RSVP利用時の帯域予約者識別環境の構築

慶應義塾大学環境情報学部徳田研究室
戸田 晴一郎
kiri@sfc.wide.ad.jp

指導教員
徳田 英幸　　村井 純
楠本 博之　　中村 修

平成11年 2月 22日

概 要

本研究では帯域予約を実現する為に提案されたRSVP(Resource reSerVation Protocol)の、現時点における実装の問題点であるポリシコントロールの欠如を補い、帯域予約者の識別を行なう為の機構を実現した。

RSVPはネットワーク上の帯域を予約する為に提案されたが、現在実現されているRSVPでは、誰もが予約を行なうことができる。それにより、現在のRSVPでは有限である帯域を悪意を持った者が占有することが可能である。

このようなRSVPの問題点を補い、帯域予約者の識別とそれによる選択的な帯域予約を行なう為に、ポリシコントロールが必要とされている。現在、RSVP利用時に行なうポリシコントロールの為に用いる情報に関する議論、提案は行なわれているものの、実現はされていない。

一方で、帯域予約許可を決定するポリシ判断を行なう際に交換する情報に関する枠組として、COPS(Common Open Policy Service) Protocol(以下COPS)が提案されている。ただし、COPSではポリシコントロール時のメッセージ交換方法について述べているだけで、ポリシコントロールの実現方法に関しては関知しない。又、現在COPSは提案のみで利用可能な実装は公開されていない。

そこで、本研究ではCOPSを実現すると同時に、COPSを利用するポリシサーバ(PDP : Policy Decision Point)、ポリシクライアント(PEP : Policy Enforcement Point)を設計、実装する。そして、PDP、PEPとCOPSの環境を利用する際にこれまでのRSVPを拡張し、ポリシ情報の取り扱いを可能にする。

本システムと、拡張されたRSVPを組み合わせて利用することで、PDP、PEP内でRSVP利用者の認証を行ない、これまで実現していなかった帯域予約者識別環境を構築する。また、本システムとRSVPによる帯域予約者識別機構は、選択的な帯域資源予約を行なうことで、帯域の不当な占拠を防止する。それにより、帯域資源の管理や予約サービスが可能となり、必要に応じて要求された帯域を確保できる。

本論文ではまず、RSVPとCOPSについて述べ、次にRSVPの問題点を補う機構とPEP、PDPを設計する。その後設計したシステムの実装、評価を行ない、RSVPの変更について有効性を実証する。

Abstract

In this research, a policy control system for RSVP implemented to authenticate the person who made the reservation.

RSVP is a protocol for reserving network bandwidth. Current RSVP had no limitation on accessible protocol for reservation. Consequently, it is posible for the person having spite to occupy the limited resource intentionally.

Policy control must be necessary to distinguish the subscriber and to make a selective resource reservation using RSVP. At present, there is a discussion in using policy control, but there is no implementation.

The COPS(Common Open Policy Service) protocol is proposed to exchange policy informations when it is necessary to enforce policy control like the case of the RSVP. The COPS protocol is only proposed to exchange policy informations, and is not concerned how policy control has been realized. Also, there is a discussion of the COPS protocol, but there is no implementation.

This research realizes the COPS protocol using RSVP, the policy server (PDP : Policy Decision Point), and the policy client (PEP : Policy Enforcement Point). When using this system, we have to extend current RSVP and make possible to use policy information.

By using this environment and extended RSVP, it is able to construct the environment of identify the user making RSVP request. This system can make selective bandwidth reservation, and will prevent excessive occupation of network bandwidth. It becomes possible to control bandwidth resource and enforce the reservation service, and when we need bandwidth resource, we can get it.

In this paper, RSVP and COPS are introduced. Second design of PEP, PDP and propose the framework against the issue of RSVP. And Finally estimate the system implemented, prove effectiveness of changed RSVP.

目次

図 目 次

表 目 次

第1章　序論

1.1　はじめに

インターネット上での通信は、最善努力型のデータ転送が基本となっている。従って、送信されたデータが相手に届くという保証は行なわれない。

そのため、ネットワークの帯域幅が狭い時や、混雑しているネットワーク上で通信を行なう時は、データが相手に到着するための時間が大きくなり、時にはデータの喪失が起こる。但し、通常行われるファイル転送のような通信は時間の制約がなく、データの喪失による再送や、データの遅延という問題は、重大な問題にはならなかった。

しかし、動画や音声など実時間性の高い情報を用いて通信を行なう場合には、データの喪失、到着遅延のばらつきが通信品質の低下につながり、その一方でそれらの情報量はデータサイズが一般に大きく、帯域の消費量が大きい。

このように、幅広い帯域と高品質な通信を要求する動画、音声などの実時間性の高い通信を、インターネット上で実現することは困難であった。そこでこれらの問題に対する解決法として帯域予約の為のプロトコル、RSVP(Resource reSerVation Protocol)[1]が提案、開発された。

1.2　本研究の目的

RSVPを利用することで、希望する大きさの帯域の予約、利用が可能になったが、既存のRSVPの実装では帯域予約の許可を受ける者を判断する手段がない。RFC2205では、ポリシコントロールを用いて、帯域予約要求者に対して可否を判断するとされているが、実際にポリシコントロール機構は未だに実装されていない。

そのため、現在のRSVPでは予約者が希望する帯域に対して、その分の帯域が利用可能であれば誰でも帯域の予約が可能である。それにより、悪意ある者が、必要以上に幅広い帯域の予約を試みた場合でもその予約は許可される。また、悪意ある者によって帯域が占有された場合、他の利用者による通信に弊害が発生する。従って現在のRSVPは実用段階にはない。

このように、現在のRSVPでは、帯域予約者の管理が不可能であり、選択的な帯域予約や予約者に対する課金等も実現できない。

一方で、RSVPのようなポリシコントロールを必要とするものに対してそれを提供する為に、COPS(Common Open Policy Service) Protocol(以後COPS) [2]が提案されている。COPSはポリシコントロールを行なうサーバ、クライアント間で取り交わされるべきプロトコルである。しかし、現在のところCOPSは提案のみで実装は公開されておらず、ポリシコントロールを行なうサーバ、クライアントも実現されていない。

本研究ではCOPSを実現し、RSVPにおけるポリシコントロールの基礎として、帯域予約者の識別環境を構築することを目標とする。

1.3 本論文の構成

本論文では、第2章でRSVPの概要、問題点と提案されている解決法について触れ、第3章でCOPSの概要について触れる。また、第4章でRSVPの為の帯域予約者識別環境の設計について述べ、第5章で第4章の設計に基づく実装について述べる。その後第6章で構築した環境について評価を行ない、第7章の結論をもって本論文を終える。

第2章　RSVPの特徴と問題点

RSVP(Resource reSerVation Protocol) は、実時間性を保証すべき通信など、これまでインターネットで困難であった種類の通信を可能にするために開発された帯域資源予約プロトコルである。

RSVP の特徴として以下の3点が挙げられる。

- 情報受信者による予約
- 受信者予約を可能にするPath、Resv メッセージ
- ソフト状態による管理

本章では、帯域予約を可能にしたRSVP の特徴と概要、RSVP の問題点について述べ、ポリシコントロールの必要性に関して考察する。また、問題点を元に本研究が提案するRSVP 利用時の帯域予約者識別環境の構築について述べる。

2.1　RSVPの概要

本節ではRSVP の特徴および利用形態について概説する。

2.1.1　情報受信者による予約

ネットワーク上で通信を行なう時、通信帯域を予約する為に考えられる方法としては以下の二種類が考えられる。

- 情報送信者による予約
- 情報受信者による予約

これら二種類の予約方式の中で、RSVP は情報受信者による予約方式を用いる。その理由としては、受信者が通信への動的な参加を行なう場合に起こる、状況変化に対

応し、送信者への負担を軽減することが挙げられる。これにより、情報受信者による予約モデルは、情報送信者による予約モデルと比較して大規模な環境で運用可能である。

しかし、情報受信者による予約モデルでは、情報送信者による予約モデルが持たない、以下のような問題も存在する。

- 受信者がいつ予約を行なえばよいのか知る手段がない
- 送信者からの情報がどの経路でネットワークを通るかを判断できない為、適切な経路に従って予約ができない

第一の問題に関しては、RSVP はその時期に関して関与しないものとし、事前に情報受信者と送信者の間で解決すべき問題であるとしているため、本論文での議論の対象とはしないが、第二の問題に関しては Path、Resv メッセージ、ソフト状態による予約管理を用いて解決している。

2.1.2 Path、Resv メッセージ

ネットワーク上の二点間を情報が流れる時、双方が送信する情報が同一の経路を通ることはインターネットでは保証されていない。従って、双方が送信する情報はそれぞれ異なる経路を通って通信されることが多々あるが、このモデルは、RSVP を用いる際には障害になる。

前節に述べたように、RSVP では情報受信者が通信帯域の予約を行なう為、情報受信者から情報送信者への経路上で通信帯域の予約が行なわれる。しかし、インターネットの経路制御機構では、予約要求は、実際に情報送信者が送信する情報が、予約した経路以外を通ることが考えられる。図 2.1 に情報受信者と送信者のお互いの経路が異なる場合を示す。

図 2.1 では情報送信者から情報受信者への情報が伝達される経路と、情報受信者から情報送信者への情報伝達の経路を示している。この図の例では、情報受信者から情報送信者への経路に従って帯域が予約された場合、双方の経路が一致しない為、情報送信者から情報受信者への情報転送時に予約された帯域は利用されない。

この問題を回避し、適切な経路制御を保証する為に、RSVP では情報送信者が受信者へ向けて送信する経路情報メッセージ (Path メッセージ) と、情報受信者が送信者へ向けて送信する予約情報メッセージ (Resv メッセージ) を規定している。RSVP ではそれらを用いることで正しい経路に従って通信帯域の予約を行なうことを可能にする。

情報受信者は Path メッセージを用いることで Resv メッセージを送信すべき経路を認識し、その情報によって Resv メッセージは送信され、情報送信者までの正しい経路に従って予約が行なわれる。

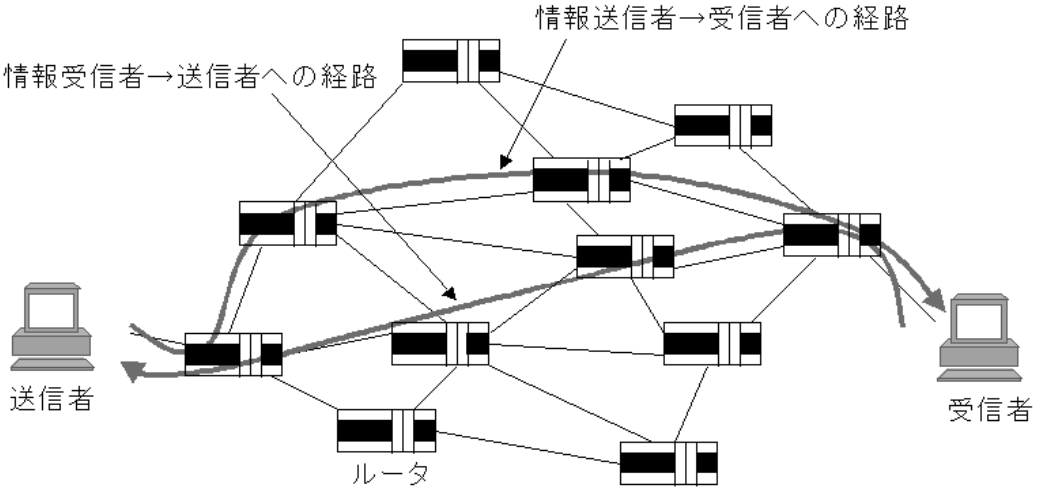

図 2.1: 情報伝達経路の相違

2.1.3 ソフト状態による管理

インターネット上で通信を行なう時、情報が伝達される経路は動的に変化する。これは、経路上のルータが停止したり、新たにより良い経路が出現する為であるが、一度行なわれた予約の更新を行なわない帯域予約モデルをとる場合はその変化への対応が不可能である。

上に述べた動的なネットワークの変化という問題に対して、RSVP はソフト状態による管理方針を採用し、問題点を回避している。ソフト状態とは、帯域予約の状態が一時的なものであることを意味する。すなわち RSVP では、Path、Resv メッセージを情報送信者、受信者がそれぞれ送信することによって帯域を確保するが、その予約は一定時間内にリフレッシュされない場合は破棄される。

そのため、RSVP を用いて通信帯域を予約する時は、ある時間的間隔ごとに Path、Resv メッセージを送信し続けなければならない。このように、頻繁に Path、Resv メッセージが送信されることでネットワーク上の経路の動的な変化に対応可能になる。

2.1.4 予約判断方法

情報送信者までの各ルータ内に存在するRSVPプロセスはResvメッセージを受け取った場合、アドミッションコントロールとポリシコントロールを行なって予約許可を判断する。

アドミッションコントロールは、要求された帯域に従って予約の可否を判断する為に用いられ、ポリシコントロールは、帯域を要求した者が予約を行なう資格を持つことを判断する為に用いられる。しかし、現在のRSVPではアドミッションコントロール機構は提供されているものの、ポリシコントロール機構は提供されていない。

2.2 帯域予約時の問題

前節で述べたように、帯域予約を行なう際、RSVPプロセスはポリシコントロールを受ける必要があるとされる。ただし、現在実装されているRSVPではポリシコントロールが実現されていない為、アドミッションコントロールによって採択されれば、予約者に関わらず帯域の予約が可能である。

しかし、ポリシコントロールを行なわない場合、帯域を予約する者の識別が不可能であり、誰もが帯域を予約可能になる。その結果、悪意ある第三者が不当に帯域を占有する可能性が有り得る。そこで、RSVPでポリシコントロールを行なう際に用いる枠組みとして、以下のものが提案されている。

- POLICY_DATAオブジェクトに関する提案[3]
- RSVP利用者の認証に関する提案[4]

2.3 提案されている解決法

前節で述べたように、現在のRSVPではポリシコントロールを行なえない。本節ではRSVPでポリシコントロールを可能にする為に提案された方法について述べる。

2.3.1 POLICY_DATAオブジェクトに関する提案

現在までのRSVPの規格では、ポリシコントロールを行なう為に利用すべきPOLICY_DATAオブジェクトが未定義の状態に置かれている。そこでこの提案では、未定義であったPOLICY_DATAオブジェクトを新たに図2.2、2.3のように規定している[3]。

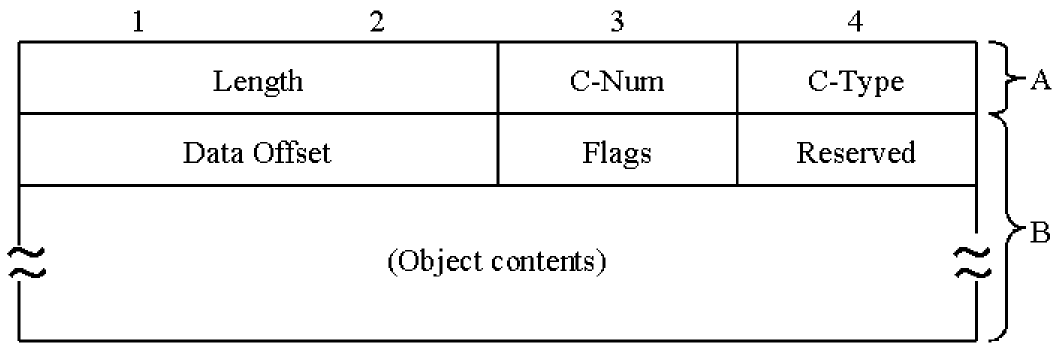

図 2.2: POLICY_DATA オブジェクトヘッダ

A : RSVP オブジェクトヘッダ部。

Length : 16 ビット。
RSVP オブジェクトの長さを記述する。

C-Num : 8 ビット。
クラス番号部。POLICY_DATA クラスの C-Num は 14 である。

C-Type : 8 ビット。
RSVP オブジェクトごとに定義されるクラスのタイプ。[3] では、POLICY_DATA オブジェクトとして 1 が規定されている。

B : POLICY_DATA オブジェクト固有のヘッダ部

Data Offset : 16 ビット。
オブジェクトヘッダの先頭部から数えたデータの場所

Flags : 8 ビット。
POLICY_DATA オブジェクトにフラグメントが起きたことを示す。

Reserved : 8 ビット。
予約済み。常に 0 にセットされる。

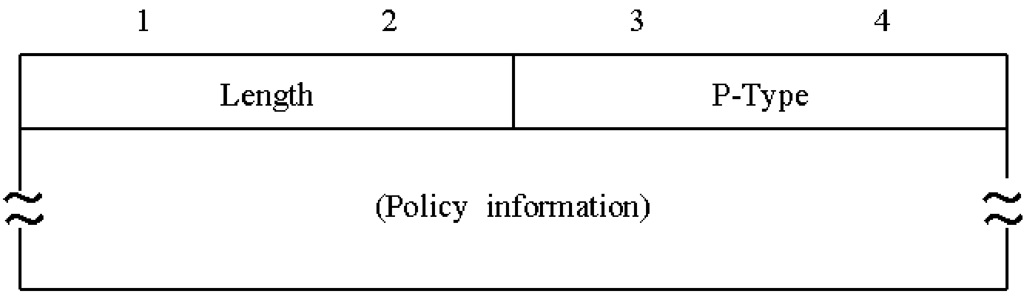

図 2.3: policy element ヘッダ

Length : 16 ビット。
policy element の長さを記述する。

P-Type : 16 ビット。
policy element のタイプを記述する。

2.3.2 RSVP 利用者の認証に関する提案

ポリシコントロールを行なう際に、予約要求者の識別方法を記述する為にPOLICY_DATA オブジェクト内に含まれる情報であるpolicy element としてAUTH_DATA が提案されている。AUTH_DATA はユーザプロセスによって生成され、RSVP プロセスに引き渡され、その中に帯域を利用する者の認証を行なう為の情報を格納する。

AUTH_DATA は、更に詳しい情報を格納する為、AUTH_DATA ヘッダを規定し、その中にAuthMethod と呼ばれる詳細な情報を定義可能にしている。図2.4 に新たに提案されたpolicy element、AUTH_DATA の定義を記す。AUTH_DATA は、その中に帯域利用者の情報を記述するATTRIBUTE LIST を持つ。図2.5 にATTRIBUTE LIST ヘッダの定義を示す[4]。

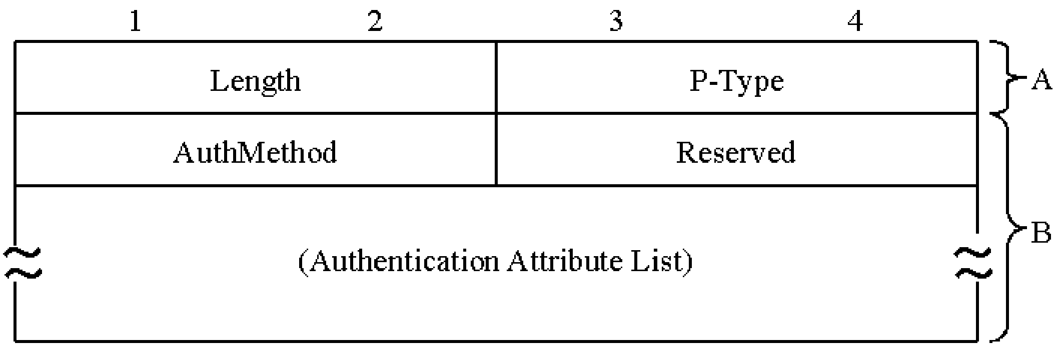

図 2.4: AUTH_DATA ヘッダ

A： policy element ヘッダ部。
P-Type は 1 とする。

B： AUTH_DATA ヘッダ部。

AuthMethod： 16 ビット。
認証方式を記述する。

1 = AUTH_SIMPLE　暗号化を行なわない認証

2 = AUTH_KERBEROS　認証プロトコルとして Kerberos を利用

3 = AUTH_PKEY_DC　デジタル署名を用いる公開鍵による認証

Reserved： 16 ビット。
予約済み。常に 0 にセットされる。

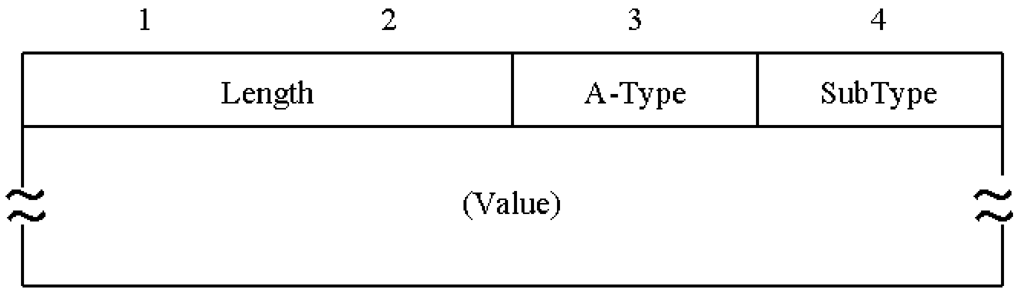

図 2.5: ATTRIBUTE LIST ヘッダ

Length : 16 ビット。
ATTRIBUTE LIST の長さを記述。

A-Type : 8 ビット。
ATTRIBUTE LIST のタイプを記述。

1 = USER_ID User network login ID.

2 = USER_DN User Distinguished Name[RFC 1779].

3 = USER_CRED User credentials.

SubType : 8 ビット。
ATTRIBUTE LIST のタイプごとに定義される。

第3章　COPS

COPS(Common Open Policy Service) Protocol は、RSVP などのプロトコル内でポリシコントロールを行なう為に提案されたプロトコルである。

COPS は、RSVP などのポリシクライアントとポリシサーバとの間の通信を行なう為のメッセージの交換方法を定義しており、RSVP などに応用可能である。本章では、COPS の概要と設計について述べ、最後に RSVP 上での COPS を用いたポリシコントロールの実現手法について述べる。

3.1　COPS の基本モデル

COPS はポリシコントロールを必要とする様々なプロトコルに対応できるように設計され、図 3.1 に示すモデルで利用される。

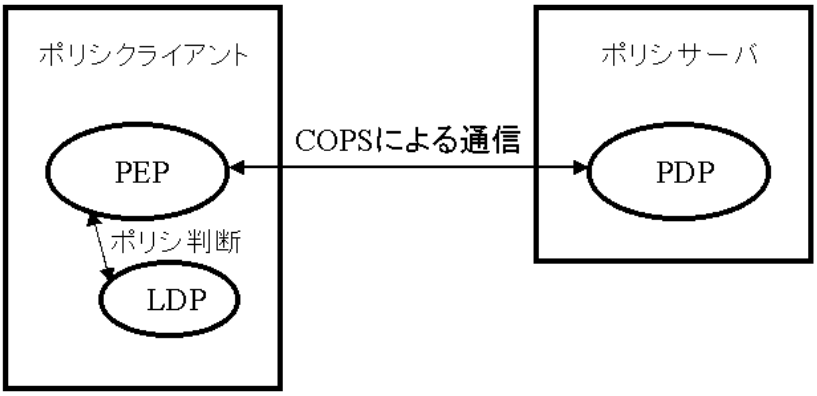

図 3.1: COPS 利用モデル

図 3.1 は簡単なクライアント・サーバモデルである。クライアントである PEP(Policy Enforcement Point) は、ポリシ判断の要求に呼応する形で PDP(Policy Decision Point)

に対してTCPを用いて通信を行なう。PEPからPDPへと転送されたポリシ判断のリクエストはPDPで処理された後、その結果がPEPへ戻され、PEP内でポリシ判断の結果が処理される。

ただし、PEPはPDPに対してポリシ判断の要求を行なう前に、PEPと同じノード内に存在するLDP(Local Decision Point)でのポリシ判断を行なうこともできる。PEPが受け取ったポリシ情報を、リモートのPDPに転送する前にLDPで処理することによって、個々のノードに固有で、より詳細なポリシ判断を行なえる。

本節の冒頭で述べたように、COPSはある単一のポリシクライアントをサポートする為に設計されたのではなく、様々な種類のポリシクライアントの取り扱いを考慮している。しかしそれぞれのポリシクライアントに対するPEP、PDPの振る舞いは異なると考えられるので、COPSではそれぞれのポリシクライアントを識別し、処理する。

COPSはPEP-LDP間、PEP-PDP間でメッセージを交換する際に用いられる。次節では、COPSメッセージの詳細を述べる。

3.2 COPSヘッダ及びオブジェクト内容

COPSメッセージは、全て共通のCOPSヘッダを持ち、その後ろにそれぞれいくつかのオブジェクトを連結することによって構成される。そこで、まずCOPSヘッダの定義を示した後にオブジェクトの内容を簡単に示し、COPSメッセージの役割について述べる。COPSオブジェクトの定義については付録Aに示す。

3.2.1 COPSヘッダ

COPSヘッダは、COPSメッセージを作成する時には必ず付加される。その内容はCOPSのバージョン情報、以下に続くメッセージの種類、COPSメッセージを利用するポリシクライアントのタイプ、メッセージ長を記述することを可能にする。図3.2にCOPSヘッダの定義を記し、以下にそれぞれのフィールドの役割について簡単に述べる。

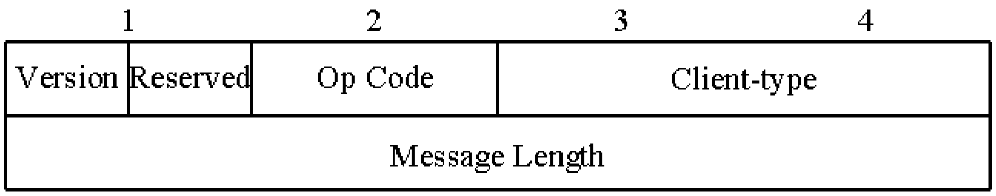

図 3.2: COPS ヘッダ

Version : 4 ビット。
バージョン番号を表す。現在のバージョンは 1 である。

Reserved : 4 ビット。
予約済み。常に 0 にセットされる。

Op Code : 8 ビット。
以下に COPS メッセージの種類を表す。

1 = Request (REQ)

2 = Decision (DEC)

3 = Report State (RPT)

4 = Delete Request State (DRQ)

5 = Synchronize State Request (SSQ)

6 = Client-Open (OPN)

7 = Client-Accept (CAT)

8 = Client-Close (CC)

9 = Keep-Alive (KA)

10 = Synchronize State Complete (SSC)

Client Type : 16 ビット。
クライアントの種類を表す。

Message Length : 32 ビット。
COPS header を含むメッセージ全体のバイト数。

3.2.2 COPS オブジェクトヘッダ及びオブジェクト内容

COPS オブジェクトは、COPS メッセージ内にいくつか含まれることがある。その時にそれぞれのオブジェクトを認識する為に COPS オブジェクトヘッダが規定されている。図 3.3 に COPS オブジェクトヘッダの内容を示し、表 3.1 にオブジェクトのクラスとその目的を示す。

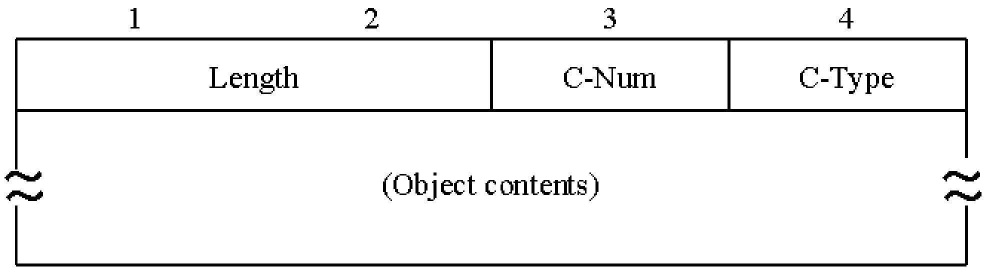

図 3.3: COPS オブジェクトヘッダ

Length : 16 ビット。
Object header を含むオブジェクト全体のバイト数

C-Num : 8 ビット。
表 3.1 にオブジェクトのクラスとその目的を示す。

C-Type : 8 ビット。
それぞれのオブジェクトによって規定される。

表 3.1: COPS オブジェクト一覧

C-Num	オブジェクト名	目的
1	Handle	許可された予約が与えられる PEP、PDP 内での識別子。
2	Context	COPS メッセージを引き起こした 予約要求の種類の識別子。
3	In Interface	予約要求が到着したインターフェイス名の識別子。
4	Out Interface	予約要求が送信されインターフェイス名の識別子。
5	Reason code	予約状態破棄理由の識別子。
6	Decision	予約要求に対する判断理由の識別子。
7	LDP Decision	LDP 内での、予約要求に対する判断理由の識別子。
8	Error	COPS プロトコルエラーの識別子。
9	Client Specific Info	予約者識別に必要なの情報を記述子。
10	Keep-Alive Timer	Keep-Alive を何秒おきに行なうかの記述子。
11	PEP Identification	PEP の識別子。 内容は IP アドレス、又は文字。
12	Report Type	handle に対する保持される状態の識別子。
13	PDP Redirect Address	PDP が PEP とのセッションを終了する時に 再接続を行うべき PDP アドレスの記述子。
14	Last PDP Address	PEP が以前にコネクションを持っていた PDP アドレスの記述子。
15	Accounting Timer	何秒おきに PEP 内の情報を PDP に対して送信するかの記述子。

3.3 COPS メッセージ

COPS メッセージでは、COPS ヘッダによって規定された 10 種類のメッセージが想定されている。本節では各 COPS メッセージが構成されるオブジェクトの組み合わせを記し、その後に各 COPS メッセージの役割を簡潔に述べる。それぞれのメッセージ構成は、含まれるオブジェクトの順序に従って記述されており、必ずしも必要でないオブジェクトは [] で囲んで表現している。

3.3.1 Request(REQ)

```
<Request Message>  ::=  <Common Header>
                        <Client Handle>
                        <Context>
                        [<In-Int>]
                        [<Out-Int>]
                        [<ClientSI>]
                        [<LDPDecision>]
```

REQ メッセージは PEP から PDP へ向けて送信される。REQ メッセージには Handle オブジェクトと Context オブジェクトを含む必要があり、PDP は Handle オブジェクトを用いて予約内容を管理する。ここで用いられた Handle オブジェクトは REQ オブジェクトによって要求された予約に対して用いられ、その後の COPS メッセージ交換の際に処理を軽減する為に利用される。

3.3.2 Decision(DEC)

```
<Decision Message>  ::=  <Common Header>
                         <Client Handle>
                         <Decision> | <Error>
```

DEC メッセージは PDP から PEP へ向けて送信される。このメッセージは PEP からの REQ メッセージに対応して生成され、対応する Handle オブジェクトと一つ以上の Decision オブジェクトによって構成される必要がある。

PEP は PDP から送信された DEC メッセージによって、要求した予約の PDP による判断結果を認識する。DEC メッセージの結果、予約が採択された場合には Decision オブジェクトを含んで DEC メッセージが構成され、却下された場合には Error オブジェクトを含んだ DEC メッセージが構成される。

3.3.3 Report State(RPT)

<Report State Message> ::= <Common Header>
<Client Handle>
<Report-Type>
[<ClientSI>]

RPT メッセージは PEP から PDP へ向けて送信される。RPT メッセージは PDP によって許可された予約内容に変更があった場合に送信され、Handle オブジェクトと Report Type オブジェクトが必要である。PDP では Handle オブジェクトによって変更すべき予約内容を知り、その予約内容に関して Report Type オブジェクトの情報から変更を行なう。

3.3.4 Delete Request State(DRQ)

<Delete Request Message> ::= <Common Header>
<Client Handle>
<Reason>

DRQ メッセージは PEP から PDP へ向けて送信され、Handle オブジェクトと Reason オブジェクトが含まれなければならない。PDP は DRQ メッセージによって、Handle オブジェクトに対応する予約が無効になることを知り、対応する予約内容を消去する。また、PDP は Reason オブジェクトによって何故その予約が破棄されるべきなのかを知る。

3.3.5 Synchronize State Request(SSQ)

<Synchronize State Request Message> ::= <Common Header>
[<Client Handle>]

SSQ メッセージは PDP から PEP へ向けて送信される。SSQ メッセージは特にオブジェクトを含まず、COPS ヘッダのみによって構成されてもよい。PEP は SSQ メッセージを受け取った時、PEP 内に保持されているクライアントの情報を PDP へ向けて転送する。転送するクライアントの情報を指定したい時は Handle オブジェクトを用いる。

3.3.6 Client-Open(OPN)

```
<Client-Open Message>   ::=   <Common Header>
                              <PEPID>
                              [<ClientSI>]
                              [<LastPDPAddr>]
```

OPNメッセージはPEPからPDPへ向けて送信され、PEP Identificationオブジェクトを含まねばならない。このメッセージはPEPが起動した時に、サポートするクライアントタイプをPDPに対して知らせるために用いられ、PDPはこのメッセージを元にClient-Openメッセージを送信したPEPがサポートするクライアントタイプを確認する。

3.3.7 Client-Accept(CAT)

```
<Client-Accept Message>   ::=   <Common Header>
                                <KA Timer>
                                [<ACCT Timer>]
```

CATメッセージはPDPからPEPへ向けて送信され、Keep-Alive Timerオブジェクトが含まれなければならない。このメッセージはOPNメッセージに対応して生成され、Keep-Aliveを行う間隔を記述する。

3.3.8 Keep-Alive(KA)

```
<Keep-Alive Message>   ::=   <Common Header>
```

KAメッセージはPDP、PEPの両方向から送信される。KAメッセージはCOPSヘッダーのみで構成され、お互いにネットワーク上で通信が行なえることを知るために用いられる。KAメッセージ、あるいはその他のメッセージがある一定期間受信されない時は、相手のノード、またはネットワークに問題があるものとして通信を切断する。

3.3.9 Client-Close(CC)

<Client-Close Message> ::= <Common Header>
<Error>
[<PDPRedirAddr>]

CC メッセージはPDP、PEP の双方から送信され、Error オブジェクトが含まれなければならない。PDP が CC メッセージを受け取ったとき、CC メッセージを生成したPEP がそれまでサポートしていたクライアントタイプをそれ以上サポートしないこと知る。

PEP がCC メッセージを受け取ったとき、もしPDP Redirect Address オブジェクトが含まれていれば、新たにPDPRedirAddr オブジェクト内のアドレスを利用してOPN メッセージを発行する。しかし、受け取ったCC メッセージがError オブジェクトのみで構成される単純なメッセージであった場合、PEP は終了する。

そして、PEP、PDP 共にError オブジェクトによって何故CC メッセージが生成されたのかを知る。

3.3.10 Synchronize State Complete(SSC)

<Synchronize State Complete Message> ::= <Common Header>

SSC メッセージはPEP からPDP に向けて送信される。SSC メッセージはSSQ メッセージに対応して生成され、PDP からのSSQ メッセージに対してPEP 内の情報を送信し終わったことを示す。SSC メッセージは単純にCOPS ヘッダのみで構成される。

3.3.11 COPS メッセージ交換の流れ

ここまで述べてきたCOPS メッセージはPEP とPDP との間で取り交わされる。これらのメッセージには、それぞれ対応関係が決定しているものがあり、それらは受け取ったメッセージに対して返答する為に用いるメッセージが決まっている。表3.2 にそれぞれのメッセージの対応関係について記す。

表 3.2: COPS メッセージの対応関係

受け取ったメッセージ	返答に用いるメッセージ
Client-Open	Client-Accept(成功時) Client-Close(失敗時)
Request	Decision
Syncronize State Request	Report State の後に Synchronize State Complete

3.4 関連研究

COPS に関連する研究としては、ポリシコントロールの枠組を提案しているものとしてポリシコントロールが行なわれるモデルについて考察を行ない、PEP、PDP、LDP の提案を行なっている文献[5] を挙げることができる。

しかし、この研究は枠組を提案したものである為、ポリシコントロールを行なう際に PEP、PDP、LDP の果たすべき役割のみについて考察が行なわれている。そのため、実際に PEP、PDP、LDP でどのような処理が行なわれるべきかに関しては全く述べられていない。

COPS の提案の中では[5] で考察される PEP、PDP、LDP を利用してポリシコントロールを行なうと考慮している。しかし、それらは単純にポリシ情報を交換することのみが想定されており、ポリシコントロールをどのように行なうべきかということに関しては COPS を用いる個々の状況に応じた PEP、PDP、LDP の定義が必要となる。

3.5 RSVP における COPS の利用

本章の冒頭で述べたように、COPS は様々なポリシクライアントに対応可能な設計となっている。それにより、RSVP を利用してポリシコントロールを行なう時、RSVP プロセスをポリシクライアントとして COPS による情報交換を行なうことができる。

また、COPS を用いることで RSVP に対応した PDP、PEP を構築し、その環境の利用によって RSVP の為の予約者識別が可能になる。本章までに述べた RSVP の問題点の解決法と COPS の提案を元に、次章より RSVP の為の予約者識別環境を構築する為の設計、実装を述べる。

第4章　RSVP利用時の帯域予約者識別環境の設計

本章ではRSVPを使用する際に帯域予約者の識別が行なえることを目標として、COPSを利用するPDP及びPEPの設計を述べる。これまでRSVP内のPOLICY_DATAオブジェクトは定義されたものの利用はされておらず、COPSも提案のみで実装が行なわれていない。

さらに、COPSにおけるPEP、PDPは、ポリシコントロールを必要とするポリシクライアントごとにそれぞれ任意の処理を行なうものとされる。そのため、それらが持つ振る舞いはポリシクライアントごとに異なるものとされるが、RSVPをポリシクライアントとするPEP、PDPは実現していない。

本システムではRSVPのPOLICY_DATAオブジェクトとCOPSを用いることで、RSVPでの帯域予約者識別を実現することを目標とする。同時に、これまでに提案されたPOLICY_DATAオブジェクトを拡張し、情報転送の効率化を図る。

4.1　POLICY_DATAの拡張

RSVPでは、ポリシコントロールを行なう為のオブジェクトとして、POLICY_DATAオブジェクトを利用する。しかし、現在提案されているPOLICY_DATAオブジェクトは単純に帯域予約者情報を伝達する為のもので、ポリシコントロールを行なう際のPEP内での処理軽減に関して改良の余地が残されている。そこで本節では新たにPOLICY_DATA内のpolicy elementとしてPPDP_ADDR(Previous PDP address)を提案する。

4.1.1　RSVPにおけるポリシコントロールモデル

POLICY_DATAオブジェクトはポリシコントロールを行なう為に用いられる、RSVPのオブジェクトの一つである。POLICY_DATAオブジェクトが含まれたRSVPパケットはPEPへ予約判断が委ねられ、その後PDPでRSVPから受け取った情報について

ポリシ判断を受ける。しかし、PDP によって施行されるポリシは Autonomous System など、ある程度の範囲で共通と考えられる。

共通のポリシが施行される AS などの中には、多くのノードが存在しており、その中には複数の PEP が存在すると考えられる。図 4.1 に示すように、帯域の予約はそれらの PEP 間で行なわれ、同一ポリシに管理される PDP に基づいて RSVP パケットは処理される。

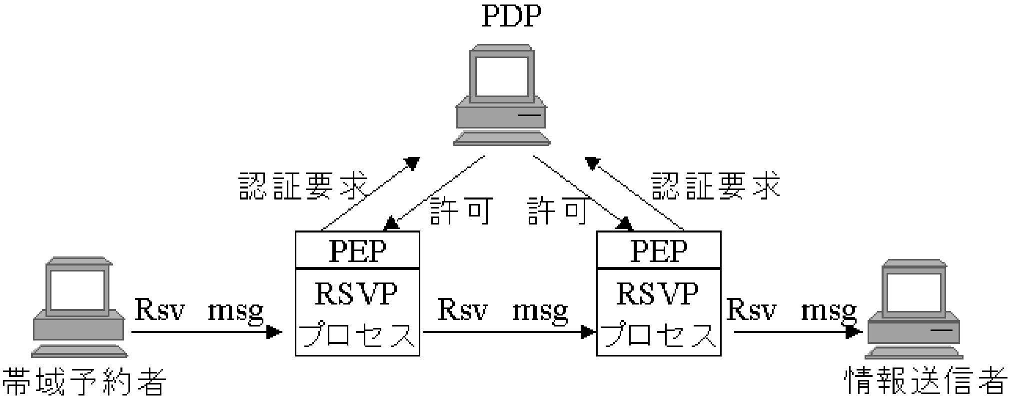

図 4.1: RSVP のポリシコントロールモデル

4.1.2 PPDP_ADDR の提案

前節に述べた RSVP のポリシコントロールモデルとして、図 4.1 に表される、同一ポリシが施行される PEP を RSVP パケットが通過する時を考える。その際、PEP を通過する度にポリシコントロールを受けることは冗長な処理と考えられる。これを避ける為、RSVP メッセージが転送される際、そのメッセージの中にポリシコントロールを受けた PDP のアドレスを含めることとする。

そこで本節では、POLICY_DATA オブジェクト内でそれに必要な policy element を新たに規定し、その利用方法について述べる。

改善されたポリシコントロールモデル

RSVP メッセージ内にポリシ判断を受けた PDP アドレスを含めた場合、RSVP メッセージが次ホップの PEP に到着した時、メッセージに含まれた PDP のアドレスを用いることで処理の軽減を図る。

図 4.2 で表すように、PEP がポリシコントロールを受けるべき PDP が、前ホップの PEP と同じポリシによって管理されるものであった場合、そのメッセージは既に妥当なポリシコントロールを受け、受理されていると判断できる。その際は、PEP は PDP に対して予約要求の判断を行なう必要がなく、PEP と PDP の間で行なわれる予約判断のが不要になる。

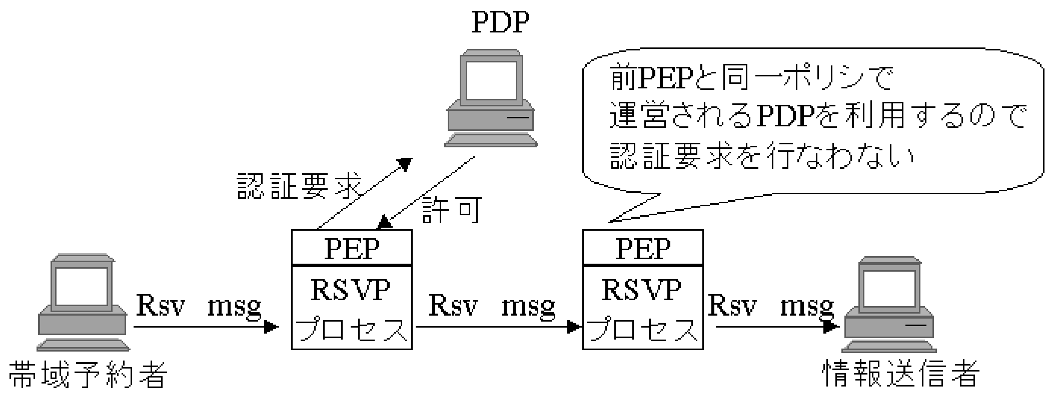

図 4.2: 改善されたポリシコントロールモデル

PPDP_ADDR の設計

次に、ポリシ判断を受けた PDP アドレスを、RSVP メッセージ内に含める方法を考える。前節までに述べたように、ポリシ判断を受けた PDP のアドレスを含めて次ホップの PEP へと RSVP メッセージを転送することは、冗長なポリシ判断を避ける為に役立つと考えられる。そこで、RSVP の POLICY_DATA 内に新たな policy element を規定し、その中にポリシ判断を受けた PDP アドレスを記述する。

ここで新たに、ポリシ判断を受けた PDP アドレスを格納する為の policy element を PPDP_ADDR(Previous PDP Address) と名付ける。policy element type(P-type) は 2 とし、policy information 部にはポリシ判断を受けた PDP アドレスを格納する。格納

するアドレスがIPv4とIPv6のどちらかを記述する為にpolicy element subtypeも規定し、IPv4アドレスの場合は1、IPv6アドレスの場合は2とする。

図4.3にPPDP_ADDRのデータフォーマットを示す。

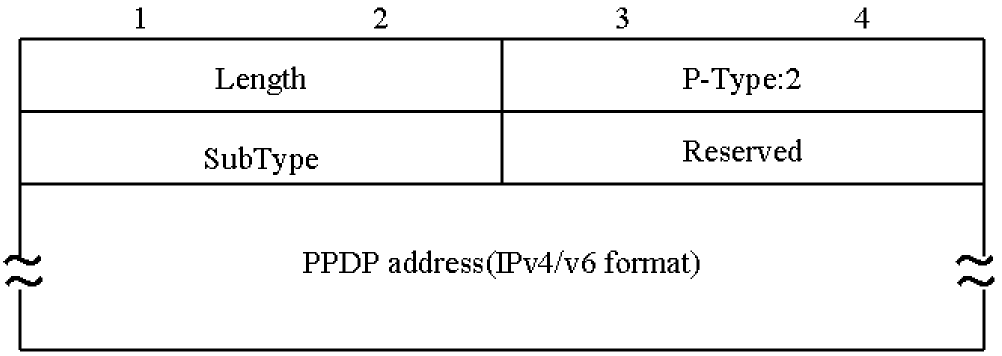

図 4.3: PPDP_ADDR データフォーマット

Length： 16 ビット。
Object headerを含むオブジェクト全体のバイト数

P-Type： 16 ビット。
PPDP_ADDRを示すP-Typeは2

SubType： 16 ビット。

1 = IPv4 address

2 = IPv6 address

Reserved： 16 ビット。
予約済み。常に0にセットされる。

4.2 PDPの設計

PDPは、PEPから入力された帯域予約者の情報に基づいて予約の判断を行なう。本節ではその際の帯域予約者情報の処理手順を述べ、PEPから転送されたメッセージの内容をPDPが保持する予約リストと比較し、予約判断を行なう方法について述べる。

4.2.1　PEPからのメッセージの正当性確認

PDPは起動と共にクライアントであるPEPからの入力を待ち続ける。現在の設計では、PDPがサポートするクライアントタイプはRSVPのみであり、その他のクライアントタイプは考慮しない。PDPが入力待ち状態の時に、PEPからクライアントタイプがRSVPであるメッセージが転送された時の対応について考える。

まず、受け取ったメッセージが不正なものではないことを確認する為、PDPはまずRSVPメッセージ内のINTEGRITY OBJECTを確認する。その後、入力データがPDPに到着した時間とのずれをTIME STAMPによって確認する[6]。

INTEGRITY OBJECTの確認が失敗した時は、パケットが最初に生成された時の状態から変更されていることを示し、不正なものであるのでそのメッセージは却下される。また、TIME STAMPと入力データがPDPに到着した時間がPDPによって許される範囲を越えている場合、そのメッセージは入力データがコピーされて不正に利用されているものと見て却下する。

4.2.2　予約要求と予約リストの比較

PEPからの入力データが不正なものではないことを確認した後、PDPは入力データの内容と予約リストとの整合性の確認を行なう。

PEPからの入力データはRSVPプロセスからの入力データであり、その中にはPOLICY_DATAが含まれる。PDPではPOLICY_DATAオブジェクトを用いて、ユーザidを暗号化したものと復号化に用いる鍵を取り出す。その後復号化されたユーザidと、PDPが保持する予約リストに記述されている予約者名とを比較し、それらが合致することを確認する。ユーザidが確認された後、入力データのTIME STAMPが予約リストに記述された予約可能時間に含まれることを確認する。

以上のそれぞれの確認に対して整合性が確認できた場合のみ予約許可メッセージをPEPに対して返送し、いずれかに失敗した場合は予約却下メッセージを送信する。

図4.4にPDP内処理の状態遷移図を示す。

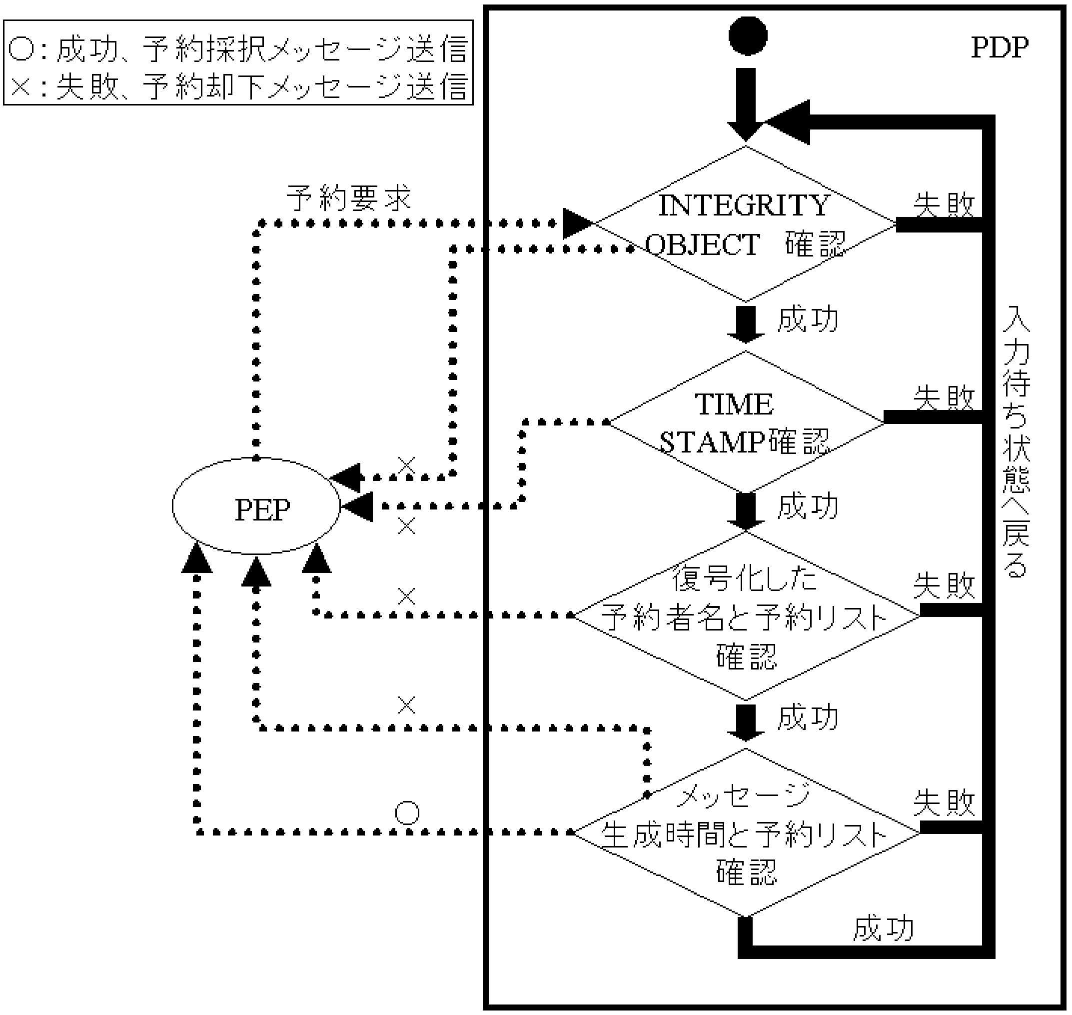

図 4.4: PDP 内処理の状態遷移図

4.3 PEP の設計

PEP は RSVP プロセスからの帯域予約メッセージを元にして予約の可否を PDP と通信して確認する。その際に PEP では PDP に対して予約判断を要求する前にその判断の必要性を PEP 内部で判断する。本節では PEP が RSVP プロセスから帯域予約メッセージを受け取った時の処理方法について述べる。

4.3.1 PPDP_ADDR 検査

PEP は RSVP メッセージを待ち、入力があった時に処理を開始する。検査は RSVP プロセスからの入力データ内に存在する、POLICY_DATA オブジェクト内の PPDP_ADDR から開始する。PPDP_ADDR に記述されている PDP アドレスと PEP がポリシ判断を受ける PDP が同じポリシで運営されていることを検査する。

PPDP_ADDR に記述された PDP アドレスが、PEP が現在接続している PDP と同じポリシで運営されていた場合、その入力データは既に妥当なポリシコントロールを受けていると判断し、その後の処理を行なわずに RSVP プロセスに対してポリシ判断の成功を伝える。

4.3.2 メッセージの正当性確認

PPDP_ADDR に記述された PDP アドレスが PEP が現在接続している PDP と異なったポリシで運営されていたものであった場合、PEP ではポリシ判断に向けての処理を開始する。まず、INTEGRITY オブジェクトの検査を行ない、パケットが不正に変更されていないことを確認する。次に TIME STAMP を検査し、PEP 内で許される範囲の時間内に入力データが生成されたものであると確認する。INTEGRITY OBJECT、TIME STAMP の検査の結果、失敗した場合には RSVP プロセスに対して予約却下を通達する。

4.3.3 キャッシュ走査

RSVP プロセスからの入力データが正当なものであると確認された時、PEP ではキャッシュを走査し、以前にその入力データと等しいデータの到着したかを調査する。PEP 内のキャッシュには、入力データの内容とポリシ判断の結果が記述されている。そして、キャッシュにヒットした時に、そのキャッシュが持つポリシ判断の結果によって RSVP プロセスに対するメッセージの内容を決定する。

そして、RSVP プロセスからの入力データがキャッシュにヒットしなかった場合、PDP に対してポリシ判断を要求し、その結果と入力データをキャッシュに書き込み、ポリシ判断の結果を RSVP プロセスに対して受け渡す。

図 4.5 に以上に述べた PEP の状態遷移図を示す。

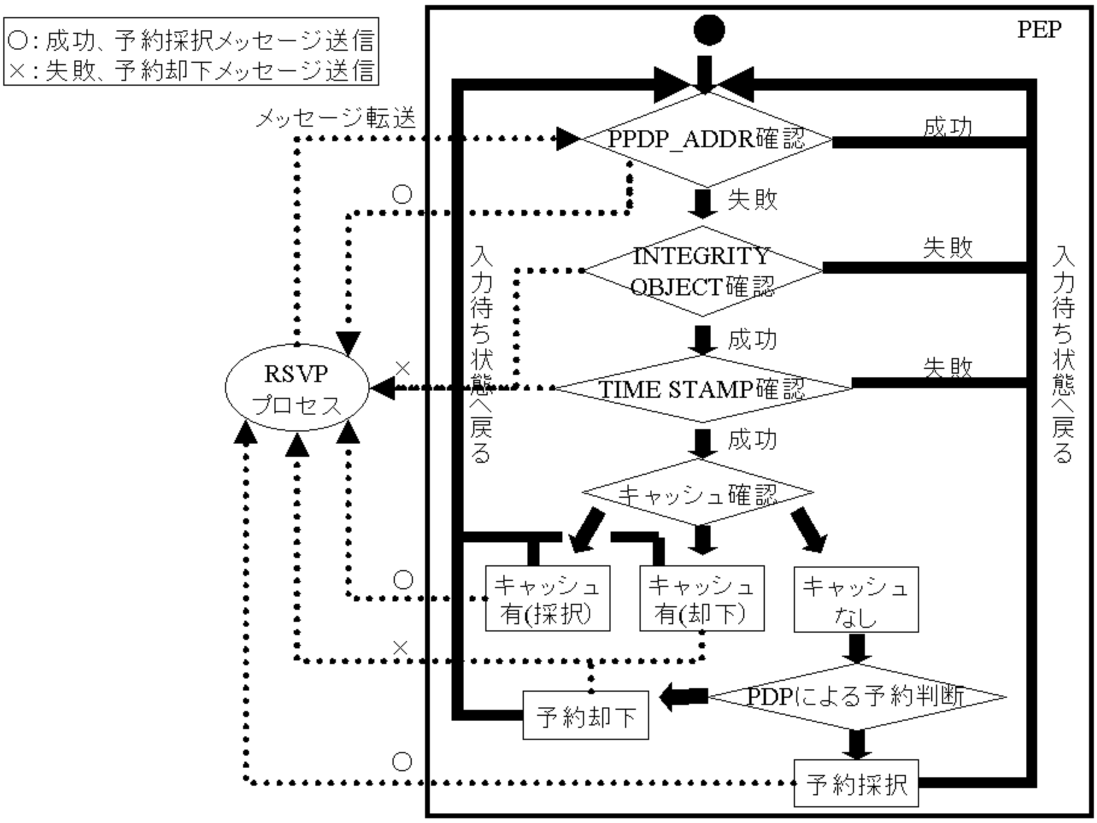

図 4.5: PEP 内処理の状態遷移図

第5章　RSVP利用時の帯域予約者識別環境の実装

本章では[2]に基づいたCOPSの実装と、前章の設計を元にしたPDP及びPEPの実装に関して述べる。また、予約者識別に関して、公開鍵暗号を用いる為、公開鍵、秘密鍵の生成ライブラリ、及び暗号化ライブラリの実装も同時に行なった。その際に、鍵生成、暗号化部分の実装に関しては、THE GNU MP LIBRARY(gmp-2.0.2)を利用した。また、RSVPメッセージ内のINTEGRITY OBJECTを生成する際はMD5メッセージダイジェストアルゴリズムを利用することとし、RSA Data Securityによって配布されているMD5 routinesを利用した。

5.1　COPSの実装

COPSに関する本実装は、[2]に可能な限り忠実に行ない、COPSに規定されたそれぞれのメッセージに対してcops_op_***という名前の関数を実装した。COPSに規定されたそれぞれのメッセージに対応する関数の一覧を表5.1に示す。

表5.1: COPS messageと関数の対応

COPS message	対応する関数
Request	cops_op_request()
Decision	cops_op_decision()
Report State	cops_op_report_state()
Delete Request State	cops_op_delreqstate()
Synchronize State Request	cops_op_syncstatereq()

COPS message	対応する関数
Client-Open	cops_op_clientopen()
Client-Accept	cops_op_clientaccept()
Keep-Alive	cops_op_keepalive()
Client-Close	cops_op_clientclose()
Synchronize State Complete	cops_op_syncstatecomplete()

5.2 POLICY_DATA オブジェクトの構築

本実装において、RSVP メッセージ内の POLICY_DATA オブジェクトは、帯域予約者の認証を行なう為に用いられる。そこで、以下の順序で必要な情報を配置し、PDP、PEP で扱うことができる POLICY_DATA オブジェクトを構成した。

1. POLICY_DATA オブジェクトヘッダ (C-Num:14 C-Type:1)
2. policy element ヘッダ (P-Type:2)
3. PPDP_ADDR
4. INTEGRITY オブジェクト
5. policy element ヘッダ (P-Type:1)
6. AUTH_DATA ヘッダ (AuthMethod:1)
7. ATTRIBUTE LIST ヘッダ (A-Type:8)
8. TIME STAMP
9. ATTRIBUTE LIST ヘッダ (A-Type:1 SubType:1)
10. USER_ID(平文)
11. ATTRIBUTE LIST ヘッダ (A-Type:1 SubType:3)
12. USER_ID(暗号文)
13. ATTRIBUTE LIST ヘッダ (A-Type:3 SubType:3)
14. USER_CRED

5.3 暗号化関数の実装

本節では POLICY_DATA 内に含まれる暗号化された USER_ID を用いる為、暗号化に用いる鍵生成関数と暗号化関数の実装に関して述べる。

5.3.1 鍵生成関数

本システムでは、予約者識別の際に行なう認証の為に公開鍵暗号を用い、そのアルゴリズムにはRSA暗号アルゴリズムを利用した。RSA暗号は鍵を生成する際に、扱う数字が大きいほどその安全性が高まる。そこで一般に計算機が扱う整数型の範囲を越えた値を扱う為にGNU MP LIBRARYを利用した。

RSA暗号では二つの素数を元に暗号化鍵を生成するが、その素数の桁数は多い方がよい。本実装ではそれぞれ10進数で100桁ほどの素数を用いる。それを元に公開鍵、暗号鍵をそれぞれ求め、ファイルに保存する。但し、プログラムを起動した時点で起動者が鍵のファイルを所持している場合、鍵を求める処理は行なわず、メッセージを残して処理を終了する。

以上の考えを元にした、暗号化に必要な手順は次のようになる。

1. 鍵を格納するファイルが存在することを確認する
2. 乱数を発生させ、それを元に素数の組(p, q)を求める
3. 二つの素数を掛け合わせて鍵の要素の一つ(n)を求める
4. $LCM(p-1, q-1) = L$を計算し、Lと互いに素な数(e)を求める
5. $ed = 1(\mathrm{mod} L)$となる数(d)を求める
6. 公開鍵は(e, n)、秘密鍵は(d, n)となる
7. それぞれの鍵をファイル(secret_key, pub_key)に保存する

5.3.2 暗号化、復号化関数

次に、生成した鍵を用いた暗号化について述べる。平文Mを暗号文Cへ変換する式は式5.1のようになる。

$$C = M^e \bmod n \tag{5.1}$$

また、暗号文の復号化は式5.2のように行なわれる。

$$M = C^e \bmod n \tag{5.2}$$

式5.1と式5.2の違いは、鍵だけであり、実際には一つの関数encipherで暗号化、復号化を実現した。

本実装では暗号化をPOLICY_DATAオブジェクトの生成を行なう際に用い、復号化はRSVPプロセスから入力されたCLIENTSIオブジェクトの検査をする際に用いる。暗号化を行なう際はRSVPメッセージ作成者の持つ鍵ファイルを元にして暗号化を実行し、復号化を行なう際はCLIENTSIオブジェクト内に格納された復号鍵を元に復号化を行なう。

そこで、本実装では暗号化を行なう時、引数として暗号化するメッセージと秘密鍵を用いて暗号化を行なうことを知らせ、ファイルから鍵を得て暗号化を行なう。また、復号化を行なう際は引数として復号化するメッセージと公開鍵を用いて復号化することを知らせ、更にCLIENTSIオブジェクト内から抽出した鍵を引き渡して復号化を行なう。

5.4 PDPの実装

PDPは起動と共にPEPからの接続を待ち続ける。PEPからデータの入力が行なわれた際にPDPは子プロセスを生成し、その後の処理をそのプロセスに任せる。

5.4.1 Client-Openメッセージ受信時の動作

PEPは始めにClient-Openメッセージを送信してくることが期待される。PDPは、Client-Openメッセージに対し、rcv_op_clientopen関数を呼び出し、問題がなければClient-Acceptメッセージを、メッセージに何らかの問題が存在する時はClient-Closeメッセージを送信する。

また、PEPからのClient-OpenメッセージにはPEPIDオブジェクトが含まれる。そこでPDPは、Client-AcceptメッセージをPEPに対して送信する前にPEPIDオブジェクトを検査する。本実装ではPEPIDオブジェクトに記述されるものはPEPのIPv4アドレスであり、PDPは読み込んだアドレスに記入されたPEPがRSVPメッセージを示すクライアントタイプ(COPS_CLIENT_RSVP_ROUTER)をサポートすることを知る。

そこでPDPは、PEPIDオブジェクトに記載されたPEPアドレスと、COPSメッセージヘッダーに記述されたクライアントタイプを、PDPが管理するPEPのリストファイル(pdp_known_list_pep)に書き加える。図5.1にPEPからClient-Openメッセージを受け取った時のPDPの動作を示す。

5.4.2 Requestメッセージ受信時の動作

本節ではPDPがPEPからRequestメッセージを受け取った時の動作を示す。RSVPクライアントからのRequestメッセージの場合、その中にはRSVPパケットが格納されている。そこでPDPではヘッダー部を取り除き、RSVPパケットを取り出し、POLICY_DATAオブジェクトを検査する。

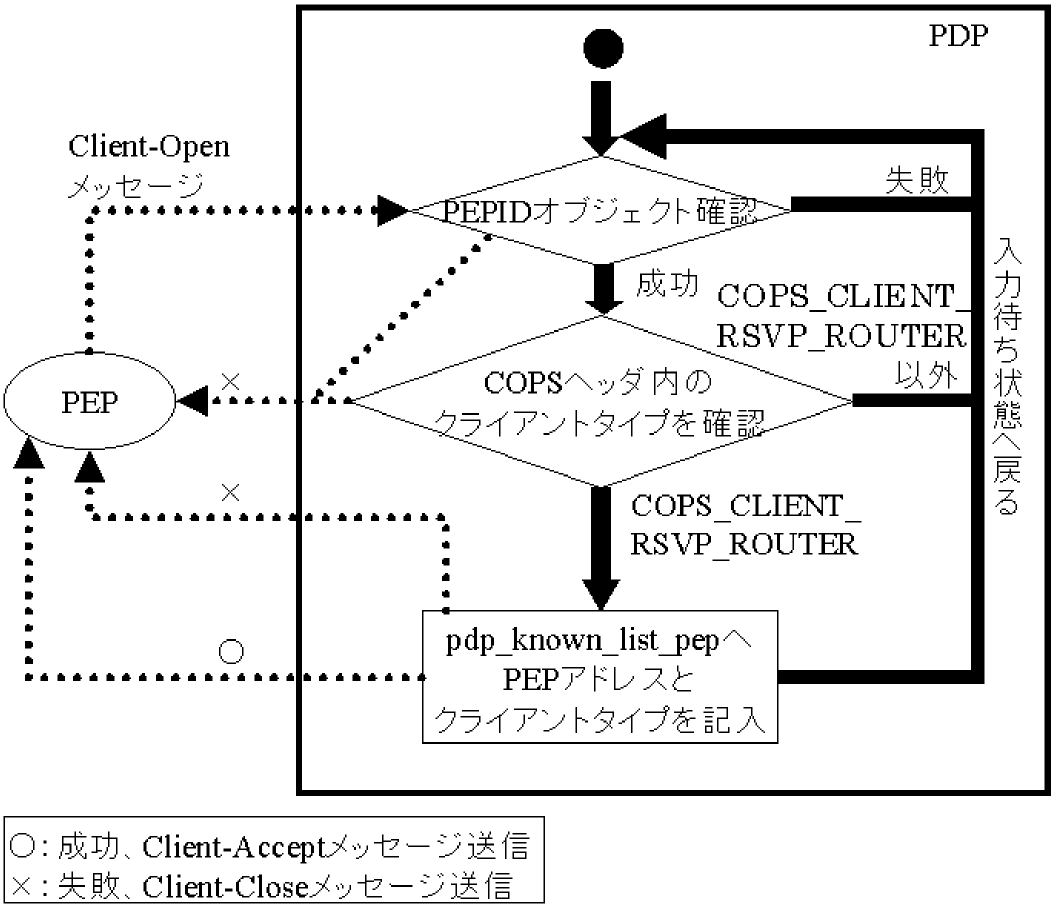

図 5.1: Client-Open メッセージ受信時の動作

メッセージの正当性の確認

POLICY_DATA オブジェクトを検査する際には、まず INTEGRITY オブジェクトを取り出してメッセージが不正に変更されていないものであるかを確認する。実際に INTEGRITY オブジェクトの中に格納されている情報は、ハッシュ関数である MD5 によって元のメッセージから生成されたものである。

そこで、メッセージが改ざんされていないことを確認する為、INTEGRITY オブジェクトを除いたメッセージを MD5 関数に渡し、その結果と INTEGRITY オブジェクト内の情報とを比較する。比較した結果、INTEGRITY オブジェクト内の情報と等しい場合はメッセージが正当なものであると判断し、異なっている場合は不正にメッセー

ジが改ざんされていると判断する。

次にRSVPのPOLICY_DATAオブジェクト内のpolicy elementの一つ、authentication attribute list[4]の中に規定されているTIME_STAMPを利用し、現在時刻と比較することで、メッセージがある一定の時間内に生成されたものであると確認する。

予約リストとの整合性確認

それらのメッセージの正当性の確認に成功した場合、メッセージ内の情報とPDP内に保持されている予約リスト(pdp_resv_list)との間の整合性を検査する。現在予約リストの中に書き込める情報は以下の3種類である。

- 予約開始時刻
- 予約終了時刻
- 予約者

予約リストの記述方式の例を以下に挙げる。

01 02 03 04 01 02 04 05 who

この例では、1月2日3時4分から1月2日4時5分までwhoが予約可能であるということを意味している。

上に述べた予約リストの内容を用いて、PEPから入力されたメッセージの情報を比較するが、比較の順序としては、まず予約者名の比較から行なう。PEPから入力されたメッセージの中には、予約者の秘密鍵によって暗号化された情報が格納されている。そこで、暗号化された情報をencipherで復号化し、その結果と予約リスト内に保持された予約者名とを比較する。

予約者名の比較が失敗した場合、予約リストの次のエントリに移って比較を行なうが、予約者名が予約リストの内容と一致した場合、次にTIME STAMPと予約リスト内の予約可能時間との比較を行なう。TIME STAMPの情報が予約可能時間内に入っていた場合、予約要求は採択される。予約リストの最後まで要求された予約との整合性がとれなかった場合、予約要求は却下される。

req_state構造体

また、予約が採択された場合、その予約情報はPDP内のreq_state構造体に保存され、以後の予約情報の変化に備える。図5.2にreq_state構造体を、図5.3にRequestメッセージ受信時のPDPの動作を示す。

```
typedef struct req_state {
        struct req_state  *next_req_state;
        struct in_addr     pep_addr;
        struct timeval     handle;
        int                from_month;
        int                from_day;
        int                from_hour;
        int                from_min;
        int                to_month;
        int                to_day;
        int                to_hour;
        int                to_min;
        char               user_id[MAX_ID_LEN];
}     REQ_STATE;
```

図 5.2: req_state 構造体

5.4.3 Client-Closeメッセージ受信時の動作

本項ではPEPからClient-Closeメッセージを受信した時の動作について述べる。

Client-CloseメッセージはPEPが終了する時にPDPに対して送られる。PDPはそのメッセージを受け取った後、Client-Openメッセージを受け取った時にPEPリストのファイル(pdp_known_list_pep)に書き込んだエントリを消去する。その際に、PEPに対応するPDPのプロセスが保持するPEPアドレスとClient-Closeメッセージ内に格納されるClient-Typeを元にPEPのリストファイルを走査する。

PEPリストのファイルと該当するPEPアドレス、Client-Typeを比較、削除する時、PDPはPEPリストを格納する為の仮ファイルを作成し、そのファイルへ従来のファイルからのリストを転写して行く。その際に該当するエントリがあった場合、その部分は仮ファイルに書き込まれず、次のエントリの書き込みに移っていく。

その結果、仮ファイルには該当するエントリを除いた元ファイルの内容が書き込まれており、最終的に仮ファイルの名前を元ファイルの名前に変更してエントリ削除は終了する。その後、PEPからのClient-Openによって生成されたPDPの子プロセスは終了し、Client-Closeメッセージを受け取った時の処理を終了する。

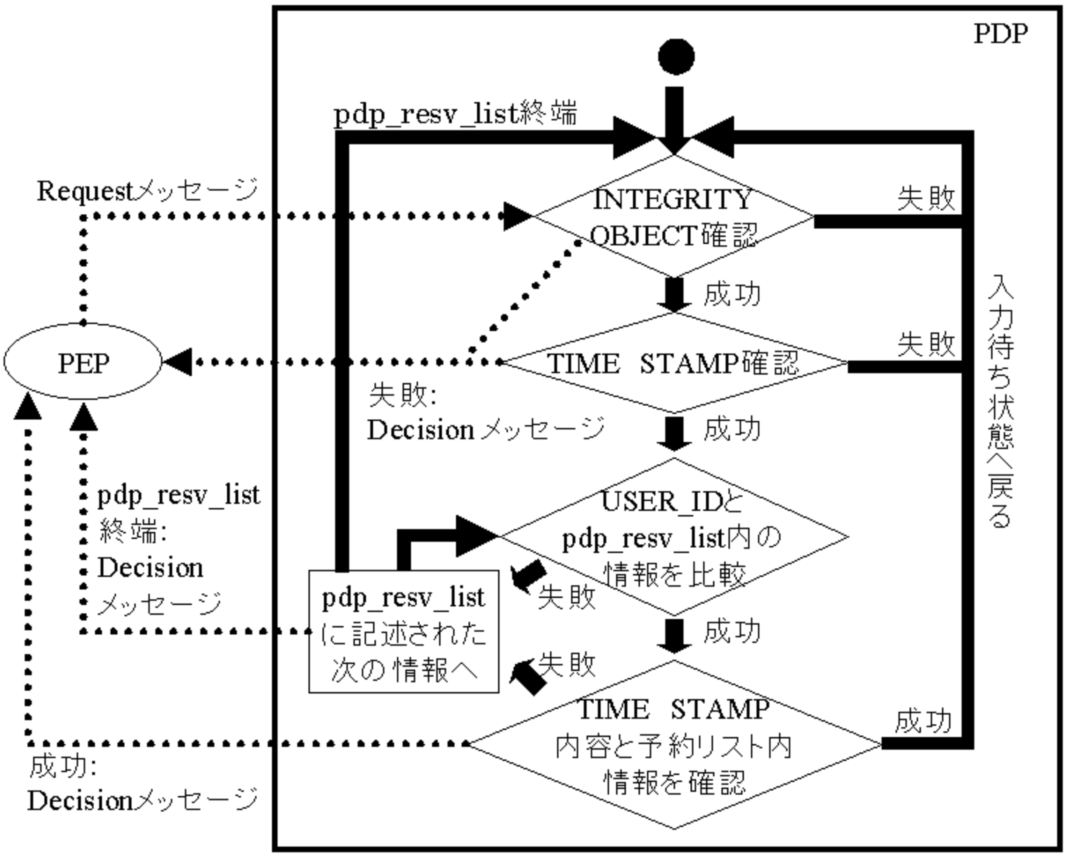

図 5.3: Request メッセージ受信時の動作

5.5 PEP の実装

PEP は、RSVP プロセスからのメッセージを受け取った時に処理を開始し、必要に応じて PDP と通信を行なう。

5.5.1 RSVP メッセージ到着時の動作

メッセージの正当性の確認

PEP に対して RSVP プロセスからのメッセージが入力された時、PEP はそのメッセージの解析を行なう。まず PEP は受け取ったメッセージから POLICY_DATA 部を取

り出し、その部分が改ざんの可能性と、メッセージが作成された時間からPEPがメッセージを受け取った時間までに許容範囲を越えて時間が経過していないか、ということを調べる。

これらのメッセージの正当性の検査は、5.4.2に記述されたPDPで行なわれるメッセージ正当性の検査と等しい動作である。メッセージが変更されていないことを確認する為にINTEGRITYオブジェクトを利用し、メッセージの作成時刻を知る為にPOLICY_DATA内のTIME STAMPを利用する。

キャッシュ判断

次にPEPは内部に保存されているポリシ判断のキャッシュと入力されたメッセージとの間の整合性を確認する。キャッシュ内に保持される情報を、以下に示す。

- PDPからの判断の結果
- 平文の予約者名
- 暗号化された予約者名
- 予約者の暗号化されたidを復号化する時に用いる鍵
- PDPから転送された、予約内容が格納されているreq_state構造体
- 最後にRSVPプロセスからメッセージを受け取った時間

図5.4に以上の情報を格納する為のキャッシュ構造体を示す。

```
typedef struct cache_table {
        int                  cache_flag;
        char                 user_id[MAX_ID_LEN];
        char                 user_eid[MAX_EID_LEN];
        char                 user_cred[MAX_CRED_LEN];
        REQ_STATE            req_state;
        unsigned long        last_rcv_time;
}    PEP_CACHE;
```

図5.4: キャッシュ構造体

PEP では、PDP に対して Request メッセージを送る前に、キャッシュの内容と比較し、その内容と適合した場合は PDP に対して Request メッセージを送らずにキャッシュ判断処理を終了する。その際は、キャッシュ内に保存される RSVP メッセージを受け取った時間を更新し、予約採択か却下かのどちらかのキャッシュ判断の結果が伝えられる。

また、PEP 内に保存されているキャッシュ情報の全てと適合しなかった場合はキャッシュ内に含まれる RSVP メッセージを受け取った時間を元に、最も利用されていないキャッシュの情報を新たに書き換え、PPDP_ADDR の検査を行なってどのキャッシュを書き換えたかを PEP 内に保持する。図 5.5 にキャッシュ判断処理の動作を示す。

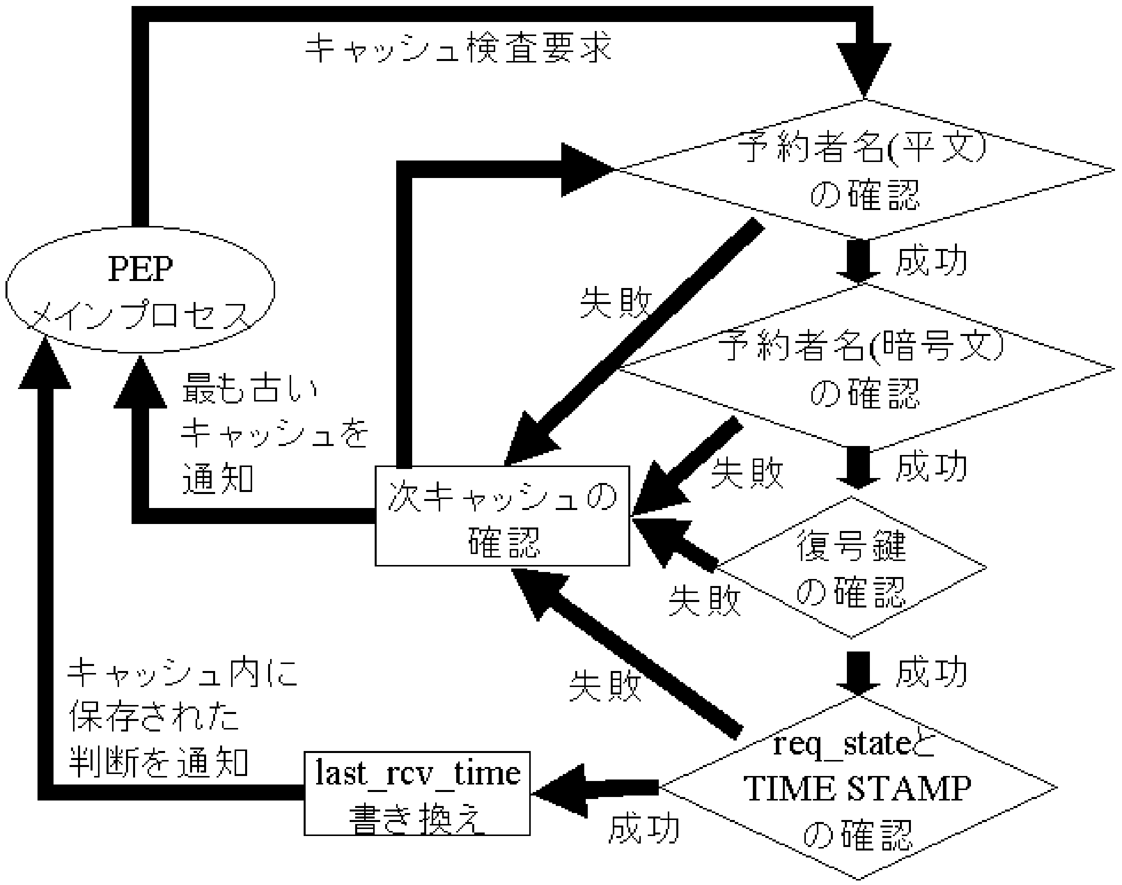

図 5.5: キャッシュ判断処理の動作

5.5.2 PPDP_ADDR の検査

キャッシュ判断の結果、該当するキャッシュを発見できなかった場合、POLICY_DATA オブジェクトに含まれる AUTH_DATA 以外のもう一つの policy element である PPPD_ADDR を検査する。

PEP は、まず現在接続している PDP アドレスと PPDP_ADDR に含まれるアドレスを比較する。それに失敗した場合、PEP が保持する、現在接続している PDP と同じポリシで運営されている PDP のアドレスを記述するファイル (pep_pdplist) を確認する。もしその中に PPDP_ADDR に含まれるアドレスが存在していた場合は PPDP_ADDR の検査は成功したものとする。

PPDP_ADDR の検査が成功した場合、予約は採択されたものと判断できる。そこで PEP 内に保存される書き換えるべきキャッシュの、予約判断を記述する部分に予約採択を書き込み、RSVP プロセスから送られたメッセージを返送する。

5.5.3 Decision メッセージ受信時の動作

PDP から送信される Decision メッセージは、単純に

- 予約採択
- 予約却下

の二種類の判断結果を含んでいる。そこで、PEP ではこれら二種類の PDP による決定に従って処理を行なう。

予約採択メッセージ受信時

PDP が PEP に対して予約採択メッセージを送信した時、PEP はその結果をキャッシュに書き込まなければならない。PEP 内に保持された書き換え対象のキャッシュ情報のうち、以下の二つの情報を書き換える。

- PDP からの Decision メッセージに格納された判断の結果
- PDP から転送された、予約内容が格納されている req_state 構造体

その後、PPDP_ADDR を書き換え、RSVP プロセスに対して新たなパケットの内容を転送する。

予約却下メッセージ受信時

PDPから送信されたDecisionメッセージの結果が予約却下の場合、PEPは予約採択メッセージ受信時と同様にその結果をキャッシュに書き込まなければならない。但し、RSVPプロセスへ返送すべきメッセージは、ポリシ判断の結果が予約却下であることを伝えるだけなので、PPDP_ADDRの書き換えは行なわずRSVPプロセスに対してエラーコードを転送する。

5.5.4 Client-Closeメッセージ受信時の動作

Client-CloseメッセージをPDPから受信した場合、通常PEPはそれに伴って終了する。しかし、Client-Closeメッセージ内にPDP Redirect Addressオブジェクトが含まれている場合、PEPは単純に終了しない。

PDP Redirect Addressオブジェクトに含まれるアドレスは、それまでPEPが接続していたPDPと同じポリシで運営されるPDPである。そこで、PEPはPDP Redirect Addressオブジェクトに含まれたアドレスに対して新たに接続し、Client-Openメッセージを送信する。

そして、Client-Openメッセージが受理された場合、PEPはそれまでと変わらずにRSVPプロセスからの入力を待ち続ける。但し、失敗した場合はPDP Redirect Addressオブジェクトがない時と同様にPEPは正常に終了する。

5.6 実装環境

本節では実装を行なった環境について表5.2に記す。

表5.2: 実装環境

項目	説明
OS	FreeBSD2.2.7R
実装言語	C (gcc-2.7.2.1)
RSVPパッケージ	ISI rel4.2a3
GNU MPパッケージ	gmp-2.0.2

第6章　評価

本章では前章までに設計、実装を行なったRSVPの為の予約者識別環境について性能の評価を行なう。

6.1　測定環境

本システムの評価を行なう為に用いた測定環境を図6.1に記す。ホストAはPEPとして動作し、ホストBはPDPとして動作する。

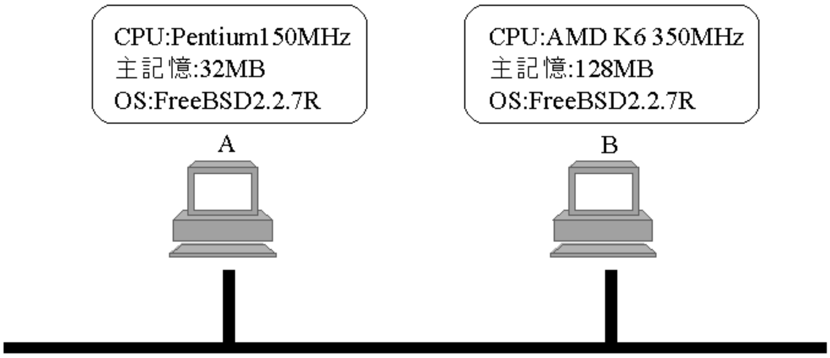

図 6.1: 測定環境

6.2　機能評価

本システムは現状のRSVPを拡張し、帯域予約者の認証を可能にした。表6.1に現状のRSVPと、本システムを用いたRSVPとの特徴比較を記す。

表 6.1: RSVP と本システムを用いた RSVP との機能比較

評価項目	RSVP のみ	RSVP と本システム
アドミッションコントロール	可	可
帯域予約者の認証	不可	可
ポリシコントロール	不可	拡張可

6.3 暗号化鍵生成関数の評価

本節では暗号化鍵生成関数の評価を行なう。暗号化鍵の生成は 6.1 節に示した Host A で行なった。暗号化鍵を生成する際には、鍵の生成に用いる素数の桁数かを指定できる。表 6.2 に鍵生成の為に用いた素数の桁数と鍵生成に要した時間の平均 (500 回実行) 値、最大値、最小値を記す。

表 6.2: 暗号化鍵生成に要する時間と鍵生成に用いた素数桁

素数桁	鍵生成に要した時間の平均 (秒) host A/host B	鍵生成に要した時間の最大 (秒) host A/host B	鍵生成に要した時間の最小 (秒) host A/host B
10	0.025/0.006	0.056/0.018	0.008/0.004
20	0.052/0.016	0.535/0.216	0.012/0.005
30	0.119/0.041	0.755/0.184	0.021/0.007
40	0.260/0.084	1.031/0.320	0.033/0.013
50	0.475/0.160	4.078/0.616	0.047/0.019
60	0.843/0.283	3.641/1.251	0.069/0.021
70	1.417/0.442	8.142/2.067	0.131/0.040
80	2.125/0.700	8.794/3.228	0.158/0.049
90	3.094/1.076	12.391/5.964	0.142/0.842
100	4.548/1.417	20.516/7.251	0.307/0.135
110	6.283/2.064	31.055/9.387	0.334/0.186
120	8.544/2.876	35.865/11.330	0.550/0.153

鍵生成の為に必要な時間は生成する鍵の生成に用いる素数の桁数によって変化する。これは素数を求める際に必要とされる時間が、求める素数の桁数が増えることで変化する為である。但し、一度生成された鍵はその後繰り返し用いられる為、生成に時間

はかかるものの、生成する鍵の元になる素数桁は大きい方が望ましい。又、鍵生成に要する時間は生成を行なうホストの能力によって変わる為、鍵生成を能力の高いホストで行ない、生成された鍵を使用するホストに移動して利用することが効率的である。

6.4 予約要求処理の評価

PEP はRSVP プロセスからのメッセージに基づいて処理を行なうが、処理を軽減する為の方法として以下の二種類が挙げられる。

- PPDP_ADDR の利用
- キャッシュの利用

これらの処理を軽減する為の方法が、通常PDP に対して行なう予約判断に対して持つ有効性を示す為に測定を行なった。測定項目はPPDP_ADDR 利用時のPEP 処理時間、キャッシュ利用時のPEP 処理時間と、PDP と通信を行なう時の処理時間とした。PEP 内での処理の流れは図4.5 を参照のこと。また、本測定の結果を表6.3 に示す。

表6.3: PEP 内処理の軽減

	平均処理時間 (msec)	最大処理時間 (msec)	最小処理時間 (msec)
PPDP_ADDR 利用時	0.32	0.33	0.24
キャッシュ利用時	1.14	1.23	1.04
通常時	174.21	318.68	126.90

表6.3 に見られるように、本節の冒頭に述べた二種類のPEP 内処理の軽減方法は効果的であることが分かった。又、通常時の処理時間にはPEP、PDP 間でのRequest、Decision メッセージ交換に要する通信時間を含む為、ネットワークの状況によって所要時間が更に増えることも考えられる。

第7章　結論

7.1　まとめ

RSVP を用いて帯域予約を行なう時、これまではアドミッションコントロールのみによって予約の可否が決定されていた。これは、既存の RSVP にポリシコントロール機構が実装されていない為である。

現在の RSVP では上に挙げた理由によって、悪意ある者によって帯域が占有される可能性があった。そのため現在の RSVP は未だに実用段階にあるとはいえず、この問題を避ける為に RSVP 利用時の帯域予約者識別環境を構築する必要があった。

本研究では、帯域予約の可否を判断する際に情報を交換する為のプロトコルとして、COPS を利用した。COPS は、提案はなされているものの実装は公開されていない為、[2] に従って COPS の実装を行なった。

また、RSVP においてポリシコントロールを利用する際に用いる POLICY_DATA オブジェクトも、現在提案されているものの実現はされていない。そこで現在の RSVP を拡張し、POLICY_DATA オブジェクトの利用を可能にした。更に、現在提案されている POLICY_DATA オブジェクトは情報を伝達する際に、その処理時間を軽減する方法について考慮されていない為、新たな policy element として PPDP_ADDR を提案、実装し、その効果を確認した。

次に POLICY_DATA オブジェクトを利用する際にそのオブジェクトの不正な再利用を防止する為に用いる暗号を形成する為の関数と、その暗号に用いる鍵を生成する関数を実装した。

最後に、これらの枠組を用いて RSVP と共に利用する為の PEP、PDP の設計、実装を行なった。本研究によって構築された PEP、PDP によって、RSVP を利用する際に帯域予約者の識別が可能になった。帯域予約者の識別は、ポリシコントロールを行なう為の最初の段階であり、本研究によって構築された環境を拡張することで RSVP を利用する際のポリシコントロールが可能になる。

RSVP を利用する際にポリシコントロールの実行が可能になることで、RSVP 利用時にネットワーク管理者による選択的な帯域予約許可が可能になる。それにより、帯域の不当な占拠を防止することができ、帯域資源の管理や予約サービスを行なうことが可能になる。

7.2　今後の課題

本研究で構築されたシステムはRSVP内のPOLICY_DATAオブジェクトを用いた認証と、PDP内に保存された予約時間と受け取ったメッセージの時間が合致するかという二点についてのみ考慮している。

今後RSVPでポリシコントロールを行なう為には、RSVPメッセージ内に含まれる予約情報を元にして帯域予約希望者がその予約情報に書かれた帯域幅を予約する権利を持つかを判断していく必要がある。RSVPにおけるポリシコントロールはこのような条件を満たすことで完成する。

又、本システムで新たに提案したpolicy elementであるPPDP_ADDRについても考慮する必要が残されている。PPDP_ADDRは、それがPEP内の情報と合致するとそのPEP内でのポリシ判断を行なわないことによって処理時間の軽減を目指している。しかし、PPDP_ADDRはその有効性の裏に危険性も含んでいる。

PPDP_ADDRが不正に変更された場合、現在のPPDP_ADDRではPEP内でそれを関知する方法がない。そのため、PPDP_ADDRが不正に変更されたものであっても、それが合致した時はポリシコントロールを行なわずに予約が採択されたものと判断されてしまう。よって、PPDP_ADDRを利用する際にはその安全性を高めていく必要がある。

付録A　COPS オブジェクト

付録 A として、文献 [2] に記された COPS オブジェクトそれぞれについての構造体とその内容について述べる。COPS オブジェクトはヘッダ部とオブジェクト内容の部分で構成されるが、ここでは COPS オブジェクトヘッダ以外の部分を記す。

A.1　Handle オブジェクト (Handle)

予約状態を識別する為に用いる。Handle オブジェクトは PEP によって指定され、個々の予約状態毎に異なる値が用いられる。本実装では PEP が予約要求パケットを受け取った時間を用いた。

C-Num :　1

```
typedef struct tag_Handle {
        struct timeval  handle;
}    COPS_HANDLE;
```

A.2　Context オブジェクト (Context)

Context オブジェクトはどのような要求によって COPS メッセージが生成されたかを記述する。

C-Num :　2

```
typedef struct tag_Context {
        unsigned short rType;   #(R-Type)
        unsigned short mType;  #(M-Type)
}    COPS_CONTEXT;
```

R-Type : 16 ビット。
Request の種類を示す。R-Type の種類は以下のものである。

0x01 = Incoming-Message request

0x02 = Resource-Allocation request

0x04 = Outgoing-Message request

0x08 = Configuration request

M-Type : 16 ビット。
それぞれのクライアント毎に定められる。

A.3 In-Interface オブジェクト (IN-Int)

要求を受け入れたインターフェイスを記述する。

C-Num : 3

C-Type : 1, IPv4Address + Interface

```
typedef struct tag_INTERFACEV4 {
        struct in_addr addrV4;  #(IPv4 Address)
        unsigned long ifindex;   #(interface index)
}    COPS_INTERFACEV4;
```

C-Type : 2, IPv6Address + Interface

```
typedef struct tag_INTERFACEV4 {
        struct in6_addr addrV6;  #(IPv6 Address)
        unsigned long ifindex;     #(interface index)
}    COPS_INTERFACEV4;
```

A.4 Out-Interface Object(OUT-Int)

要求が送信されるインターフェイスを記述する。このオブジェクトのフォーマット及び C-Type の定義は IN-Int オブジェクトと等しい。

C-Num : 4

A.5 Reason オブジェクト (Reason)

このオブジェクトは何故 PDP 内に保存された予約状態が破棄されたのかを示す。

C-Num : 5

```
typedef struct tag_REASONCODE {
        unsigned short reasonCode;       #(Reason-Code)
        unsigned short reasonSubCode;  #(Reason Sub-Code)
}    COPS_REASON;
```

Reason-Code : 16 ビット。
予約状態が破棄された理由を示す。

1 = Unspecified

2 = Management

3 = Preempted

4 = Tear

5 = Timeout

6 = RouteChange

7 = Insufficient Resources

8 = PDP's Directive

9 = Unsupported decision

10 = Synchronize Handle Unknown

11 = Transient Handle

12 = Malformed Decision

Reason Sub-code : 16 ビット。
将来的に、クライアント毎のより詳細な予約破棄の理由を記述する。

A.6 Decision オブジェクト (Decision)

PDP による決定を記す為に用いる。

C-Num : 6

C-Type : 1, Decision Flags

```
typedef struct tag_DECISIONFLAGS {
        unsigned short commandCode;  #(Command-Code)
        unsigned short flags;        #(Flags)
}    COPS_DECFLAG;
```

Command-Code : 16 ビット。
決定事項を記す。

0 = NULL Decision

1 = Install

2 = Remove

Flags : 16 ビット。
現在は予約失敗を示すフラグのみが定義されている。

0x01 = Trigger Error

C-Type :

2, Stateless Data

3, Replacement Data

4, Client Specific Decision Data

5, Named Decision Data

A.7 LDP Decision オブジェクト (LDPDecision)

PEP 内に存在する LDP による決定を記す為に用いる。LDPDecision に用いるフォーマットは Decision オブジェクトと変わらない。

C-Num : 7

A.8 Error オブジェクト (Error)

このオブジェクトは COPS プロトコルのエラーを示す。

C-Num : 8

```
typedef struct tag_ERR {
        unsigned short errCode;      #(Error-Code)
        unsigned short errSubCode;  #(Error Sub-Code)
}    COPS_ERR;
```

Error-Code : 16 ビット。
COPS プロトコルのエラーを示す。

1, Bad handle

2, Invalid handle reference

3, Bad message format

4, Unable to process

5, Mandatory client-specific info missing

6, Unsupported client-type

7, Mandatory COPS object missing

8, Client Failure

9, Communication Failure

10, Unspecified

11, Shutting down

Error Sub-Code : 16 ビット。
クライアントタイプ毎の詳細なエラーを記述する。

A.9 Client Specific Informationオブジェクト(ClientSI)

COPS を用いるクライアントに依存した情報を格納する為に利用する。

C-Num : 9

C-Type :

1, Signaled ClientSI

2, Named ClientSI

A.10 Keep-Alive Timerオブジェクト(KATimer)

KA TimerオブジェクトはCOPSメッセージが送信、受信される最大の間隔を示す。

C-Num : 10

```
typedef struct tag_TIMER {
        unsigned short reserved;  #(Reserved)
        unsigned short timer;     #(KA/ACCT Timer Value)
}   COPS_TIMER;
```

Reserved : 16ビット。
常に0にセットされる。

KA Timer Value : 16ビット。
Keep-Aliveを行なう間隔を指定する。0は無限値を示す。

A.11 PEP Identificationオブジェクト(PEPID)

PDP内でPEPを区別する為に用いられる。PEPIDオブジェクト内に記述される情報はASCII文字かIPアドレスのどちらかを使用する。

C-Num : 11

```
typedef struct tag_PEPID {
      union {
         struct in_addr addrV4;  #(IPv4 Address)
         unsigned short name;    #(ascii name)
      }
}   COPS_PEPID;
```

A.12 Report-Type オブジェクト (Report-Type)

PEP 内の予約状態を PDP に伝える為に用いられる。handle オブジェクトと共に用いてどの予約についての状態かを識別する。

C-Num : 12

```
typedef struct tag_REPORTTYPE {
         unsigned short reportType;  #(Report-Type)
         unsigned short reserved;    #(Reserved)
}   COPS_REPORTTYPE;
```

Report-Type : 16 ビット。
PEP 内の予約状態を示す。

1, Commit

2, No Commit

3, Accounting

4, Installed

5, Removed

6, NotInstall

7, NotRemoved

Reserved : 16ビット。
常に0にセットされる。

A.13 PDP Redirect Addressオブジェクト(PDPRedirAddr)

PDPが終了する時にどのPDPに対して再接続を行なうべきかを記述する。

C-Num : 13

C-Type : 1, IPv4Address

```
typedef struct tag_PDPADDRV4 {
        struct in_addr addrV4;  #(IPv4 Address)
}    COPS_PDPADDRV4;
```

C-Type : 2, IPv6Address

```
typedef struct tag_PDPADDRV6 {
        struct in6_addr addrV6;  #(IPv6 Address)
}    COPS_PDPADDRV6;
```

A.14 Last PDP Addressオブジェクト(LastPDPAddr)

PEPがClient-Openメッセージを送る時に、PEPが起動した時以降、どのPDPに対して接続が成功したかを示す。LastPDPAddrオブジェクトに用いるフォーマットはPDPRedirAddrオブジェクトと同じものである。

C-Num : 14

C-Type :

1, IPv4 Address

2, IPv6 Address

A.15 Accounting Timer オブジェクト (AcctTimer)

PDP 内の情報を更新する間隔を指定する。0 はいつでも更新可能であることを示す。オブジェクトフォーマットは KATimer と等しい。

C-Num : 15

ACCT Timer Value : 16 ビット。
PDP 内の情報を更新する間隔を指定する。

謝辞

本研究を進めるにあたり御指導を頂きました、慶應義塾大学環境情報学部 徳田英幸博士、村井純博士、楠本博之博士、中村修博士に深く感謝致します。

また、杉浦一徳氏、小川晃通氏、中澤仁氏、大越匡氏をはじめとする慶應義塾大学徳田、村井、楠本、中村研究会の諸先輩方には大変有益な御助言、御指導、そして励ましを頂きました。最後に、様々な点で支えとなって下さった友人に深く感謝致します。

1999年2月
戸田 晴一郎

関連図書

[1] R.Braden L.Zhang S.Berson S.Herzog S.Jamin, "Resource ReSerVation Protocol (RSVP)", RFC2205, 1997

[2] J.Boyle R.Cohen D.Durham S.Herzog R.Rajan A.Sastry, "The COPS (Common Open Policy Service) Protocol", Internet Draft, 1998

[3] S.Herzog, "RSVP Extensions for Policy Control", Internet Draft, 1998

[4] S.Yadav R.Paddati P.Ford S.Herzog "User Identity Representation for RSVP", Internet Draft, 1998

[5] Yavatkar, "A Framework for Policy-Based Admission Control", Internet-Draft, 1997

[6] F.Baker B.Lindell M.Talwar, "RSVP Cryptographic Authentication", Internet-Draft, 1998