示す。

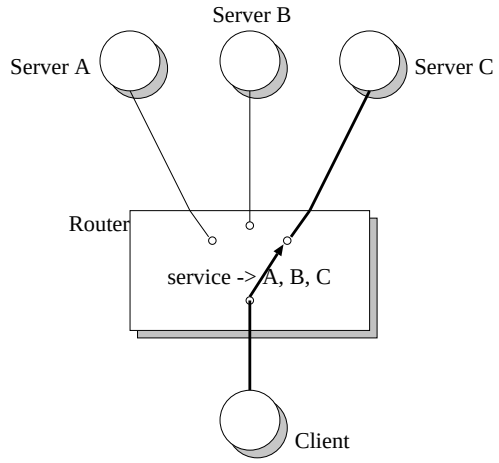



図 2.24: 1 対多の対応表を用いた負荷分散

この方法では、1 対多の対応表を使ってどれか一つのホストにパケットを中継することができる。ここで中継する先を乱数的に、もしくは順に変えることにより、サーバの負荷を分散することができる。

このような分散については 2.11.2節でも少し触れた。2.11.2節で述べた方法では対応表管理プログラムが対応表を管理する時に、定期的にその対応先を変更して負荷を分散する方法だった。ここで、1 対多の対応表による方法と対応表管理プログラムによる方法では以下の様な違いが挙げられる。

- 1 対多の対応表による方法がパケット毎に制御を変更できるのに対して対応表管理プログラムによる方法では一定時間単位でしか制御できない。
- 1 対多の対応表による方法ではパケット処理の時点での処理しなければならない事項が多くなり、性能の悪化が予想される。
- 1 対多の対応表による方法ではパケットの送信元ホストなどによって制御を変えることができるが、対応表管理プログラムによる方法ではこの様なきめ細かな制御はできない。

このような違いを有効に利用するためには、以下のようなことに注意をして吟味しなければならない。

- パケット転送を切替えるポリシがどのくらい複雑か。
- また、そのポリシはパケット毎に制御しなければならないものか。

- それによって得られる利点はどうか。
- 転送時に必要なオーバヘッドはどれくらいか。

また、この方法を用いる場合に限らず、負荷分散をする場合、サーバに状態を持つようなプロトコルではうまく動作しないことに注意しなければならない。

**●まとめ**
以上のように 1 対多の対応表を用いることにより、マルチキャストや負荷分散に応用することができることがわかった。マルチキャストについては幾つかのタイプのアプリケーションに対して現在の IP マルチキャストを用いるより、より、アプリケーションに調和する。また、負荷分散についてもきめ細かな負荷分散を行う事ができる等の特徴がある。しかし、実際このような機構を導入するためにはオーバヘッドなど様々な部分に対する細かい考察が必要である。

また、このような、機構を実現するためには特に次のような問題点について考慮する必要がある。

- 経路のループ
- パケットの複製による増加

これらの問題が解決された時にここで行った提案ははじめて実現することができる。

## 2.15　結論

本論文では、今日のネットワークアプリケーションについて分析し、ネットワークアプリケーションの推移について明らかにした。ネットワークアプリケーションは日々、多様化/複雑化しており、10 年以上前に設計され、基本的にそのまま利用されているネットワークアーキテクチャとの間に狭間ができつつある。

この様な中で、特にネットワーク上の「サービス」と言うものに注目し、サービスを主体とした経路制御機構であるサービス名による経路制御機構を提案した。この機構は限られた条件の基では非常に良く動作し、計算機ネットワークのユーザやネットワークアプリケーションプログラマの負担を減らすことができることが分かった。また、対応付けを用いることにより、現在、実際に動作しているネットワークの中にこの様な新しい経路制御機構を採り入れることができることも分かった。

また、本論文の後半では対応付けを用いたサービス名による経路制御機構の応用として、サービス名だけではなく、様々な名前について同じような機構が実現できる可能性があることについて述べた。更に、最後に今回の研究で得た経験を基に新しいネットワークアーキテクチャの基本的なアイディアについて記した。

これからの計算機ネットワークは徐々に、この様なサービスベースのアプリケーションのようにユーザを意識したものへ移行していくことが予想される。しかし、現在のネットワークアーキテクチャでこの様なアプリケーションを支えるのには限界がある。今日、いよいよ新たなネットワークアーキテクチャを研究/開発する段階に来ていることをここに記して本論文を締めくくる。