NAIST-IS-DT0061007

博士論文

インターネットにおける高機能サービス実現のための通信データに対する処理の選択機構に関する研究

垣内 正年

2004 年 2 月 6 日

奈良先端科学技術大学院大学

情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士（工学）授与の要件として提出した博士論文である。

論文番号: NAIST-IS-DT0061007

提出者: 垣内 正年

審査委員: 砂原 秀樹
山口 英
藤川 和利

提出日: 2004 年 2 月 6 日

インターネットにおける高機能サービス実現のための通信データに対する処理の選択機構に関する研究*

垣内 正年

内容梗概

ネットワークサービスとは，情報を目的地に送り届ける仕組みだと言える．インターネットは多数の小さなネットワークの集合であるため，通常，情報が目的地に到着するまでには，ネットワーク間を中継する複数のノードを経由する．正しい目的地に情報が送り届けられるためには，各中継点で情報が次の中継点へ正しく転送される必要がある．言い換えると，各ノードは転送先別に情報を分類する必要がある．しかし，インターネットサービスの多様化にともない，情報の分類は複雑化し，複数のサービス間で分類ポリシが複雑に関係するようになった．この結果，サービス間のポリシの関係が不明瞭になり，ネットワーク全体の制御が困難になった．本研究ではこの問題に対し，複数のポリシに対応可能な一般化した分類機構を構築することを目標としている．このために，サービス依存・非依存の分離を中心にモデルを構築し，設計・実装を行い，実証実験を通して検証と評価を行った．

本論文では，従来の分類機構について述べ，複数のサービスが存在するネットワークにおけるポリシの競合について議論する．そのあと，サービスから分類機構を分離したパラメータフィルタの概念を明確化する．パラメータフィルタはデータを分類するデータベースの要素として，フィルタ規則であるレコードパラメータと処理内容であるアクションの組で構成する分類レコードを持つ．パラメータフィルタは入力データの属性値であるキーパラメータを入力すると，キーパラメータとレコードパラメータを比較し，適合したレコードパラメータに対応するアクションを出力する．つまり，データを分類するためにフィルタ規則が記述された分類レコードを分類する．

次に，サービス非依存部分とサービス依存部分の 2 段階のステージに分離したパラメータフィルタ，KUMA's Universal Parameter Filter (KUPF) を提案する．サービス非依存部分の第 1 ステージは，キーパラメータとレコードパラメータの比較方法を定義した分類スキーマに従って，入力データと分類レコードを比較する．サービス依存部分の第 2 ステージは，サービス毎のポリシに従って，第 1 ステージの結果からデータに適用するアクションを保持する分類レコードを選択する．これにより，第 1 ステージは分類スキーマにのみ依存するために汎用的に使用でき，第 2 ステージ内でサービス依存

部分が完結するため，サービス毎のポリシに依存する処理が明確になる．

上記モデルに基づいたフレームワーク，KUPF フレームワークを構築し，DiffServ, MPLS による QoS 保証ネットワークを構築して実証実験を行い，従来の DiffServ 実装と比較した処理性能測定を行った．その結果，本研究の提案モデルが実運用にも耐えうることが示せたが，サービスに特化したデータ分類機構に比べ，処理速度性能が劣ることが明らかになった．

処理速度向上を図るために，パラメータフィルタの第 1 ステージに多次元空間上での効率的な検索手法として知られる R*-tree を応用した．この結果，フレームワークの汎用性を維持しながら処理速度の大幅な向上を達成し，分類フィルタ数が多い場合にサービスに特化した実装を上回る性能を示した．

本研究で提案するパラメータフィルタをネットワークサービスに適用することで，新たなサービスの導入を容易にし，既存のサービスの組み合わせを柔軟なポリシで制御可能となる．これにより，今後のネットワークサービスの多様化・高度化に大きく寄与すると考える．

キーワード: パラメータフィルタ，パケット分類，QoS

---

* 奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DT0061007, 2004 年 2 月 6 日.

A Study on Selection Mechanism of Action on Communication Data which Realizes Highly Functional Services on the Internet†

KAKIUCHI Masatoshi

Abstract

It is thought that network services are the structure which sends information into the destination. Usually, information is transmitted via some nodes on a network, in order to send information into the destination, because the Internet is an aggregating of small networks. Each node relaying over inter-networks should correctly transmit to the next node, in order to send information into the right destination. In other words, each node needs to classify information according to a transmission node. However, diversification of the Internet services made the classification of information complex, and classification policies have complicated relation among multiple services. Consequently, the relation of the policies among services became not clear, and control of the whole network became difficult. In my study, in order to solve this problem, it aims at building the normalized classification mechanism which can respond to multiple policies. For this reason, I built the model focusing on separation of service dependence and not depending, designed and implemented to perform verification and evaluation through the actual proof experiment.

In this paper, I discuss conventional classification mechanisms and conflict of policies on network with multiple services. Then I clarify the concept of the parameter filter of having separated the classification mechanism from service. The parameter filter contains a pair of record parameters which mean a filter rule and an action which means contents of processing as an element of a database to classify data. The parameter filter compares inputted key parameters which are attributes of input data and record parameters, and outputs action corresponding to

the record parameter which adapted. That is, the parameter filter classifies the classification record describing the filter rule in order to classify data.

I propose KUMA's Universal Parameter Filter (KUPF), to divide the parameter filter into two stages of a service independence portion and a service dependence portion. Stage 1 non-depending on service, compares inputted data and classification records, based on the classification schema which specifies comparison method of key parameters and record parameters. Once the schema is defined, Stage 1 proceeds automatically, regardless of the service for which these classification records are used. Stage 2 depending on service, selects classification record which contains action applied to inputted data, from results of Stage 1. This 2-phase selection model of classification records exposes the parts depending on the service or its implementation.

I implemented a framework based on 2-phase selection model, built the network which provides QoS using Diffserv and MPLS, and evaluated availability and performance compared with Diffserv. This shows that, my model can apply to actual network services, but a processing speed performance is inferior to conventional parameter filter which specialized in one service.

In order to improve processing speed, I applied R*-tree which is known efficient search method on multi dimensional space. As a result, the framework has improved processing speed sharply with maintaining the flexibility of the framework and showed the performance exceeding implementation which specialized in service, when many classification filters were contained.

By applying the parameter filter proposed by this paper to network services, we can introduce new services easily and control by the flexible policy about the combination of conventional services. I think that this work contributes to diversification and an advancement of future network services greatly.

Keywords: parameter filter, packet classification, QoS

---

† Doctor Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT0061007, February 6, 2004.

目次

図一覧

表一覧

第1章 序論

1.1. ネットワークサービスとデータ分類

ネットワークサービスとは，情報を目的地に送り届ける仕組みだと言える．通常，情報が目的地に到着するまでには，ネットワーク上の複数のノードを経由する．正しい目的地に情報が送り届けられるためには，各中継点で情報が次の中継点へ正しく転送される必要がある．インターネット上のサービスもこの例外ではない．インターネット上では，情報はパケットという単位に分割され，目的地に送り届けられる．各パケットは宛先としてパケットヘッダに IP アドレスを持つ．インターネットの中継点，ルータは，この宛先をキーに各ルータが持つ経路表を検索することで次の転送先を決定する．言い換えると，ルータは経路表に従って転送先毎にパケットを分類する．このことは，ルータの IP 転送に限らず，プロトコル番号に基づいた ICMP/TCP/UDP のプロトコルの選択，ポート番号に基づいたサービスの選択，HTTP における URL に基づいたコンテンツの選択など，あらゆるネットワークサービスに該当する．このように，ネットワークサービスにとってデータ分類は必要不可欠である．

インターネットは当初，研究用の学術ネットワークであったが，1987 年に世界初の商用インターネット接続サービス，1993 年に日本初の商用インターネット接続サービスが始まり，商用にも用いられる情報基盤となった．それに伴い，ネットワークサービスに対する要求も高度化・多様化してきている．今日のインターネットにおいては，セキュリティ保護のためにネットワーク境界でのファイアウォール設置が一般化している．ファイアウォールはルータ本来のパケットの IP アドレスに基づいたパケット転送を越え，プロトコル番号，ポート番号にも基づいて転送パケットの取捨選択を行う．サービス品質（Quality of Service; QoS）を保証するネットワークでは，特定のフローを構成するパケットの優先処理などを行う．DiffServ[1] ネットワークでは，ネットワーク入口でフィルタ等を用いてパケットを分類し，パケットヘッダ内の Type-of-Service（ToS）フィールドに QoS 保証のための取り扱いポリシを記す．ネットワーク内では ToS フィールドに基づき，パケットの取り扱いを決定する．同様に，Multi-Protocol Label Switcing（MPLS）[2] ネットワークでは，ネットワーク入口でフィルタ等を用いてパケットを分類し，Label Switched Path（LSP）を構築するためのラベルを決定する．

このように，今日のインターネットでは複数のパラメータに基づいた複雑なデータ分類が必要とされている．設計当初から拡張を重ねたインターネットでは，階層化アーキテクチャの範疇に収まらな

くなった．そのため，ファイアウォール，QoS 保証等サービス間のポリシの関係が複雑化し，ネットワーク運用，新たなサービスの導入が難しくなってきている．この問題を解決するために，サービス間のポリシ関係を柔軟に扱えるデータ分類機構が必要となる．

1.2. 本研究の取り組みと位置づけ

社会基盤としてインターネットが広く普及するとともに，インターネットにおけるサービスの多様化・高度化が求められるようになり，ネットワークサービスに対する要求が複雑化してきている．本研究では，複雑化したネットワークサービス内におけるデータ分類に着目し，分類機構の一般化，個々のサービス毎に依存する分類ポリシの明確化を目標とする．

本論文では，はじめにネットワークサービスが必要とする分類機構について論じる．分類機構を明確化により，ネットワークサービスから独立した分類機構の構築が可能となる．次に，この分類機構を一般化することで，個々のサービスに依存しないモデルを構築するとともに，実ネットワークに適用することで一般化したモデルに基づく分類機構の問題点を明らかにする．

1.2.1. モデル構築の取り組み

モデル構築では，サービスに依存しない分類機構を検討し，2 段階選択モデルによるパラメータフィルタモデルを構築した．

従来のネットワークサービスでは，サービス毎に特化したデータ分類機構を使用することが一般的であった．IP 転送では，次ホップルータを決定するために最長一致則に基づく経路表が使用されている．ハードウェア処理が主流である基幹ルータ，ソフトウェア処理が主流である低価格ルータおよび末端ノードのいずれに実装においても，経路表検索は最長一致則という統一された規則に基づいて処理する．ファイアウォールでは，パケットの通過可否を決定するためにフィルタ規則を順に並べたアクセスリストが使用されることが多い．また，このアクセスリストは実装毎に処理が異なる．サービス間の処理順序等の関係は，初めにファイアウォールの処理をした後にパケット転送を行うというように，あらかじめ決められた手順に固定されていた．この場合，ネットワーク上のノードに同一のフィルタ規則を与えても，ネットワーク全体で一貫したサービス提供が困難である．

本研究では，データ分類機構をパラメータフィルタという新しいモデルに転換し，2 段階選択モデルによりパラメータフィルタのサービス依存部分，非依存部分を明確にすることで，サービスから独立したパラメータフィルタの一般化したモデルを提案する．

1.2.2. モデル構築の位置づけ

インターネットのプロトコル・モデルは，当初開放型システム間相互接続（Open Systems Interconnection; OSI）参照モデルを代表例とする階層化アーキテクチャで開発・実装され，運用されてきた．階層化アーキテクチャでは，各層は上位層によるデータ転送要求を必要に応じて下位層を呼び出すことでネットワークを実現する．また，各層の制御は層毎に独立して行うことができた．しかし，ネットワークサービスの高機能化要求により，階層を越えた制御・転送が行われるようになってきた．この結果，制御の対象は複数の階層をまたがり，構造が複雑化した．これにより，制御が複雑化し，機能拡張にも困難が伴うようになった．この問題を解決する概念として，制御と転送の分離モデルが提案されている．このモデルは，ネットワーク上の各ノード毎の転送機構から分離して，ネットワーク内の各ノードを一体とした制御機構を可能とする．

一方，転送機構はサービス毎の制御機構から制御されることになる．入力されたデータは，サービス毎の制御パラメータに従って転送機構で処理される．しかし，今までは一般化されたデータ分類機構のモデルがなかったため，入力データと処理の対応をサービスに依存せずに記述することができなかった．そのため，データ分類機構はサービス毎に存在し，サービスを越えて入力データに対して行う処理の順序・排他関係を制御することが不可能だった．

データ分類機構を一般化したパラメータフィルタのモデル構築は制御・転送からデータ分類機構をパラメータフィルタとして分離することで，この問題を解決する．

1.2.3. フレームワーク構築の取り組み

フレームワーク構築では，2 段階選択モデルによるパラメータフィルタを実装し，実証実験に取り組んだ．これにより明らかになった問題点を解決するために，分類処理の速度性能を向上させる手法を提案する．

構築したフレームワークを使用して，DiffServ, MPLS による QoS 保証ネットワークを構築して実証実験を行い，DiffServ と比較した処理性能測定を行った．その結果，従来の特定サービスに特化したデータ分類機構に比べ，処理速度性能が劣ることが明らかになった．

本研究では，パラメータフィルタのサービス非依存分に多次元空間上での効率的な検索手法として知られる R*-tree [3] を応用することで，フレームワークの汎用性を維持しながら処理速度の大幅な向上を達成した．この結果，分類フィルタ数が多い場合にサービスに特化した実装を上回る性能を示した．

1.2.4. フレームワーク構築の位置づけ

基幹ルータ等のトラフィックが集中し高速処理が必要なネットワーク機器は，ハードウェア処理による転送機構を使用する．低価格ルータおよび末端ノードは，ソフトウェア処理による転送機構を使用する．これらネットワーク機器では異なるデータ分類機構の実装が用いられるが，モジュール間インタフェイス，入出力データの構造が統一されていれば，データ分類機構および関連するモジュールの実装の可用性が高まる．このため，提案モデルに基づいたデータ分類機構を広く用いるために，パラメータフィルタのフレームワークを設計し，API を提供することは重要である．また，ソフトウェア処理を行うネットワーク機器では，実装したフレームワークを移植することで応用可能である．

本研究では，汎用的なパラメータフィルタを実際に実装し，運用することで，提案モデルが実運用にも耐えうることが示せた．条件によってはサービスに特化した実装を上回る性能を示したことは，サービスの従来の実装に組み込むことの実現性を示す．

さらに，提案するパラメータフィルタをネットワークサービスに適用することで，新たなサービスの導入を容易にし，既存のサービスの組み合わせを様々なポリシで制御可能となる．これにより，今後のネットワークサービスの多様化・高度化に大きく寄与すると考える．

1.3. 本論文の構成

本章に続き，第 2 章では，ネットワークサービスにおけるデータ分類について述べ，パラメータフィルタの概念について説明する．第 3 章では，本研究が提案するパラメータフィルタの 2 段階選択モデル，KUMA's Universal Parameter Filter (KUPF) について述べる．第 4 章では，2 段階選択モデルに基づいたパラメータフィルタ，KUPF フレームワークの設計と実装を説明する．

第 5 章では，KUPF フレームワークの実証実験について述べる．この実験では，DiffServ ネットワーク内のルータにおいて，フローフィルタとして KUPF フレームワークを使用することで，KUPF の適用性を示す．さらに，IP 入力フィルタに KUPF フレームワークを適用することで，IP パケット制御機構とパラメータフィルタを分離・独立させられることを示す．また，KUPF フレームワークのパフォーマンスを，IP 入力フィルタとしての処理性能で測定した結果について述べる．第 6 章では，KUPF フレームワークの高速化に関して，多次元空間上の仮想包囲矩形を適用した高速化手法を提案する．また，提案手法の実装と実験による評価について述べる．第 7 章では，KUPF フレームワークの高速化に関して，パラメータフィルタの 2 段階選択の連携について論じる．

最後に第 8 章では，今後の課題を述べ，本研究を総括する．

第2章 パラメータフィルタ

インターネット上のネットワークサービスは，データを入力するとそのデータに対して何らかの処理を実行し，その結果のデータを出力する機構だと考えることができる．本章では，ネットワークサービスにおける入力データの分類について述べ，その問題点を明らかにする．その後，本研究におけるパラメータフィルタのモデルを説明する．

2.1. データ分類機構

従来のインターネットでは，ネットワーク上の各ノードは階層化アーキテクチャに基づいたモジュールを持ち，それらが各層毎に自律分散的に動作する．IP 転送においては，経路制御プロトコルによって生成された経路が経路表に設定され，インターネット層は経路表に基づき転送処理を実行する（図 2.1）．モジュール内では，制御処理と転送処理が密接に連携したモジュールとなっている．そのため，インターネット層における転送処理は，同一階層の経路表検索と切り離して処理できない．

しかし，今日求められる広域での品質保証，トラフィック制御等の高機能サービスを実現するためには，複数の階層，複数のノードが一体となった制御が要求される．くまプロジェクト[1]では，転送機構と制御機構を分離し，ネットワーク上に分散した転送機構を広域に広がる単一の制御機構を導入することで，ネットワークの一貫した制御を目指している（図 2.2）．

一般に，ネットワークサービス機構が入力データを処理する場合，データを処理するためにサービス毎の基準に基づいて入力データを分類する．例えば，ファイアウォールは，入力パケットの通過可否

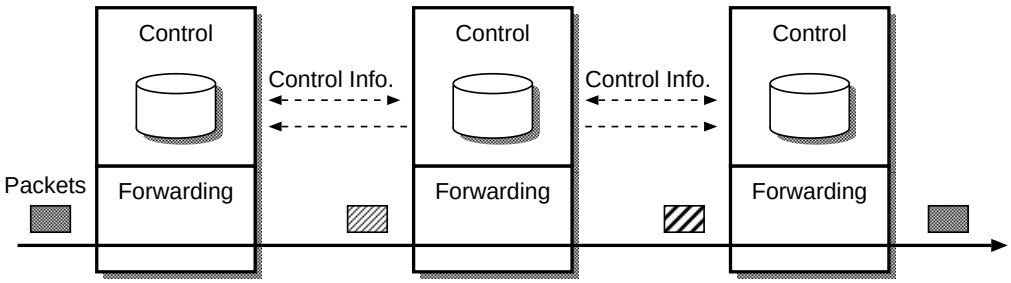

図 2.1: 従来の転送制御

[1] くまプロジェクト – http://www.kuma-project.net/

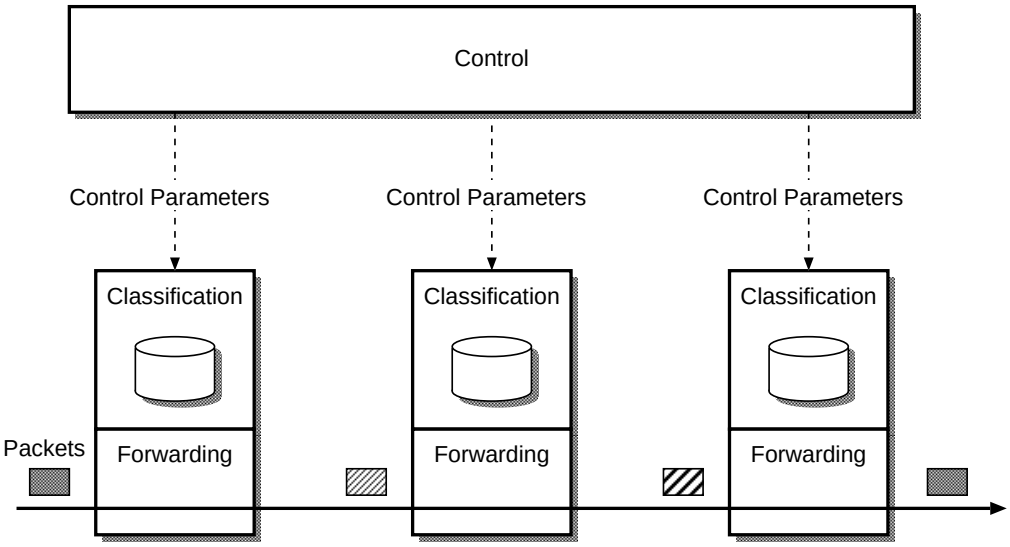

図 2.2: 制御機構と転送機構の分離

を決定するために，フィルタ規則に基づいてパケットを分類する．サービス品質（Quality of Service; QoS）を保証するネットワークでは，各ルータがパケットに適用する制御方法を決定するために，ヘッダの値によってパケットを分類する．このように，様々なネットワークサービスは，処理する入力データを入力データの属性値によって分類し，分類結果に対応する規則を適用する．

以上のことを実現するために，転送機構は処理する入力データを入力データの属性値によって分類するデータ分類機構を使用する．このデータ分類機構の処理を制御するために，制御機構はデータ分類機構に対してサービス要求に基づいた制御パラメータを与える．

2.2.　従来システム

データ分類機構の従来システムとして，経路表検索とアクセスリストについて説明する．

2.2.1.　経 路 表 検 索

インターネット上では，IP パケットは始点ホストから順次ルータで転送されながら終点ホストに到達する．この際，各ルータは入力パケットの IP ヘッダに記された終点 IP アドレスで経路表を検索し，次に転送すべきルータ（次ホップルータ）を決定する．

経路表は，プレフィックス長付き終点アドレスと次ホップルータの IP アドレスの組の一覧である．経路表の例を表 2.1 に示す．終点アドレスの “/” の後ろがプレフィックス長を表し，IP パケットの終点 IP アドレスと，経路表の IP アドレスを比較するビット数を示す．IP アドレスの先頭からプレ

表 2.1: ルーティングテーブルの例

終点アドレス	次ホップルータ
10.0.1.0/24	10.0.0.1
10.0.2.0/24	10.0.0.2
10.0.3.0/24	10.0.0.3
10.0.4.0/24	10.0.0.4
10.0.4.1/32	10.0.0.41
10.0.5.0/24	10.0.0.5

フィックス長で指定されたビット数分が一致すれば，アドレスが適合したと判断する．例えば，終点アドレスが 10.0.2.9 のパケットは，10.0.2.0/24 に適合し，次ホップルータは 10.0.0.2 となる．終点アドレスが 10.0.4.1 のパケットは，10.0.4.0/24 と 10.0.4.1/32 に適合する．この場合，経路表検索ではプレフィックス長の長いアドレスを選択する．これを最長一致則と呼ぶ.

経路表上の経路として，管理者が直接経路を設定する静的経路と自律システムにより自動的に経路を設定する動的経路がある．これらの経路はすべて終点アドレスと次ホップルータの組であり，最長一致則に基づき検索される．経路表検索は実装によらずに動作が同一であり，モジュールの可用性が高い．そのため，互いに独立して経路検索の性能を向上させる実装，新規の経路制御プロトコルの開発が可能である．

4.3 BSD Reno release 以前の BSD UNIX は，経路表にハッシュ関数を用いたハッシュ表が使用されてきた．ハッシュ値は，アドレスをクラスに基づいてネットワーク部とホスト部に分割し，ネットワーク部の値を元に生成した．そのため，クラスにより決まるプレフィックス長より長いプレフィックス長が使われた場合，実際のネットワーク部より短いビット列でハッシュ値を生成する．このためにハッシュ値の等しい経路が増え，経路検索の効率が低下する．クラスに基づかない経路を効率的に検索するため，4.3 BSD Reno release 以降の BSD UNIX は Radix Tree [4] を採用した．Radix Tree は，アドレスをビット列と見なし，各ビットの値により分岐する 2 分木を用いて経路表を検索する．全ビットに対してノードを対応させると木が冗長となるため，分岐のないノードは省略して木を縮退させる．さらに高速な処理が求められる場合，連想メモリ等のハードウェア処理が使用される．

自律システムを動作させるプロトコルとしては，距離ベクトルアルゴリズムによる Routing Information Protocol (RIP)[5]，リンク状態アルゴリズムによる Open Shortest Path First (OSPF)[6] 等がある．これらの経路制御プロトコルは，経路表の実装に関わらず使用可能である．

表 2.2: ファイアウォールポリシの例

規則番号	始点アドレス	終点アドレス	プロトコル	始点ポート	終点ポート	動作
1	*	10.1.0.0/16	TCP	*	80	転送
2	*	10.1.0.0/16	TCP	*	21	転送
3	10.2.0.0/16	10.1.0.0/16	TCP	*	23	転送
4	*	*	*	*	*	破棄

2.2.2. Linux Forwarding Information Base

始点アドレス，IP ヘッダの Type-of-Service（ToS）フィールド等の複数パラメータに基づいた IP 転送を行うために，Linux では終点アドレスによる経路検索の代わりに Forwarding Information Base（FIB）を採用している．Linux FIB においては，制御パラメータである経路情報はハッシュと線形リストの組み合わせにより管理される．FIB は終点アドレスに対し，プレフィックス長別のハッシュ表を構成し，ハッシュ値が同一の経路情報は線形リストに登録される．FIB からの検索は，始点・終点アドレス，入・出力インタフェイス，ToS，スコープの複数パラメータを検索キーとして経路検索が可能である．

Linux FIB は経路検索を終点アドレスをキーとする検索から複数パラメータをキーとする検索に拡張した．しかし，検索キーのパラメータは実装時に固定され，検索ポリシは終点アドレスが最優先に固定されている．そのため，検索パラメータの追加，制御パラメータ間の優先度等のポリシの変更が行えず，多様化するサービスに柔軟に対応できない．

2.2.3. アクセスリスト

ファイアウォールでは，入力パケットを始点・終点 IP アドレス，プロトコル番号，始点・終点ポート番号等により分類し，パケットを転送もしくは破棄する．表 2.2 にファイアウォールポリシの例を示す．‘*’ は値を問わないことを表す．このポリシでは，TCP 通信のいくつかのパケットの転送を許可している以外，基本的にパケットを破棄している．

ファイアウォールの動作を決定する方法として，アクセスリストがよく使われる．アクセスリストは，入力パケットと比較するパラメータ群から構成されるフィルタ規則と動作の組の一覧である．アクセスリストは順序に意味がある．表 2.2 のポリシに対応するアクセスリストの例を表 2.3 に示す．ここでは，フィルタ規則の上から順にパケットと比較し，最初に適合したフィルタ規則を選択する．始点アドレス 10.2.0.1，終点アドレス 10.1.0.1，プロトコル TCP，始点ポート 2000，終点ポート 23 の

表 2.3: アクセスリストの例（1）

規則番号	規則
1	permit tcp any 10.1.0.0 255.255.0.0 eq 80
2	permit tcp any 10.1.0.0 255.255.0.0 eq 21
3	permit tcp 10.2.0.0 255.255.0.0 10.1.0.0 255.255.0.0 eq 23
4	deny ip any any

表 2.4: アクセスリストの例（2）

規則番号	規則
4	block in all
1	pass in from any to 10.1.0.0/16 port = 80
2	pass in from any to 10.1.0.0/16 port = 21
3	pass in from 10.2.0.0/16 to 10.1.0.0/16 port = 23

パケットは，3 番目と 4 番目のフィルタ規則に適合する．3 番目のフィルタ規則が最初に適合するため，このパケットは転送される．始点アドレス 10.3.0.1，終点アドレス 10.1.0.1，プロトコル TCP，始点ポート 2000，終点ポート 25 のパケットは，4 番目のフィルタ規則に適合する．したがって，このパケットは破棄される．もう一つの例として，表 2.4 に示す．ここでは，フィルタ規則の上から順にパケットと比較し，最後に適合したフィルタ規則を選択する．フィルタの解釈方法が異なるため，表 2.3 とは規則の順序が異なる．始点アドレス 10.2.0.1，終点アドレス 10.1.0.1，プロトコル TCP，始点ポート 2000，終点ポート 23 のパケットは，1 番目規則 4 と 4 番目規則 3 のフィルタ規則に適合する．4 番目のフィルタ規則が最後に適合するため，このパケットは転送される．

このように，アクセスリストには実装毎に異なるフィルタ規則の表現方法，フィルタ規則の解釈がある．実装が異なる複数のネットワーク機器でネットワークを構築する場合，ネットワーク管理者は実装毎に個別にフィルタ規則を管理する．複数のサービスを並行して提供する場合，ネットワーク機器間におけるフィルタ規則の整合性の維持は困難を伴う．

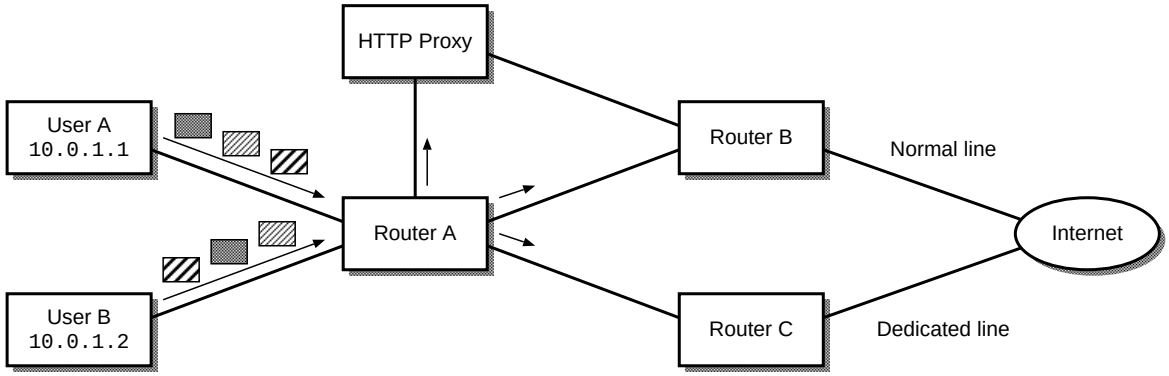

図 2.3: 透過プロキシと専用線サービスがあるネットワーク

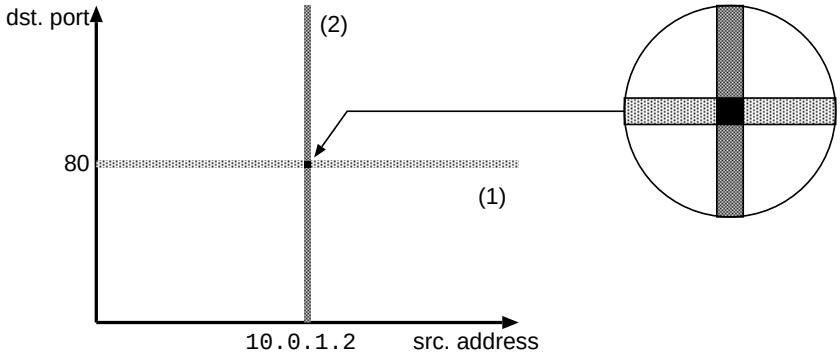

図 2.4: 結果の競合

2.3. ポリシの競合

複数のサービスが並行して提供され，サービス毎のポリシが互いに競合することがある．これらのサービスのフィルタ規則は通常無関係に指定されるため，単一の入力データに対して適合したフィルタ規則が互いに矛盾する場合がある．この場合，これらの規則を同時に適用できないため，いずれのフィルタ規則を適用するべきか決定する必要がある．そのため，データ分類機構には適切な競合解決が要求される．

例として，透過プロキシと特定利用者のための専用回線による優先制御の両方を扱うネットワークを考える（図 2.3）．このネットワークは，以下の規則で制御される．

(1) HTTP リクエストは HTTP プロキシを経由する

(2) 利用者 B は専用線を使用する

この場合，ルータ A のフィルタ規則と処理内容は表 2.5 で表される．表中の “*” は値を問わないこと

表 2.5: 競合するフィルタ規則

規則	始点アドレス	終点ポート	処理内容
(1)	*	= 80	HTTP プロキシに転送
(2)	= 10.0.1.2	*	ルータ C に転送

を示す．図 2.4 は，これらの規則を縦軸に終点ポート，横軸に始点アドレスとする 2 つのパラメータで示す．図中の水平線と垂直線はそれぞれ規則（1），（2）を満足するパケットの集合を表す．

利用者 A からの HTTP リクエストは，始点アドレスが 10.0.1.1，終点ポートが 80 である．このリクエストのパラメータは，図 2.4 の（1）の線上の点で表されるため，規則（1）によりリクエストのパケットは HTTP プロキシに転送される．利用者 B からのメール送信は，始点アドレスが 10.0.1.2，終点ポートが 25 である．このメール送信のパラメータは，図 2.4 の（2）の線上の点で表されるため，規則（2）によりリクエストのパケットはルータ C に転送される．

図 2.4 により，規則（1），（2）を満足するパケットの集合は終点ポート番号が 80，始点アドレスが 10.0.1.2 の点で交差することが示される．このことは，2 集合の共通部分にあるパケットは，規則（1），（2）の両方を満足することを意味する．利用者 B からの HTTP リクエストは，始点アドレスが 10.0.1.2，終点ポートが 80 であるため，規則（1），（2）の両方を満足する．しかし，HTTP プロキシへの転送とルータ C への転送は同時に行えない．このように互いに矛盾するポリシがある場合は，追加ポリシによりどちらか一方を選択する必要がある．

2.4. パラメータフィルタのモデル

インターネット・サービス・プロバイダ（Internet Service Provider; ISP）が，データ転送の優先処理，帯域保証等の高品質サービスを提供する場合を仮定する．この時，サービス品質契約（Service Level Agreement; SLA）を ISP と顧客の間で結ぶ．ISP は SLA に基づき，**フィルタ規則（Filter Rule）**と対応する**アクション（Action）**で構成するデータ処理規則を基幹ルータに登録する．データ処理規則がルータに登録されると，転送データは検査され，必要に応じてアクションが適用される．この際，データ分類機構は，複数のパラメータから構成されるフィルタ規則に基づいて入力データを分類し，ネットワークサービスが入力データに適用する処理を決定する．

図 2.5 にデータ分類機構のモデルを示す．データ分類機構には，データ処理規則が登録されている．データ分類機構にデータが入力されると，順にフィルタ規則と比較される．入力データとフィルタ規則が適合すれば，入力データはそのデータ処理規則に分類される．そして，データ分類機構は分類に

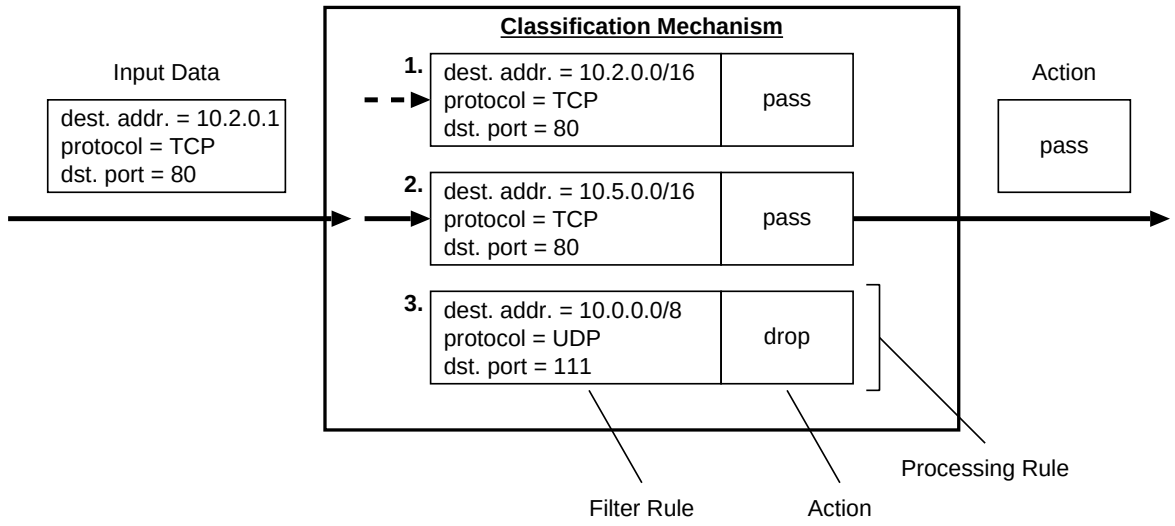

図 2.5: 従来のデータ分類機構

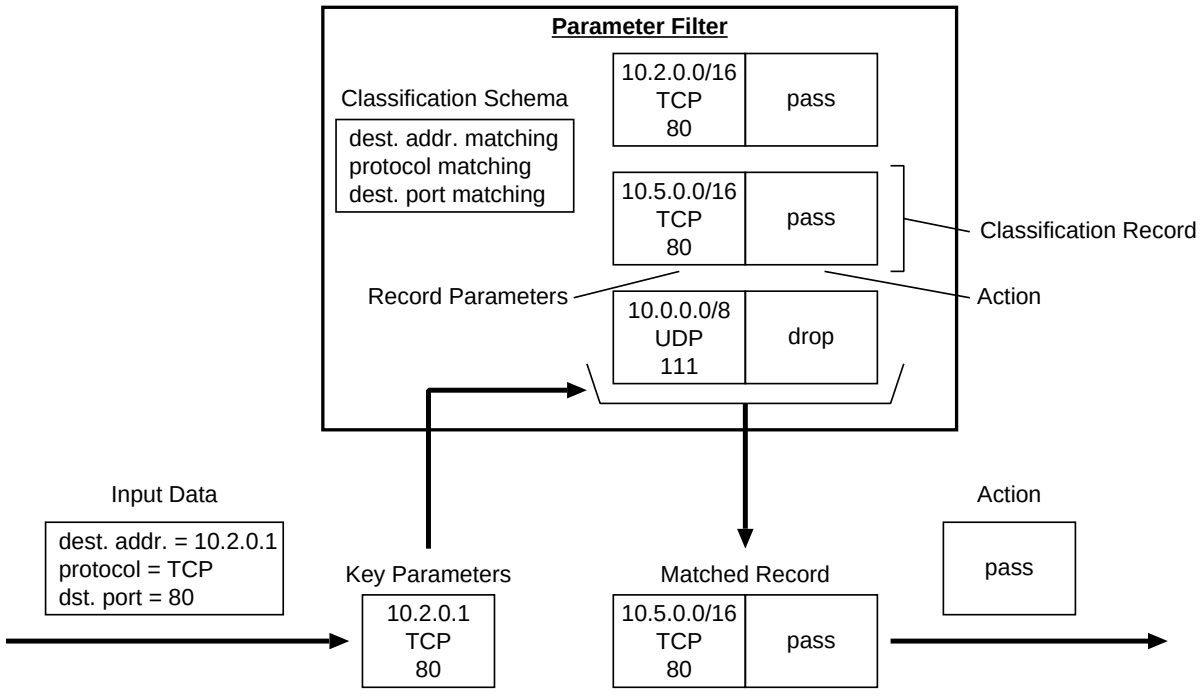

図 2.6: パラメータフィルタモデル

対応するアクションを出力する．

本研究では，図 2.5 の入力データを分類する従来のモデルに対して，入力データのパラメータに基づいて，フィルタ規則とアクションの組を選択するモデルで考える．このモデルをパラメータフィルタと呼ぶ．また，このフィルタ規則とアクションの組を**分類レコード（Classification Record）**と呼ぶ．パラメータフィルタモデルでは，一般に 1 入力データに対して複数の分類レコードが選択される．図 2.6 に本研究におけるパラメータフィルタモデルを示す．パラメータフィルタにはあらかじめ，入力データのパラメータである**キーパラメータ（Key Parameter）**と分類レコードのパラメータである**レコードパラメータ（Record Parameter）**を比較する方法を，**分類スキーマ（Classification Schema）**として定義する．パラメータフィルタには，分類レコードが登録されている．パラメータフィルタは入力データから取り出したキーパラメータが入力されると，分類スキーマに従ってレコードパラメータと比較する．そして，レコードパラメータに適合する分類レコードを出力する．出力した分類レコードからアクションが取り出される．

第3章 パラメータフィルタの2段階選択モデル

従来，インターネット上の各サービスのデータ分類機構はサービス毎に個別に実装されてきた．これは，データ分類機構の一般化されたモデルが今まで存在せず，データ分類機構の実装が個々のサービスと非常に密接に結びついているためである．本章では，第 2 章で説明したパラメータフィルタの概念に基づき，サービスに依存しない 2 段階選択モデルによるパラメータフィルタ，KUMA's Universal Parameter Filter (KUPF) を提案する．

3.1. 2段階選択モデル

複雑化するネットワークにおいて，高度化・多様化するサービスを実現するためには，複数のパラメータによって分類レコードのフィルタ規則を構成しなければならない．また，複数のサービスが並行して運用されるため，複数の分類レコードのフィルタが同時に 1 つの転送データに適合する．サービス毎に適用するポリシが異なるため，複数の分類レコードが競合する場合，パラメータフィルタは競合を適切に解決し，転送データに適用するアクションを決定する必要がある．しかし，ポリシ毎に競合関係は異なるため，フィルタ規則のみにより一意に分類レコードを選択できない．

適用されるアクションを含む分類レコード R_{result} は，ある手続き f を使って以下のように導くことができる[1]．

$$R_{result} = f(\mathbf{R_{all}}, d) \tag{3.1}$$

ここで，$\mathbf{R_{all}}$ はルータに登録された分類レコード群，d は転送データとする．次に，分類レコード群 $\mathbf{R_{all}}$ の内，転送データ d を満足するフィルタ規則から構成されるすべての分類レコード群 $\mathbf{R_{mid}}$ を考える．R_{result} は d を満足するフィルタ規則のうちのいずれかであるため，以下が成立する．

$$R_{result} \in \mathbf{R_{mid}} \subseteq \mathbf{R_{all}} \tag{3.2}$$

したがって，手続き (3.1) は以下の 2 段階に分割可能である．

$$\mathbf{R_{mid}} = f_1(\mathbf{R_{all}}, d) \tag{3.3}$$

$$R_{result} = f_2(\mathbf{R_{mid}}, d) \tag{3.4}$$

[1] 一般的には R_{result} は分類レコード群となるが，ここでは説明を簡単にするために R_{result} を単一の分類レコードとする．

表 3.1: 分類レコードの例

分類レコード	ビット/マスク	範囲	アクション
R_1	0000/0000	0–100	A_1
R_2	0000/1001	4–5	A_2
R_3	0100/0110	4–6	A_3
R_4	0100/0110	94–95	A_4

ここで，f_1 はレコード群全体からフィルタ規則が d を満足するすべてのレコード群 $\mathbf{R_{mid}}$ を抽出する．f_2 は何らかの規則・ポリシ等に従って $\mathbf{R_{mid}}$ から R_{result} を選択する．本研究では，手続き (3.3), (3.4) をそれぞれ第 1 ステージ，第 2 ステージと呼ぶ．

第 1 ステージは，転送データが分類レコードのフィルタ規則を満足するか，全分類レコードを検査し，満足したレコードを $\mathbf{R_{mid}}$ に抽出する．この検査の実行はフィルタ規則の分類スキーマにのみ依存する．したがって，分類スキーマが定義されると，第 1 ステージは分類レコードが使用されるサービスに依存せずに自動的に実行可能である．

一方，第 2 ステージは，フィルタ規則の分類スキーマに加え，サービスに強く依存する．例えば，IPv4/IPv6 の経路表検索機構は，入力 IP パケットの終点アドレスにプレフィックスが最長一致する経路を選択する．ファイアウォール，QoS 対応ルータは一般的に最初，もしくは最後に適合した分類レコードを選択する．

この分類レコードの 2 段階選択モデルは，サービスもしくはサービス実装の依存・非依存部分を分離する．したがって，このモデルの適用により，次に挙げる場合においても第 2 ステージのみの検討で対応可能である．

- レコード選択機構における競合，もしくは複数ルータにより機能する機構における不整合の検査・解決
- 新規サービスのための分類スキーマ・ポリシの実装

データが 4 ビット長ビット列と整数値の属性値を持ち，それぞれビットマスク一致と範囲一致のフィルタ規則で比較するパラメータフィルタを考える．パラメータフィルタは，表 3.1 に示す 4 分類レコードを保持するものとする．この時，

$$\mathbf{R_{all}} = \{R_1, R_2, R_3, R_4\}$$

となり，f_1 はビットマスク一致と範囲一致による入力データとフィルタ規則の比較となる．ビット

列 0100，整数値 5 の属性値を持つデータ $d = (0100, 5)$ が与えられたとき，ビットマスク一致は R_1, R_2, R_3, R_4 すべてが一致し，範囲一致は R_1, R_2, R_3 が一致する．その結果，

$$\mathbf{R_{mid}} = f_1(\mathbf{R_{all}}, d) = \{R_1, R_2, R_3\}$$

となる．R_1, R_2, R_3 のアクションが競合する場合，競合は f_2 が解決する．ポリシが範囲一致条件の範囲の大きさが小さいレコードを優先する場合，範囲が 0–100 の R_1，4–6 の R_3 より 4–5 の R_2 が優先されるので，

$$R_{result} = f_2(\mathbf{R_{mid}}, d) = R_2$$

となり，データ d に適用するアクションは A_2 に決定される．

第4章 KUPFフレームワーク

本章では，KUPF における規則表現のデータ構造，規則の管理構造，データを分類するための KUPF フレームワーク，およびその API についての概略を示し，その設計・実装をまとめる．

4.1. KUPFの構成

一般的に，入力データを処理する場合，入力データに対応する処理を選択するための規則およびその処理自体の記述が必要となる．KUPF では，前者の規則をフィルタ規則，後者の記述をアクション，さらにその対を分類レコードと呼ぶ．KUPF モデルは，分類レコードを選択する機構であるパラメータフィルタを，サービス非依存処理とサービス依存処理の 2 段階に分割する．パラメータフィルタを 2 段階に分割したことで，以下のことが実現できる．第 1 ステージは，個々の分類レコードに対してフィルタ規則の条件が入力データを満たすか否かの判定をするのみで，サービスに依存せずに処理を行うため，汎用的に設計・実装できる．第 2 ステージは，第 1 ステージの結果からサービス毎のポリシに従って，入力データに適用するアクションを持つ分類レコードを選択する．サービス依存処理は第 2 ステージで完結しているため，第 2 ステージ置き換えのみによりパラメータフィルタを異なるサービスに応用できる．

図 4.1 に KUPF の概要を示す．分類レコードは，分類表（Classification Table）で管理される．第 1 ステージは，入力データのパラメータを入力すると，パラメータに一致するフィルタを持つ分類レコードを出力する．第 2 ステージは，入力データのパラメータと分類レコードを入力すると，サービス毎のポリシに従って分類レコードを選択し，出力する．

4.1.1. パラメータと比較

ポート番号，IP アドレス等の各値をパラメータ（Parameter）と呼ぶ．KUPF では，これらの値を符号なし整数，固定長ビット列等の基本的なパラメータの型，パラメータ型（Parameter Type）で表現し，ポート番号，IP アドレス等の特定の意味を持つ型を用いない．

図 4.2 にパラメータとパラメータ型のデータ構造を示す．パラメータは，パラメータの値を表現するデータで構成する．パラメータは内部にパラメータ型に関する情報を保持しないため，パラメータだけでは特定の値を指し示さない．図中には示さないが，パラメータ型によって異なるバイト長のデー

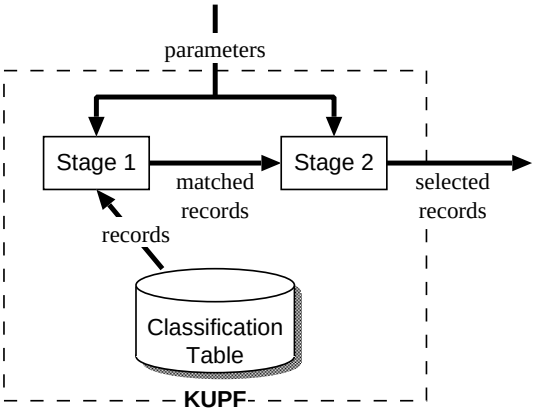

図 4.1: KUPF 概要

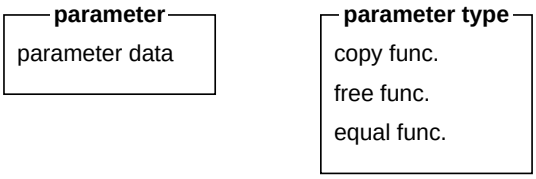

図 4.2: パラメータとパラメータ型

タの取り扱いを容易にするため，パラメータ内にパラメータのバイト長を保持する．パラメータ型は，パラメータを複製・解放する関数，2 つのパラメータの同一性を判定する関数で構成する．パラメータはパラメータ型と関連付けることで，複製・解放・同一性判定を行える．

あるパラメータとパラメータ群との関係を比較演算として定義した物を比較型（Comparison Type）と呼ぶ．比較型はパラメータの型，パラメータ群の型，比較手続きによって構成される．例えば，整数が範囲内に含まれていることを表す比較型は，

- 整数のパラメータ型（符号なし整数）
- 範囲の上限と下限のパラメータ型（それぞれ符号なし整数）

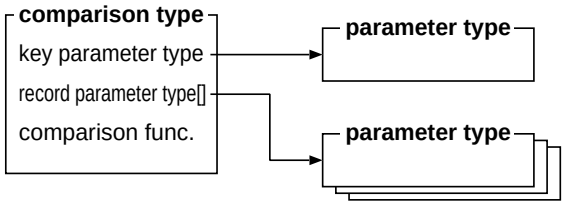

図 4.3: 比較型

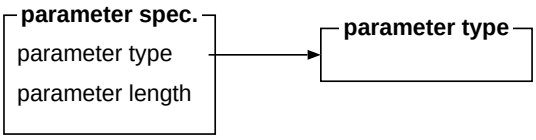

図 4.4: パラメータ仕様

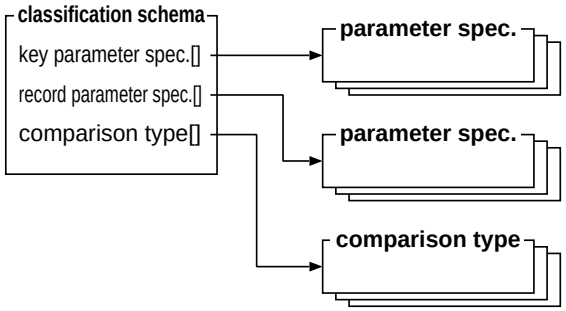

図 4.5: 分類スキーマ

- 比較手続き（下限 $\leq$ 整数 $\leq$ 上限）

と記述できる．

図 4.3 に比較型のデータ構造を示す．比較型は，キーパラメータ（前記例の整数）のパラメータ型，レコードパラメータ群（前記例の上限・下限）のパラメータ型配列，比較手続きの比較関数で構成する．

4.1.2. パラメータ仕様と分類スキーマ

ビット列が IPv4 アドレスなどの意味を持つには，32 ビットという長さの属性が必要である．このパラメータ型に長さを加えた物をパラメータ仕様（Parameter Specification）と呼ぶ．図 4.4 にパラメータ仕様のデータ構造を示す．

具体的なデータがある条件を満たしているかを検証するためには，入力データのパラメータ仕様，条件となるパラメータ群の仕様，およびそれらを比較するための比較型が必要となる．複数の入力デー

表 4.1: 分類スキーマの例

	終点アドレス	プロトコル番号	終点ポート番号の範囲
キーパラメータ仕様群	32bit 長ビット列	符号なし整数	符号なし整数
レコードメータ仕様群	32bit 長ビット列	符号なし整数	符号なし整数 符号なし整数
比較型	ビット列比較	符号なし整数比較	符号なし整数範囲比較

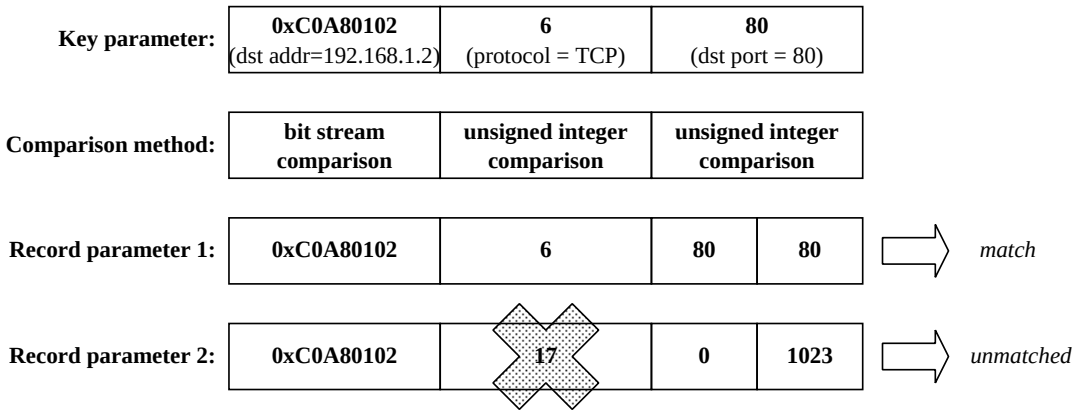

図 4.6: パラメータ比較

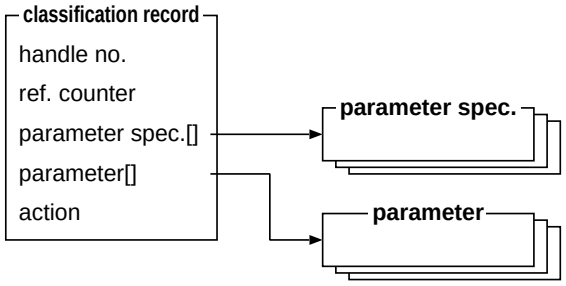

図 4.7: 分類レコード

タに対する条件の集合，すなわち入力データ毎の上記 3 要素の集合を分類スキーマ（Classification Schema）と呼ぶ．図 4.5 に分類スキーマのデータ構造を示す．

例えば，IPv4 パケットを終点アドレス，プロトコル番号，終点ポート番号の範囲の 3 条件で分類する場合，分類スキーマは表 4.1 で表現できる．分類スキーマに示されたすべての条件について，入力データのパラメータであるキーパラメータ群と分類レコードのパラメータであるレコードパラメータ群が条件を満たすとき，キーパラメータ群とレコードパラメータ群は条件を満たすと判定する（図 4.6）．

分類スキーマを構成するパラメータ仕様と比較型は KUPF フレームワーク利用者が新たに定義することで拡張可能である．そのため，KUPF は任意の分類スキーマに柔軟に対応可能である．

4.1.3. 分類レコード

分類レコードの構造を図 4.7 に示す．条件となるパラメータ群とアクション以外に，ハンドル番号，参照カウンタを保持する．ハンドル番号は分類表内で分類レコードを識別する．参照カウンタは，分

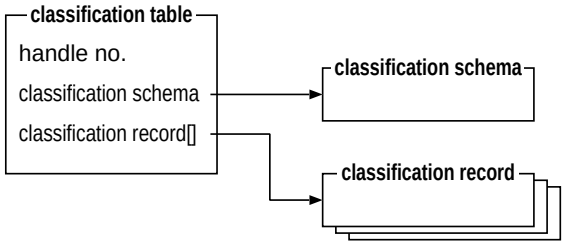

図 4.8: 分類表

類レコードが他のデータ構造から参照されるとき 1 増やし，参照がなくなるとき 1 減らす．たとえば，分類表に追加されるとき参照カウンタは 1 増やされ，分類表から削除されるとき 1 減らされる．参照カウンタが 0 になったとき，フレームワークはパラメータ仕様群から得られるパラメータ解放関数を使ってパラメータを解放し，分類レコードを解放する．この参照カウンタの仕組みにより，分類レコードはデータをコピーせずに複数のデータ構造から参照可能となる．図に示したものの他に，統計情報記録のために分類レコードの使用回数を保持するカウンタを保持する．

4.1.4. 分　類　表

分類表の構造を図 4.8 に示す．分類スキーマ，分類レコード群以外に分類表を識別するハンドル番号を保持する．図には示したものの他に，フレームワーク利用者が名付けた名前で分類表を識別できる分類表名・ユニット番号を保持する．また，統計情報記録のために分類回数，レコード追加・削除回数等のカウンタを保持する．

4.2. プログラムインタフェイス

KUPF プログラムインタフェイスは，主に表 4.2 のパラメータ操作，分類表操作，分類レコード操作，分類操作の 4 種類で構成される．以下にこれらのインタフェイスの概略を説明する．

4.2.1. パラメータ操作

パラメータ操作は，パラメータ作成・解放を行う．パラメータ作成は，パラメータのメモリを割当て，値を設定する．パラメータ作成操作は，パラメータ毎に別の関数を呼び出す．`kupf_`*xxx*`_alloc` の *xxx* はパラメータ型となる．例として，符号なし整数のパラメータ作成は，パラメータ仕様と整数値を引数として `kupf_uint_alloc` を呼び出す．可変長ビット列のパラメータ作成は，パラメータ仕様とビット長，ビット値を引数として `kupf_vlbits_alloc` を呼び出す．パラメータ解放は，パラメー

表 4.2: プログラムインタフェイス

パラメータ操作	
パラメータ作成	kupf_*xxx*_alloc(パラメータ仕様，値，...)
パラメータ解放	kupf_paramdata_free(パラメータ仕様，パラメータ)
分類表操作	
分類表作成	kupf_comp_table_alloc(名前，ユニット番号， キーパラメータ仕様群，レコードパラメータ仕様群，比較型群)
分類表削除	kupf_comp_table_free(分類表)
分類レコード操作	
分類レコード作成	kupf_entry_alloc(レコードパラメータ仕様群，パラメータ群， アクション)
分類レコード保持	kupf_entry_retain(分類表)
分類レコード解放	kupf_entry_free(分類表)
分類レコード追加	kupf_comp_table_entry_add(分類表，分類レコード)
分類レコード削除	kupf_comp_table_entry_remove(分類表，分類レコード)
分類レコード削除	kupf_comp_table_entry_remove_handle(分類表， 分類レコードハンドル番号)
分類レコード全削除	kupf_comp_table_entry_flush(分類表)
分類操作	
第 1 ステージ分類操作	kupf_comp_table_collect(分類表，キーパラメータ群)
第 2 ステージ分類操作	kupf_select_*xxx*(分類表，分類レコード群，...)

表 4.3: パラメータ型

型名	型識別名	作成関数
符号なし整数	KUPF_PT_UINT	kupf_uint_alloc(パラメータ仕様, 整数値)
符号付き整数	KUPF_PT_SINT	kupf_sint_alloc(パラメータ仕様, 整数値)
固定長ビット列	KUPF_PT_FLBITS	kupf_flbits_alloc(パラメータ仕様, ビット値)
可変長ビット列	KUPF_PT_VLBITS	kupf_vlbits_alloc(パラメータ仕様, ビット長, ビット値)

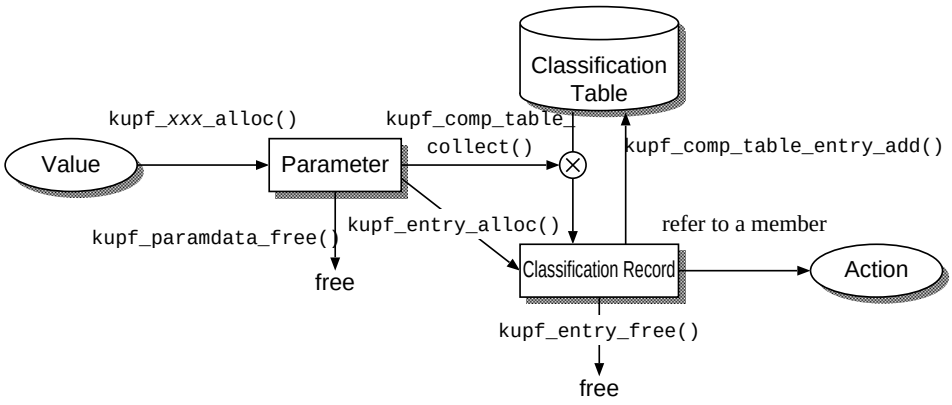

図 4.9: パラメータの変遷

表 4.4: 比較型

比較型名	型識別名	用途
符号なし整数比較	KUPF_CT_UINT_MATCH	プロトコル番号比較
符号付き整数比較	KUPF_CT_SINT_MATCH	
固定長ビット列比較	KUPF_CT_FLBITS_MATCH	
固定長ビット列先頭比較	KUPF_CT_FLBITS_PREFIX_MATCH	IP アドレスプレフィックス比較
固定長ビット列マスク付き比較	KUPF_CT_FLBITS_MASK_MATCH	IP ToS フィールドマスク付き比較
可変長ビット列比較	KUPF_CT_VLBITS_MATCH	

タに割当てられたメモリを解放する．

表 4.3 に KUPF フレームワークが用意するパラメータ型を示す．フレームワーク利用者はパラメータ作成関数とパラメータ型を作成することで，新たなパラメータ型をこの表のパラメータ型と同様にフレームワークで利用可能である．図 4.9 はフレームワークにおけるパラメータの変遷を示している．パラメータ作成関数以外は，パラメータ型に依存しないインタフェイスであるため，新たなパラメータ型を定義してもパラメータ作成後のインタフェイスを変更する必要がない．

4.2.2. 分類表操作

分類表操作は，分類表作成・削除を行う．分類表作成は，分類表のメモリを割当て，分類スキーマ，その他の情報を設定する．分類表作成時にキーパラメータ仕様群，レコードパラメータ仕様群，比較型群を用いて分類スキーマを設定するため，他の呼び出しではこれらの情報を与える必要がない．分類表削除は，分類表が持つすべてのメモリ割当てを解放する．

表 4.4 に KUPF フレームワークが用意する比較型を示す．各比較型は，キーパラメータが定められた手続きでレコードパラメータ群の条件を満たすか否か判定する．符号なし整数比較，符号付き整数比較，固定長ビット列比較，可変長ビット列比較は，それぞれの型のキーパラメータとレコードパラメータを比較し，値の一致を判定する．固定長ビット列先頭比較は，固定長ビット列のキーパラメータと可変長ビット列のレコードパラメータを比較する．キーパラメータの先頭ビット値がレコードパラメータのビット値と一致すれば，一致と判定する．固定長ビット列マスク付き比較は，固定長ビット列のキーパラメータと 2 つの固定長ビット列のレコードパラメータを比較する．レコードパラメータの 1 つをマスクとして，マスクビット値が 1 のビット桁についてキーパラメータのビット値ともう一方のレコードパラメータビット値を比較する．マスクビット値が 1 のビット桁すべてのビット値が一致すれば，一致と判定する．フレームワーク利用者は比較型を作成することで，新たな比較型をこの表の比較型と同様にフレームワークで利用可能である．

4.2.3. 分類レコード操作

分類レコード操作は，分類レコード作成・保持・解放，分類表への分類レコードの追加，分類表からの分類レコード削除を行う．

分類レコード作成は，分類レコードのメモリを割当て，レコードパラメータ仕様群，パラメータ群，アクションを設定する．分類表保持・分類表解放は，それぞれ分類レコードの参照カウンタを 1 増加・減少させる．減少させた結果，参照カウンタが 0 になれば，分類レコードのメモリ割当てを解放する．分類表等のモジュールは，分類レコードを参照する間分類レコードの参照カウンタを 1 増やす．参照するモジュールが存在しない分類レコードは参照カウンタが 0 となるため，メモリ割当てが解放される．この参照カウンタの仕組みにより，分類レコードを複製せずに複数のモジュールが同一の分類レコードを参照できる．

分類レコード追加は，分類表に分類レコードを追加し，分類表内で重複しない番号をハンドル番号として分類レコードに割り当てる．返値として割り当てたハンドル番号を返す．分類レコード削除・分類レコード全削除は，分類レコードから分類レコードを削除する．パラメータの種類は分類レコード内に隠蔽されるため，分類スキーマの種類にかかわらず分類レコード操作のインタフェイスは同一である．

4.2.4. 分 類 操 作

分類操作は，第 1 ステージ分類操作と第 2 ステージ分類操作に分かれる．第 1 ステージ分類操作は，分類表とキーパラメータ群を与えると，返値として，分類表から，キーパラメータ群を条件として満たすのレコードパラメータ群を持つ全分類レコードが返る．比較型は分類表が保持するため，分類ス

表 4.5: 優先度の例

分類レコード	分類レコード優先度	比較優先度 1	比較優先度 2
R_1	3	3	1
R_2	2	1	2
R_3	1	3	2

キーマの種類にかかわらず分類操作のインタフェイスは同一である．

第 2 ステージ分類操作は，サービス毎に依存するため個別実装が必要である．KUPF は，サービスに依存しない実装として最善一致 `kupf_select_best` を提供する．返値として，最善一致の分類レコードを返す．最善一致は，IP 経路表検索の最長一致則など一致の度合いの強弱があるパラメータ比較において，一致度が最大のパラメータを持つ分類レコードを選択する．アクセスリストにおける最初もしくは最後に適合したフィルタの選択は，パラメータ比較に依存しない分類レコード固有の優先度とみなすことができる．以上から，分類コードが最善一致のために保持する優先度は

比較優先度

パラメータ比較毎の一致度

分類レコード優先度

分類レコード固有の優先度

である．一般にパラメータは複数あるため，参照する優先度により最善の結果が異なる場合がある．この問題を解決するために，優先順位を導入する．優先順位は，最善一致を判定する優先度の参照順序を指定する．

表 4.5 の優先度の場合を考える．分類レコード優先度は，降順で並んでいる．この場合，優先順位の先頭が分類レコード優先度のとき，最善一致は最初に一致した分類レコードを選択することと同一である．優先順位が （比較優先度 1，比較優先度 2）の場合，初めに比較優先度 1 が最大の R_1, R_3 が選択され，次に比較優先度 2 が大きい R_3 が選択される．優先順位が （比較優先度 2，分類レコード優先度）の場合，はじめに比較優先度 2 が最大の R_2, R_3 が選択され，次に分類レコード優先度が大きい R_2 が選択される．このように，優先順位に従って最善一致の結果は大きく異なる．

4.3. 実　　装

2 段階選択モデルに基づいたパラメータフィルタ，KUPF フレームワークを NetBSD 1.5.2 および 1.6 上で C 言語を用いて実装した．KUPF フレームワークはユーザ空間プログラムで動作するライブラリ実装と，カーネル空間で動作するカーネルモジュール実装の両方を実装した．両実装はメモリ管理の一部を除き，同一のプログラムインタフェイスを備える．そのため，ユーザ空間プログラムとカーネル空間プログラムの間のコード可用性，カーネル空間実装前のユーザ空間におけるプロトタイプ実装ができる柔軟性などを KUPF 応用プログラムにもたらす．

4.3.1. 第 1 ステージ

データ分類機構の一例として，経路検索がある．4.3 BSD Reno release 以降の BSD UNIX では，Classless Inter-Domain Routing (CIDR) [7], Variable Length Subnet Mask (VLSM) [8] が必要とする最長一致検索を効率よく検索するために，Radix Tree [4] を採用している．Radix Tree は，マスク付きアドレスの経路表から経路を効率良く検索できる．しかし，Radix Tree の特性上，検索空間は 1 次元であり，利用は単一アドレス空間上の検索に限定される．パラメータフィルタは一般に複数パラメータで構成される多次元のフィルタ規則を持つため，Radix Tree のような 1 次元空間を対象とした従来の経路検索手法は適用できない．また，分類レコードはワイルドカードパラメータを含む複数のパラメータ群で構成されるため，単純な木構造を使用した検索はできない．これらの理由により，KUPF は分類レコードをリスト構造で管理し，分類レコード検索時には逐次検索を行う．

連続する入力データは，同一の属性を持つ傾向がある．この傾向を利用した高速化を図るため，第 1 ステージにキャッシュ機構を導入した．キャッシュ容量は 1 とし，最後の 1 回の第 1 ステージの結果の分類レコード群を，検索キーとなるキーパラメータ群とともにキャッシュに保存する．

4.3.2. 第 2 ステージ

本フレームワークで提供する第 2 ステージ分類操作，最善一致は，優先順位に従って分類レコードの優先度を比較し，優先度が最大の分類レコードを返す．

パラメータ毎の比較優先度を算出するために，図 4.3 の比較型は図で示した以外にレコードパラメータから比較優先度を算出する比較優先度関数を保持する．表 4.6 に表 4.4 の各比較型の比較優先度を示す．符号なし整数比較，符号付き整数比較，固定長ビット列比較，可変長ビット列比較は完全一致を判定する比較である．優先度の優劣はないため，これらの比較関数の比較優先度関数は常に 1 を返す．固定長ビット列先頭比較はレコードパラメータのビット長が長いほど多くのビットをキーパラメータと比較するため，比較優先度関数はレコードパラメータのビット長を返す．固定長ビット列マスク付

表 4.6: 比較優先度関数

比較型名	比較優先度
符号なし整数比較	1
符号付き整数比較	1
固定長ビット列比較	1
固定長ビット列先頭比較	レコードパラメータのビット長
固定長ビット列マスク付き比較	マスクビット値が 1 の数
可変長ビット列比較	1

き比較はマスクビット値が 1 の桁をキーパラメータと比較するため，比較優先度関数はマスクビット値が 1 の数の合計を返す．

比較優先度はキーパラメータに依存せず，レコードパラメータのみで決定する．分類操作の処理を軽減するため，比較優先度は分類表に分類レコードを追加するときに算出し，分類レコード内に保持する．最善一致の分類操作では，分類レコード内に保持している比較優先度の値を参照して分類レコードを選択する．

第5章 KUPFフレームワーク適用実験

本章では，KUPF フレームワークの適用実験について述べる．はじめに，DiffServ ネットワーク内のルータにおいて，フローフィルタとして KUPF フレームワークを使用することで，KUPF の適用性を示す．さらに，IP 入力フィルタに KUPF フレームワークを適用することで，IP パケット制御機構とパラメータフィルタを分離・独立させられることを示す．また，KUPF フレームワークのパフォーマンスを，IP 入力フィルタとしての処理性能で測定した結果について述べる．

5.1. パケットフィルタへの適用

KUPF の適用性を示すために実験を行った．DiffServ[1] の BSD UNIX 上の実装である Alternate Queueing (ALTQ)[9] のフローフィルタとして，KUPF を使用した．

5.1.1. サンプル実装

実験システムは，操作機構とパラメータフィルタから構成される．操作機構は，ALTQ のキューイングとシェーピングの機構で，入力データを制御し，出力する．パラメータフィルタは入力パケットのフローを識別し，入力パケットに適用するアクションを決定する．パラメータフィルタとして，KUPF と ALTQ 組み込みのフィルタを，対比させながら使用した．

ALTQ のフィルタは IP パケットフィルタに特化し，ハッシュ表により高速化を図っている．しかし，フィルタ規則は複数のパラメータを持つため，その一部のパラメータに対してのみハッシュは有効になる．IPv4 の場合はプレフィックス長が 32 の終点アドレス，IPv6 の場合はフローラベルに対してのみハッシュが機能する．

IPv4 パケットと IPv6 パケットのフローを識別するための分類スキーマを，表 5.1，5.2 に示す．

ALTQ は，各入力パケットに単一のアクションを適用する．本実験では，第 2 ステージとして最善一致を用いた．ALTQ との互換性のため，分類レコードの優先度は ALTQ フィルタのルール番号を使用し，最善一致の参照パラメータは分類レコードの優先度とした．

図 5.1 に，KUPF をパラメータフィルタとして使用した ALTQ の概要を示す．このシステムは，IP, KUPF, ALTQ のカーネル空間モジュールと `altqd` のユーザ空間プログラムから構成される．`altqd` は，ALTQ デバイスインタフェイスを呼び出し，フローフィルタと DiffServ 制御の情報をカーネル内

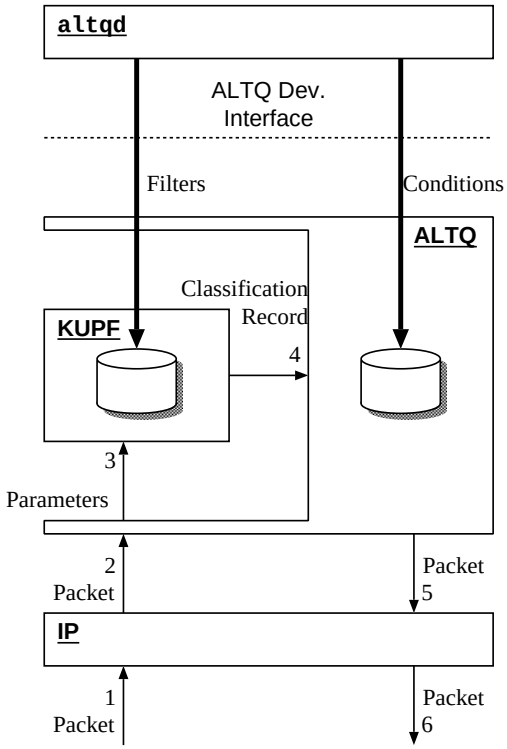

図 5.1: KUPF をパラメータフィルタとして使用した ALTQ

表 5.1: ALTQ IPv4 分類スキーマ

属性	キーパラメータ仕様	レコードパラメータ仕様	比較型
プロトコル	符号なし整数	符号なし整数	符号なし整数比較
ToS	8 ビット長ビット列	8 ビット長ビット列 8 ビット長ビット列	ビット列マスク付き比較
終点アドレス	32 ビット長ビット列	可変長ビット列	ビット列プレフィックス比較
始点アドレス	32 ビット長ビット列	可変長ビット列	ビット列プレフィックス比較
終点ポート	符号なし整数	符号なし整数	符号なし整数比較
始点ポート	符号なし整数	符号なし整数	符号なし整数比較
IPsec 情報	符号なし整数	符号なし整数	符号なし整数比較

表 5.2: ALTQ IPv6 分類スキーマ

属性	キーパラメータ仕様	レコードパラメータ仕様	比較型
プロトコル	符号なし整数	符号なし整数	符号なし整数比較
トラフィッククラス	8 ビット長ビット列	8 ビット長ビット列 8 ビット長ビット列	ビット列マスク付き比較
フローラベル	符号なし整数	符号なし整数	符号なし整数比較
終点アドレス	128 ビット長ビット列	可変長ビット列	ビット列プレフィックス比較
始点アドレス	128 ビット長ビット列	可変長ビット列	ビット列プレフィックス比較
終点ポート	符号なし整数	符号なし整数	符号なし整数比較
始点ポート	符号なし整数	符号なし整数	符号なし整数比較
IPsec 情報	符号なし整数	符号なし整数	符号なし整数比較

の KUPF および ALTQ に登録する．KUPF の操作はすべて ALTQ 経由で行い，ALTQ 利用者からは KUPF を意識させない互換性を維持した．入力パケットの処理の流れは以下の通りである．

1. IP にパケットが入力される
2. IP は入力パケットを ALTQ に送る
3. ALTQ は入力パケットからパラメータ群を取り出し，KUPF に送る
4. KUPF はパラメータ群に適合する分類レコードを ALTQ に返す
5. ALTQ は分類レコードから取り出したアクションに対応する制御を入力パケットに適用し，IP に返す
6. IP は ALTQ で処理された入力パケットを出力する

5.1.2. 適　用　性

2002 年 3 月に 4 日間にわたり開催された WIDE プロジェクト[1]研究会において，臨時ネットワークを構築し，実証実験を行った．図 5.2 に実験ネットワークの概略を示す．このネットワークを用いて，273 人の研究会参加者に IPv4/IPv6 のインターネット接続を提供した．研究会会場からの対外線は，低遅延 1.5 Mbps の ATM 回線と，高遅延上り 0.5 Mbps，下り 1.5 Mbps の衛星回線を用いた．ネッ

[1] http://www.wide.ad.jp/

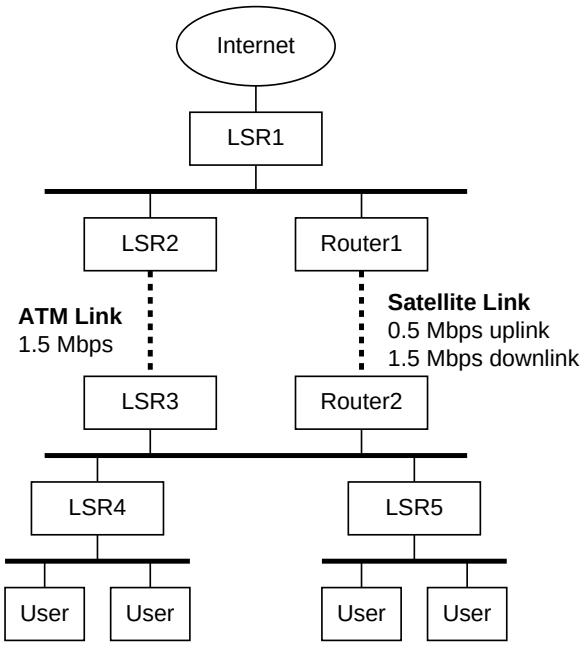

図 5.2: 2002 年 3 月 WIDE 研究会ネットワーク概要

トワーク利用者に，対外線の ATM 回線と衛星回線それぞれについて利用帯域が予約可能なサービスを提供した．

ルータの転送層として，AYAME Multi-Protocol Label Switcing（MPLS）[10][2]を使用した．制御層として，KUMA[11] を使用した．利用者収容ルータからインターネットとの接続点までのルータは，衛星回線の両端を除きすべて AYAME Label Switched Router (LSR) で構成し，MPLS ドメインを構成した．MPLS ドメイン内における通信品質（帯域）保証のために，MPLS/DiffServ の技術を用いた．実験ネットワークでは，帯域におけるボトルネックは対外線のみであった．このため，LSR エッジルータにおいてフロー毎のポリシングとマーキングを，対外線収容ルータにおいてマークに基づいた優先処理を行った．KUPF は，LSR エッジルータにおいて，制御パラメータにもとづいてフローを識別し，MPLS のラベルを決定するために使用した．予約なしの通信は遅延の大きい衛星回線を経由する．利用者が WWW インターフェースにより回線を予約すると，予約した利用者の通信のみ遅延の小さい ATM 回線を利用できる．

KUPF はこのサービスに適用でき，4 日間にわたり利用者に安定したネットワーク環境を提供した．表 5.3 に研究会 3 日目における KUPF 第 1 ステージのキャッシュヒット率を示す．LSR5, LSR6 はユーザセグメント収容 LSR エッジルータである．この結果から，第 1 ステージのキャッシュヒット率が 3 割から 4 割程度であることが分かった．したがって，キャッシュ機構のキャッシュ容量が 1 と少

[2]あやめプロジェクト – http://www.ayame.org/

表 5.3: キャッシュヒット率

属性	要求回数	ヒット数	ヒット率
LSR5	4,774,389	1,772,166	37.1%
LSR6	3,151,356	1,273,165	35.3%

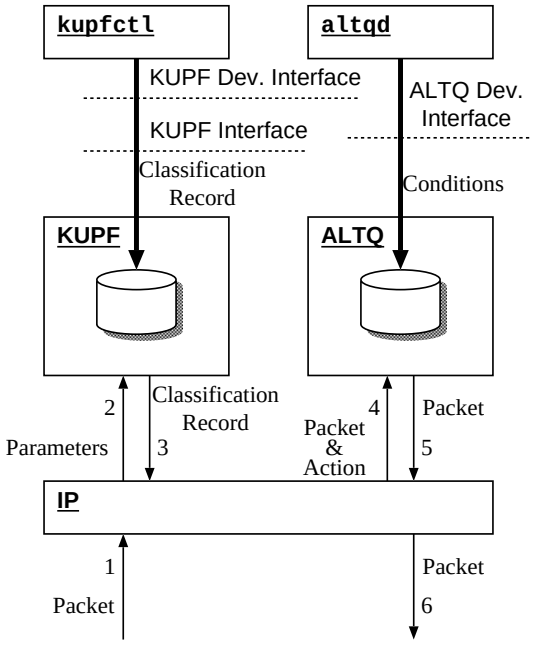

図 5.3: KUPF を用いた IP 入力フィルタ

ない場合においても，KUPF の検索速度性能を向上させるために有効であるといえる．

5.2. IP 入力フィルタへの適用

5.2.1. 設計と実装

KUPF フレームワークの適用例として，IP パケット入力フィルタを実装した．KUPF により得られたアクションで，DiffServ サービスを提供する ALTQ [9] を制御した．このシステムは図 5.3 に示すように，IP, KUPF, ALTQ のカーネル空間モジュールと `kupfctl`, `altqd` のユーザ空間プログラムから構成される．`kupfctl` は，デバイスインタフェイス経由で KUPF 分類レコード操作を呼び出し，分類レコードをカーネル内の KUPF に登録する．`altqd` は，ALTQ デバイスインタフェイスを呼び出し，DiffServ 制御の情報をカーネル内の ALTQ に登録する．IP パケットがこのシステムに入

表 5.4: IP 入力フィルタ IPv4 分類スキーマ

属性	キーパラメータ仕様	レコードパラメータ仕様	比較型
入力インタフェイス	符号なし整数	符号なし整数	符号なし整数比較
プロトコル	符号なし整数	符号なし整数	符号なし整数比較
ToS	8 ビット長ビット列	8 ビット長ビット列 8 ビット長ビット列	ビット列マスク付き比較
終点アドレス	32 ビット長ビット列	可変長ビット列	ビット列プレフィックス比較
始点アドレス	32 ビット長ビット列	可変長ビット列	ビット列プレフィックス比較
終点ポート	符号なし整数	符号なし整数	符号なし整数比較
始点ポート	符号なし整数	符号なし整数	符号なし整数比較
IPsec 情報	符号なし整数	符号なし整数	符号なし整数比較

力されると，以下の手順で処理する．

1. IP にパケットが入力される
2. IP はパケットからパラメータ群を取り出し，KUPF に送る
3. KUPF はパラメータ群に適合する分類レコードを IP に返す
4. IP はパケットと分類レコードが示すアクションを ALTQ に送る
5. ALTQ はアクションに対応する DiffServ 制御情報をパケットに適用し，IP に返す
6. IP は ALTQ で処理されたパケットを出力する

このように，KUPF を用いることでフィルタ処理がパケット制御から分離されるため，制御機構とは独立したフィルタ部分の高機能化が可能になる．また，アクションはサービスの識別番号とサービス内の制御の識別番号から構成した．これによりアクションからサービスを区別できるため，1 つのパラメータフィルタに対して複数のサービスを組み合わせることが可能である．複数の制御機構のフィルタ処理を KUPF で統合的に行うことで，フィルタ処理の効率化が図られる．

表 5.4，5.5 に IP 入力フィルタの分類スキーマを示す．表 5.1，5.2 と比較して．入力インタフェイスが増えている．これは，ALTQ は入力インタフェイス毎にフィルタを作るのに対し，本実装の IP 入力フィルタは全入力インタフェイスで共通の分類表を作るためである．

表 5.5: IP 入力フィルタ IPv6 分類スキーマ

属性	キーパラメータ仕様	レコードパラメータ仕様	比較型
入力インタフェイス	符号なし整数	符号なし整数	符号なし整数比較
プロトコル	符号なし整数	符号なし整数	符号なし整数比較
トラフィッククラス	8 ビット長ビット列	8 ビット長ビット列 8 ビット長ビット列	ビット列マスク付き比較
フローラベル	符号なし整数	符号なし整数	符号なし整数比較
終点アドレス	128 ビット長ビット列	可変長ビット列	ビット列プレフィックス比較
始点アドレス	128 ビット長ビット列	可変長ビット列	ビット列プレフィックス比較
終点ポート	符号なし整数	符号なし整数	符号なし整数比較
始点ポート	符号なし整数	符号なし整数	符号なし整数比較
IPsec 情報	符号なし整数	符号なし整数	符号なし整数比較

5.2.2. KUPF デバイスインタフェイス

カーネル空間の分類表にユーザ空間から分類表を追加・削除するために，デバイスファイルを経由したカーネルインタフェイスを実装した．デバイスファイルは，`read()`, `write()`, `ioctl()` システムコールで操作する．KUPF を制御するプログラムと状態監視するプログラムの両方が同時に動く場合を想定し，複数プログラムのデバイスファイル同時アクセスを可能としている．以下に，`read()`, `write()` で用いる KUPF メッセージヘッダと，分類レコード操作，分類表情報取得インタフェイスの概略を説明する．

KUPF メッセージヘッダは，デバイスファイルに対する `read()`, `write()` システムコールすべてに用いる．メッセージヘッダは表 5.6 をメンバとする構造体である．バイト境界に制限がある CPU でも動作可能とするため，各メンバは 4 バイト境界におく．メッセージタイプで操作の種類を区別し，分類表ハンドル，分類レコードハンドルで操作の対象を指定する．表 5.7 にメッセージタイプと操作一覧を示す．データマスクはメッセージヘッダに続くデータの種類を示す．

分類レコード操作は，分類レコードの追加・削除・逐次取得を行う．分類レコード追加は，分類表・分類レコードの情報を設定したメッセージヘッダとレコードデータを `write()` システムコールで KUPF に送る．その後，`read()` システムコールによりメッセージヘッダを取得し，メッセージヘッダから追加した分類レコードのハンドル番号を得る．分類レコード削除は，分類表・分類レコードのハンドル番号を設定したメッセージヘッダを `write()` システムコールで KUPF に送ることで行う．その後，

表 5.6: メッセージヘッダ構造体のメンバ

型	メンバ名	説明
u_int32_t	km_msglen	メッセージ全体の長さ
u_int8_t	km_version	メッセージヘッダのバージョン
u_int8_t	pad	パディング
u_int16_t	km_type	メッセージタイプ
u_int32_t	km_ctbl_handle	分類表ハンドル
int32_t	km_entry_handle	分類レコードハンドル
u_int32_t	km_data_mask	データマスク
int32_t	km_errno	エラーコード
int32_t	km_refcnt	分類レコード参照カウンタ
u_int32_t	km_use	分類レコード使用カウンタ
int32_t	km_priority	分類レコード優先度
kupf_act_t	km_action	分類レコードアクション

`read()` システムコールによりメッセージヘッダを取得し，成功不成功の結果を得る．分類レコード逐次取得は，分類表のハンドル番号を設定したメッセージヘッダを `write()` システムコールで KUPF に送り，`read()` システムコールにより，メッセージヘッダと最初の分類レコードを取得する．さらに，次の分類レコード要求するメッセージヘッダを `write()` システムコールで KUPF に送り，`read()` システムコールにより，メッセージヘッダと次の分類レコードを取得する．

カーネル空間の分類表の情報は，`ioctl()` システムコールにより取得可能である．KUPF デバイスファイルで使用可能な `ioctl()` 要求を表 5.8 に示す．分類表の統計情報取得・初期化は，全分類表に対する検索回数，分類レコード追加・削除回数等の情報を取得・初期化する．分類表情報の取得は，ハンドル番号からもしくは分類表名とユニット番号から，ハンドル番号，分類表名，ユニット番号を取得する．これにより，ハンドル番号分類表名・ユニット番号を相互に変換可能となる．分類レコード操作時に必要となる分類表ハンドル番号は，分類表生成時に動的に決定されるため，分類レコード操作前にこの `ioctl()` 要求を用いて分類表名とユニット番号からハンドル番号を得る．

5.2.3. 実験と結果

KUPF を ALTQ のコンディショナに適用し，KUPF を用いない ALTQ と KUPF を用いた ALTQ で処理速度を比較した．性能測定ネットワークを図 5.4 に示す．Sender, Router, Receiver の環境は

表 5.7: メッセージタイプ

識別名	操作内容
KUPF_LOOKUP_CTBL	ハンドル番号から分類表を検索
KUPF_NEXT_CTBL	次ハンドル番号の分類表取得
KUPF_FIRST_ENTRY	先頭分類レコードを取得
KUPF_NEXT_ENTRY	次の分類レコードを取得
KUPF_ADD_ENTRY	分類レコードを追加
KUPF_DEL_ENTRY	ハンドル番号の分類レコードを削除

表 5.8: ioctl 要求

識別名	操作内容
KUPF_GET_CTBL_STAT	分類表の統計情報を取得
KUPF_RST_CTBL_STAT	分類表の統計情報を初期化
KUPF_GET_CTBL_INFO_HANDLE	ハンドル番号から分類表情報を取得
KUPF_GET_CTBL_INFO	分類表名とユニット番号から分類表情報を取得

表 5.9 の通りである．Sender から送出されたテストパケットは，Router で IP 転送され Receiver に届く．Router 入力時にパケットはフィルタ規則に従って分類され，IP ヘッダ中の ToS フィールドの値が書き換えられる．テストパケットは，毎秒 1,000 パケット送出した．パケットの分類と ToS フィールド書き換えの処理時間を RDTSC (Read Time-Stamp Counter) 命令を用いて測定した．測定結果として，100 回の計測の平均値を用いた．

ALTQ と KUPF のフィルタ処理速度を比較するために, IPv4 ホストアドレスフィルタ（IPv4 host），IPv4 ネットワークアドレスフィルタ（IPv4 net），IPv6 ホストアドレスフィルタ（IPv6 host），IPv6 ネットワークアドレスフィルタ（IPv6 net）を使用した．IPv4 host フィルタは，受信インタフェイ

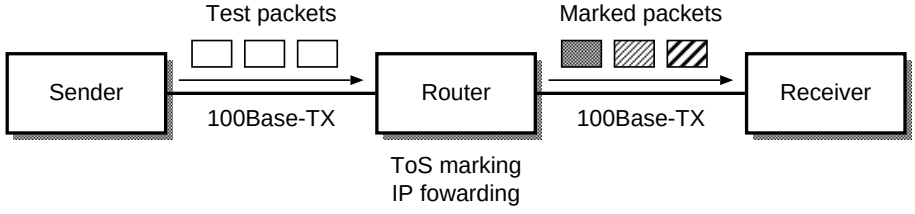

図 5.4: IP 入力フィルタ実験ネットワーク

表 5.9: IP 入力フィルタ実験環境

CPU	Pentium III (1 GHz)
メモリ	512 MByts
インタフェイス	Intel i82559
OS	NetBSD 1.6

表 5.10: IP パケット入力フィルタ処理時間

フィルタ	ALTQ	KUPF (0%)	KUPF (50%)
IPv4 host	0.002 ms	0.65 ms	0.34 ms
IPv4 net	0.021 ms	0.63 ms	0.34 ms
IPv6 host	0.25 ms	0.63 ms	0.33 ms
IPv6 net	0.25 ms	0.50 ms	0.26 ms

ス，プロトコル，終点アドレス，終点ポート番号を指定した 1,000 種類のフィルタを使用した．終点アドレスはプレフィックス長が 32 のホストアドレスを使用した．IPv4 net フィルタは，終点アドレスのプレフィックス長が 28 のネットワークアドレスであることを除き，IPv4 host と同一である．IPv6 host フィルタは，IPv6 プロトコルであることを除き IPv4 host と同一である．IPv4 net のフィルタは，終点アドレスのプレフィックス長が 64 のネットワークアドレスであることを除き，IPv6 host と同一である．

ALTQ は，IPv4 の場合は IP アドレスを 32 ビット長の整数型でアドレスを扱う．また，プレフィックス長が 32 の終点アドレスに対してハッシュ表を使用する．一方，IPv6 の場合は IP アドレスを 8 ビット長の整数型 16 個の配列でアドレスを扱う．また，フローラベルに対してハッシュ表を使用する．そのため，ALTQ において IPv4 host は高い性能，IPv6 host/net は低い性能が予想される．これに対して，KUPF においては IPv4/IPv6 に関係なくアドレスをビット列で扱っている．そのため，アドレスの種類による性能差が小さいことが予想される．

表 5.10 に ALTQ と KUPF を組み込んだ ALTQ の処理速度を示す．“ALTQ” は KUPF を組み込んでいない ALTQ を表す．KUPF の実験ではキャッシュの効果を評価するために，連続する実験パケットの属性が異なり，キャッシュヒット率が 0%となる場合と，連続するパケットが 2 個ずつ属性が同一で，キャッシュヒット率が 50%となる場合で行った．5.2 節の実験から，実運用環境におけるキャッシュヒット率は 50%に満たない程度であることが結果として得られている．“KUPF (0%)” は

KUPF を組み込んだ ALTQ で，キャッシュヒット率が 0%の場合を表す．“KUPF (50%)” は KUPF を組み込んだ ALTQ で，キャッシュヒット率が 50%の場合を表す．表 5.10 が示すように，KUPF は ALTQ より処理時間が長い．“IPv4 host” フィルタでは，KUPF と ALTQ の差が非常に大きい．これは，ALTQ が検索にハッシュ表を使用しているのに対し，KUPF は検索に逐次検索を使用しているためである．一方 IPv6 フィルタでは，差は 3 倍以内で，特にキャッシュヒット率が 50%の場合は 1.5 倍以下となった．これは，ALTQ，KUPF ともに検索に逐次検索を使用しているためである．

5.3. 実験のまとめ

5.2 節の結果から，KUPF は互換性を保持しながら従来の IP パケットフィルタに適用可能であることが分かった．また，4 日間の短い期間であるが，KUPF が実際にサービスを行うネットワークで運用可能であることが実証された．

5.2 節において，KUPF を用いることでフィルタ処理をパケット制御機構から分離でき，制御機構からパラメータフィルタを独立可能であることを示した．これにより，制御機構から独立にパケットフィルタの高機能化が可能であるとともに，複数の制御機構のフィルタ処理を KUPF で統合できる．複数の異なるサービスにおいて個々のパケットフィルタにより通信データを分類する代わりに，パラメータフィルタにより通信データを分類した後に各サービスを呼び出すことで，通信データに対するサービスの適用順序が制御可能となる．また，複数のパケットフィルタの処理の代わりに 1 回のパラメータフィルタの処理により通信データの分類が完了する．単一のサービスを想定した場合，IP パケットフィルタとしての処理速度は，IP パケットに特化した実装に比べ性能が劣ることが判明した．KUPF はキャッシュ機能の導入により，汎用性を維持しながら高速化を図っている．キャッシュによる効果により，悪条件の場合の IP パケットに特化した実装との性能差は，わずかであることが判明した．

第6章 仮想包囲矩形に基づくKUPF高速化

第 4 章の KUPF フレームワークの実装は，抽象化したモデルをそのまま実装しているため，高速化にはいくつかの問題がある．問題点の一つは，ポリシの抽出をサービス非依存に行う第 1 ステージが複数のパラメータを扱うため，単一のパラメータに対するフィルタで培われてきた高速化手法が使えないことである．

本章では，多次元空間上での効率的な検索手法として知られる R*-tree [3] を応用することで，KUPF の第 1 ステージにおける検索処理を高速化した．提案手法では，パラメータフィルタの特性に対応するため，仮想的な包囲矩形表現を導入し，R*-tree に対してデータ挿入アルゴリズムに変更を加えた．本手法の有効性を示すために，パラメータフィルタを実装し，評価実験を行った．

6.1. R*-tree

多次元空間上で効率的に検索できる手法として R-tree [12] が知られている．R-tree は多次元空間上の領域を包囲矩形で表し，その包囲矩形を葉ノードとする階層化された探索木を用いて検索する．R*-tree[3] は R-tree の改良手法で，データ挿入アルゴリズムの改善により性能を向上させている．

R-tree は，データの扱いを容易にするために，多次元空間上のデータ領域を最小包囲矩形（Minimum Bounding Rectangle; MBR）で表現する．MBR は，データ領域を包囲する矩形で，かつ各辺が空間軸に平行である領域である．2 次元上の MBR の例を図 6.1 に示す．斜線領域の MBR は，2 点 $P_1(x_1, y_1)$, $P_2(y_1, y_2)$ を対角とする長方形となる．MBR 利用により，複雑なデータ領域を簡単化して表現できる．R-tree はデータ領域の MBR を葉ノードとした多分木である（図 6.2）．各ノードは MBR と複数

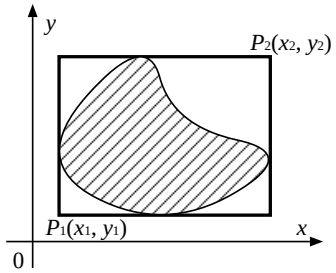

図 6.1: MBR の例

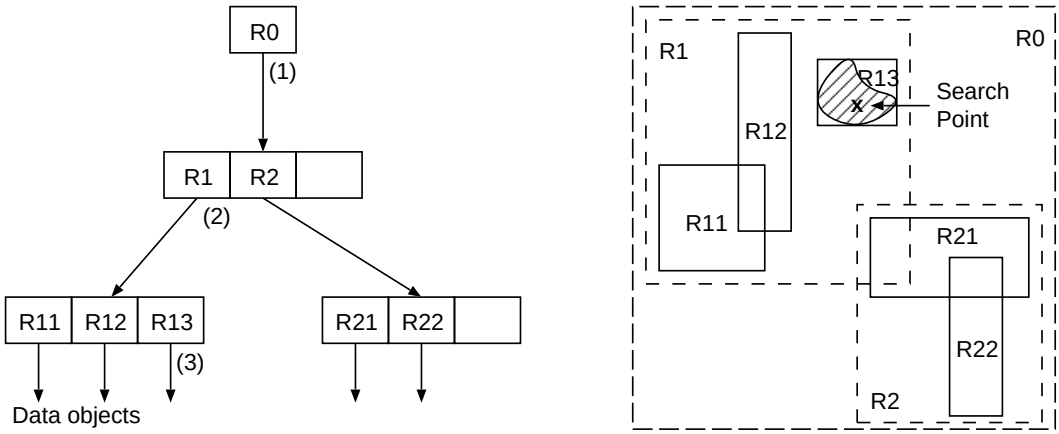

図 6.2: R-tree の構成

の子ノードへのポインタを要素とする．各ノードの MBR は，子ノードの MBR すべてを含む MBR とする．根ノードは全データ領域を含む MBR となる．データ検索時は，根ノードから葉ノードまで順に，検索データのオブジェクトと交差する MBR を要素とするノードをたどっていく．検索データの領域と葉ノードのデータ領域が交差すれば，検索結果となる．図 6.2 の “×” で示された点で表されるデータを検索する時，探索木のノードを (1) → (2) → (3) → の順にたどる．矩形 R13 のオブジェクト（斜線部分）と点 “×” を比較すると交差しており，検索結果となる．

6.2.　R*-tree のパラメータフィルタへの適用

R*-tree はデータオブジェクトをすべて MBR で表現する．そのため, パラメータフィルタに R*-tree を適用するには，フィルタ規則を多次元空間上の矩形で表現する必要がある．フィルタ規則のパラメータには以下の特性がある．

- パラメータ毎に定義域が著しく異なる．
- パラメータの区間がフィルタ毎に著しく異なる．特に，あらゆるデータに適合するワイルドカードは，全区間のパラメータになる．
- 1 次元上の区間で表現できないパラメータがある．

これらの特性は，各パラメータを多次元空間上の各次元の区間に変換する時，探索木を構築する時の問題となる．本提案では，仮想包囲矩形（Virtual Bounding Rectangle; VBR）[13] の導入，パラメータフィルタに適したデータ挿入アルゴリズムの適用によりこれらの問題を解決した．

6.2.1. 仮想包囲矩形を用いた正規化

パラメータフィルタでは，パラメータ毎に著しく異なる定義域のパラメータを扱う．一例として TCP, UDP のポート番号と IPv6 アドレスがある．TCP, UDP ポート番号は符号なし 16 ビット長整数で，IPv6 アドレスは 128 ビット長のビット列である．ビット列は整数と見なすことが可能だが，16 ビット長と 128 ビット長では定義域の広さが著しく異なる．R*-tree の探索木構築には，ノードを挿入するノードの選択，ノードを分割する軸の選択において Area, Length, Margin の値を使う．軸毎に単位が異なり定義域が著しく異なる場合，これらの値は定義域が広い軸の影響を多く受ける．そのため，矩形の分割は定義域の広い軸に偏り，検索性能が低下する．性能改善には Area, Length, Margin の正規化が有効である [14].

本論文では，仮想包囲矩形（Virtual Bounding Rectangle; VBR）[13] の導入による正規化を提案する．VBR は，MBR を包含し，かつ近似する矩形で，親ノードを基準とした各次元のサイズが数ビットの相対座標によりノードの矩形を表す．VBR は座標値を少ないビット数データに近似することでデータ処理を軽減する提案だが，子ノードの矩形を相対座標に変換することに着目した．親ノードを基準とした相対座標を次元毎に定義域の異なる探索木に用いれば，親ノード毎に Area, Length, Margin が正規化される．これにより，定義域の格差を要因とする探索木の偏りを解消できる．

6.2.2. ビットマスク一致の区間表現

ビット列からなるパラメータを考える時，特定のビットの ‘0’, ‘1’ に注目してパラメータを比較することがある．IP ヘッダの ToS フィールドのビット値にるパケット分類はこの一例である．ビット単位のパラメータ比較をビットマスク一致と呼ぶ.

定義 1 n ビット長ビット列 $X = \{x_1, x_2, \ldots, x_n\}$，$Y = \{y_1, y_2, \ldots, y_n\}$，$M = \{m_1, m_2, \ldots, m_n\}$ が存在し，式 *(6.1)* が成立する時，X，Y はマスク M において**ビットマスク一致**する．

$$x_i \cdot m_i = b_i \cdot m_i, \quad (i = 1, \ldots, n) \tag{6.1}$$

ビットマスク一致は各ビット毎の比較であるため，パラメータそのものを座標軸上の区間で表現できない．パラメータを VBR の 1 辺として扱うために，ビット列中の ‘0’, ‘1’ の数に着目したビットマスク一致の区間表現を提案する．初めに以下の関数を定義する．

$$f(x) = \begin{cases} -1 & (x = 0) \\ +1 & (x = 1) \end{cases} \tag{6.2}$$

$$f_0(b,m) = \begin{cases} +1 & (b = 1 \wedge m = 1) \\ -1 & (\text{otherwise}) \end{cases} \tag{6.3}$$

$$f_1(b,m) = \begin{cases} -1 & (b = 0 \wedge m = 1) \\ +1 & (\text{otherwise}) \end{cases} \tag{6.4}$$

X と B がマスク M においてビットマスク一致する時, $X = \{x_1, x_2, \ldots, x_n\}$, $B = \{b_1, b_2, \ldots, b_n\}$, $M = \{m_1, m_2, \ldots, m_n\}$ として次式が成立する.

$$f_0(b_i, m_i) \le f(x_i) \le f_1(b_i, m_i), \quad (i = 1, 2, \ldots, n) \tag{6.5}$$

これより，ビットマスク一致の必要条件として以下が得られる.

$$\sum_{1 \le i \le n} f_0(b_i, m_i) \le \sum_{1 \le i \le n} f(x_i) \le \sum_{1 \le i \le n} f_1(b_i, m_i) \tag{6.6}$$

マスク M におけるビットマスク一致より多くの値に一致するマスクとして，マスク M のビットの 1 つを 0 にしたマスク $M' = \{m'_1, m'_2, \ldots, m'_n\}$ を考える．ある $j(1 \le j \le n)$ について次式が成立する.

$$\begin{aligned} f_0(b_j, m'_j) &= f_0(b_j, m_j) - 1 \\ f_1(b_j, m'_j) &= f_1(b_j, m_j) + 1 \\ f_0(b_i, m'_i) &= f_0(b_i, m_i) \\ f_1(b_i, m'_i) &= f_1(b_i, m_i), \quad (1 \le i \le n \wedge i \ne j) \end{aligned}$$

ゆえに,

$$\sum_{1 \le i \le n} f_0(b_i, m'_i) < \sum_{1 \le i \le n} f_0(b_i, m_i) \tag{6.7}$$

$$\sum_{1 \le i \le n} f_1(b_i, m_i) < \sum_{1 \le i \le n} f_1(b_i, m'_i) \tag{6.8}$$

したがって，ビットマスク一致は式 (6.6) の第 1 項を下限，第 3 項を上限とする区間で表現できる．この区間表現を用いて，ビットマスク一致のパラメータを VBR の 1 辺に変換する.

6.2.3.　手続き ChooseSplitIndex と交差面積

1 つの親ノードの子ノードが上限を越えると，親ノードが 2 つに分割される．子ノードをどちらの親ノードに分配するか決める時，手続き ChooseSplitIndex が使われる．この手続きは，分割した領

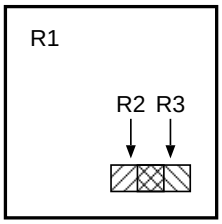

図 6.3: 手続き ChooseSplitIndex と交差面積

域の交差領域の面積（交差面積）が最小となる分割を選択する．定義域の全区間を辺とする MBR を持つ子ノードがある場合，小さな MBR を持つ子ノードが全区間を辺とする MBR と同じ親ノードに分配される．図 6.3 で矩形 R1 は 2 辺が全区間を辺とする MBR，矩形 R2, R3 は小さな矩形の MBR とする．R2, R3 は横長の長方形で，互いの一部が重なっている．これら 3 つの矩形を 2 つに分割する．(R1), (R2, R3) の 2 つに分割する場合，交差領域は R2 と R3 を含む MBR となる．(R1, R2), (R3) もしくは (R1, R3), (R2) の 2 つに分割する場合，交差領域はそれぞれ R3 もしくは R2 の矩形となる．そのため，分割した領域の交差領域が最小となる分割を選択すると，R2 もしくは R3 のどちらかが R1 と同じ親ノードに分配される．

以上の理由により，全区間を辺とする MBR を持つ子ノードの親ノードは，他の子ノードをノード数の上限まで持つ．同じ親ノードの子ノードが別の次元の辺で大きな区間を持つ場合，互いに引き寄せ合うため，巨大な MBR を持つノードは分割されない一方で，隣接する小さな MBR を持つノードは分割される．そのため，探索木でノードが特定のノードの下に偏り，検索性能が低下する．パラメータフィルタでは，ワイルドカードにより全区間が頻繁に現れるため，この問題が顕在化する．

ノード分割時のノードの偏りを避けるために，子ノード数を考慮した交差領域の比較法を提案する．R*-tree では交差面積の値をそのまま比較するが，これを面積が小さい MBR を持つ方の親ノードの子ノード数で割った値で比較する．この比較法は，小さい MBR を持つ親ノードの子ノード数について，1 ノードあたりの交差面積で比較するため，交差面積が倍となってもノード数増加が倍以下となる分割が選択される．図 6.3 では，(R1), (R2, R3) の交差面積は 2 で割り，(R1, R2), (R3) もしくは (R1, R3), (R2) の領域は 1 で割り，比較する．この場合，(R1), (R2, R3) の分割が選択される．これにより，隣接する小さな MBR 群が巨大な MBR から分離される．

6.2.4. 手続き ChooseSubtree と交差コスト，領域コスト

手続き ChooseSubtree は，探索木にデータを挿入する部分木を決定する．この手続きは，挿入データの MBR をある子ノードに追加した時に，子ノード間の交差領域の面積増加（交差コスト），子ノードの矩形領域の面積増加（領域コスト）が最小となる子ノードを選択する．ある子ノード N の MBR

表 6.1: 仮想包囲矩形の例

分類レコード	ビット/マスク	区間	範囲	区間
R_1	0000/0000	$[-4, 4]$	0–100	$[-4, 4]$
R_2	0000/1001	$[-4, 0]$	4–5	$[-4, -3]$
R_3	0100/0110	$[-2, 2]$	4–6	$[-4, -3]$
R_4	0100/0110	$[-2, 2]$	94–95	$[3, 4]$

が定義域の全領域となる場合，その MBR はそれ以上大きくならないため，ノード N を選択した場合の交差コスト，領域コストは挿入データに関係なく 0 となる．したがって，手続き ChooseSubtree は常にノード N を選択するため，探索木に偏りが生じる．パラメータフィルタでは，ワイルドカードにより全区間が頻繁に現れるため，この問題が顕在化する．

データ挿入時に全領域矩形と小さな矩形が分割されるように，追加されるノードの交差面積，領域面積を考慮した交差コスト，領域コストの算出法を提案する．R*-tree ではノード追加前から追加後の面積増加をコストと考え，次式によりにコストを算出する．

$$(\text{交差コスト}) = (\text{ノード追加後交差面積}) - (\text{ノード追加前交差面積}) \tag{6.9}$$

$$(\text{領域コスト}) = (\text{ノード追加後領域面積}) - (\text{ノード追加前領域面積}) \tag{6.10}$$

本提案では，ノード追加前から追加後の面積増加に加えて，追加されるノードについて追加される前から追加された後の面積増加もコストと考え，次式によりコストを算出する．

$$\begin{aligned}(\text{交差コスト}) =& 2 \cdot (\text{ノード追加後交差面積}) \\ & - (\text{ノード追加前交差面積}) - (\text{追加ノード交差面積})\end{aligned} \tag{6.11}$$

$$\begin{aligned}(\text{領域コスト}) =& 2 \cdot (\text{ノード追加後領域面積}) \\ & - (\text{ノード追加前領域面積}) - (\text{追加ノード領域面積})\end{aligned} \tag{6.12}$$

これにより，領域面積の大きい子ノードへ挿入データの MBR を追加する場合のコストが大きくなり，定義域の全領域を MBR とするノードから小さな MBR を持つノードが分離される．

6.2.5.　探索木の作成

表 3.1 の分類レコードを用いて, 探索木のノード分割, データ挿入部分木の決定を説明する．説明を簡単にするために, 仮想包囲矩形の座標値を -4 から $+4$ の整数値とする．R_1 を基準とした, 仮想包囲矩形の

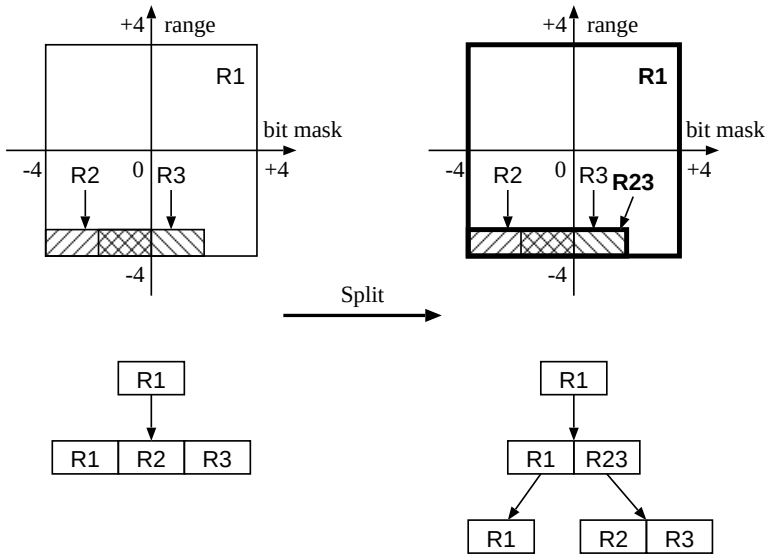

図 6.4: ノード分割

表 6.2: 交差面積

分割	交差面積 (R*-tree)
$(R_1), (R_2, R_3)$	3 (6)
$(R_1, R_2), (R_3)$	4 (4)
$(R_1, R_3), (R_2)$	4 (4)

座標軸上の区間を表 6.1 に示す．ビットマスク一致の区間は, 式 (6.6) により得られる．範囲一致の区間は範囲を 0 から 100 の値域から -4 から 4 の整数値に正規化した値で, $p_0 = 0, p_1 = 100, q_0 = -4, q_1 = 4$ として，範囲 r_0–r_1 に対して次式により $[v_0, v_1]$ と得られる．

$$v_0 = q_0 + \left\lfloor \frac{r_0(q_1 - q_0)}{p_1 - p_0} \right\rfloor \tag{6.13}$$

$$v_1 = q_0 + \left\lceil \frac{r_1(q_1 - q_0)}{p_1 - p_0} \right\rceil \tag{6.14}$$

図 6.4 は，R_1, R_2, R_3 がある探索木のノードが分割される様子を示している．6.2.3 節の算出法による子ノード数を考慮した交差面積を表 6.2 に示す．これより，交差面積が最小の $(R_1), (R_2, R_3)$ の分割が選択される．

図 6.5 は, R_1, R_2, R_4 がある探索木に R_3 が挿入される様子を示している．6.2.4 節の式 (6.11), (6.12) による交差コスト，領域コストを表 6.3 に示す．これより，交差コストが最小の R_2 の部分木が選択される．

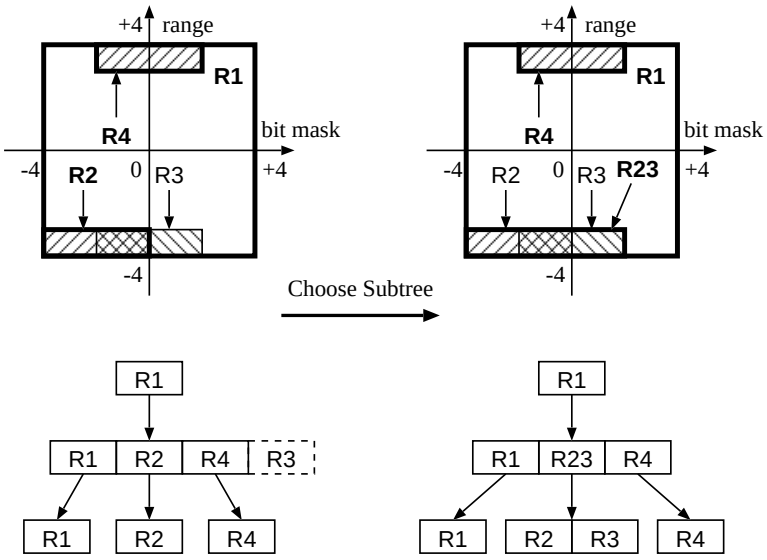

図 6.5: 挿入部分木選択

表 6.3: 交差コスト，領域コスト

部分木	交差コスト (R*-tree)	領域コスト (R*-tree)
R_1	6 (0)	60 (0)
R_2	4 (2)	4 (2)
R_4	10 (6)	8 (4)

6.3. 設計と実装

仮想包囲矩形に基づく探索木を用いた KUPF フレームワークの拡張，KUPF-VR を NetBSD 1.6 上で C 言語を用いて実装した．比較型，分類表のデータ構造の拡張により，KUPF-VR を実現するための区間の演算手続き，探索木はすべて分類表からたどることを可能とした．パラメータ操作，分類レコード操作，分類操作は KUPF フレームワークと同一のプログラムインタフェイスを持つ．この互換性により，KUPF フレームワークを使用するプログラムは分類表作成部分を書き換えるだけで KUPF-VR に対応可能である．

KUPF-VR において，パラメータの比較はすべて MBR の比較に置換される．そのため，パラメータの比較演算を定義する比較型を区間を扱うために拡張した．比較型の構造を図 6.6 に示す．KUPF 比較型（図 6.6）と比べ，区間パラメータを扱うデータ（下線部）を追加した．これらのデータにより，パラメータを区間に変換し，区間同士で比較演算する．例えば，符号なし整数比較の場合，以下

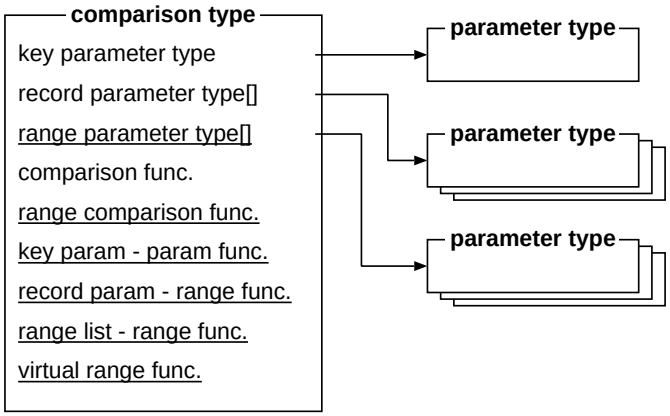

図 6.6: KUPF-VR 比較型

が登録される．

区間パラメータ群のパラメータ型

符号なし整数 2 個（下限値，上限値）

区間比較関数

区間 $R_1(a_1, b_1)$ が区間 $R_2(a_2, b_2)$ に含まれるか判定

キーパラメータ－区間変換手続き

キーパラメータ x に対し，区間 $R(x, x)$ を返す

レコードパラメータ－区間変換手続き

レコードパラメータ x に対し，区間 $R(x, x)$ を返す

区間一覧－区間変換手続き

区間一覧 $R_i(a_i, b_i), (i = 1, ..., n)$ に対し，区間 $R(min(a_i), max(b_1))$ を返す

仮想区間手続き

親ノードの区間 $R_0(a_0, b_0)$ の区間 $R_1(a_1, b_1)$ に対し，区間 $R(\frac{a_1-a_0}{b_0-a_0}, \frac{b_1-a_0}{b_0-a_0})$ を返す

仮想区間はすべて整数値を上限，下限とする区間で表現する．上限，下限は固定ビット長で表される整数とすることで，探索時の処理を軽減する．

表 6.4 に表 4.4 の比較型に対する区間表現の型を示す．符号なし整数比較，符号付き整数比較は，数学的な上限値・下限値で区間を表す．固定長ビット列比較，可変長ビット列比較はパラメータを 2 進数表現と見なして，同様に数学的な上限値・下限値で区間を表す．

固定長ビット列先頭比較は, レコードパラメータに適合するキーパラメータ集合を区間と考える．これにより, 区間は可変長ビット列で表現可能である．区間比較関数は, 可変長ビット列 $A = \{a_1, a_2, ..., a_n\}$,

表 6.4: 区間表現

比較型	区間パラメータ群パラメータ型
符号なし整数比較	符号なし整数（上限，下限）
符号付き整数比較	符号付き整数（上限，下限）
固定長ビット列比較	固定長ビット列（上限，下限）
固定長ビット列先頭比較	可変長ビット列
固定長ビット列マスク付き比較	固定長ビット列（ビット値，マスク値）
可変長ビット列比較	可変長ビット列（上限，下限）

$B = \{b_1, b_2, ..., b_m\}$ に対して次式が成立するとき，ビット列 A はビット列 B に含まれると判定する．

$$a_i = b_i, (i = 1, 2, ..., m) \tag{6.15}$$

キーパラメーター区間変換手続き, レコードパラメーター区間変換手続きはともに, 同じビット長, 同一ビット列の可変長ビット列を返す．区間一覧－区間変換手続きは, ビット列 $R_i = \{b_1, b_2, ...\}, (i = 1, 2, ...)$ に対し，全ビット列で共通する先頭のビット列を抜き出した可変長ビット列を返す．仮想区間手続きは，親ノードのビット列 $P = \{p_1, p_2, ..., p_n\}$，子ノードのビット列 $Q = \{q_1, q_2, ..., q_m\}$ に対し，

$$A = \{a_1, a_2, ..., a_m\} \tag{6.16}$$

$$a_i = \begin{cases} p_i (i = 1, 2, ..., n) \\ 0 (otherwise) \end{cases}$$

$$B = \{b_1, b_2, ..., b_m\} \tag{6.17}$$

$$b_i = \begin{cases} p_i (i = 1, 2, ..., n) \\ 1 (otherwise) \end{cases}$$

とし，可変長ビット列を 2 進数表現と見なして区間 $R(\frac{Q-A}{B-A}, \frac{Q-A}{B-A})$ を返す．

可変長ビット列比較は，レコードパラメータにビットマスク一致するキーパラメータ集合を区間と考える．これにより, 区間はビット値とマスク値の固定長ビット列で表現可能である．n ビット長のビット列 $A = \{a_1, a_2, ..., a_n\}$, $B = \{b_1, b_2, ..., b_n\}$, マスクビット列 $P = \{p_1, p_2, ..., p_n\}$, $Q = \{q_1, q_2, ..., q_n\}$ が存在すると考える．次式が成立するとき $\{A, P\}$ は $\{B, Q\}$ に含まれると判定する．

$$p_i = q_i, a_i = b_i, (q_i = 1) \tag{6.18}$$

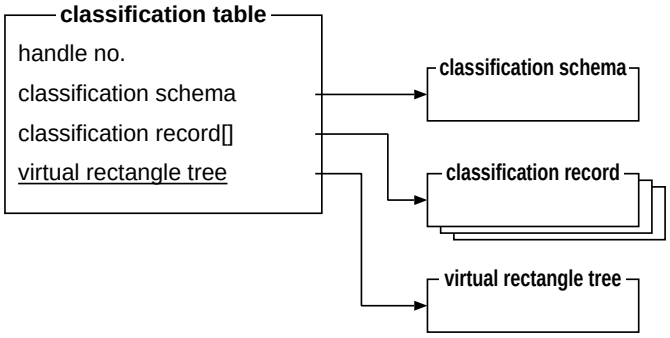

図 6.7: KUPF-VR 分類表

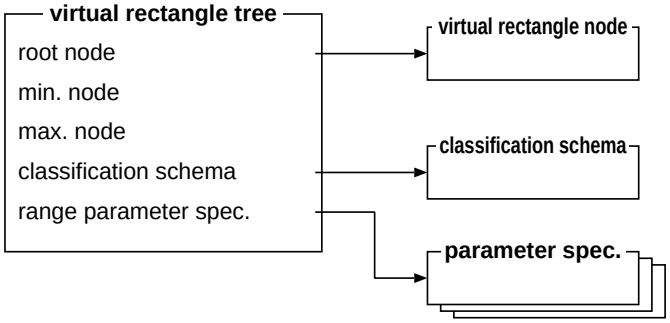

図 6.8: KUPF-VR 探索木

キーパラメーター区間変換手続きはキーパラメータ A に対し，$\{A, P\}(p_i = 1)$ を返す．レコードパラメーター区間変換手続きはレコードパラメータ $\{A, P\}$ に対し，$\{A, P\}$ を返す．仮想区間手続きは，式 (6.6) の第 1 項を下限，第 3 項を上限とする区間を返す．

分類表の構造を図 6.7 に示す．KUPF 分類表（図 4.8）と比べ，仮想包囲矩形の探索木（下線部）が増えている．この探索木の構造を図 6.8 に，ノードエントリの構造を図 6.9 に示す．木構造構築のために，ノードエントリへの参照を保持する．エントリの矩形は各次元毎の区間を表すパラメータ群，仮想包囲矩形は各次元毎の正規化された区間を表す整数値群で保持する．

図 6.2 に示した R-tree は，KUPF-VR において図 6.10 の木構造で表現する．KUPF-VR の各エントリが R-tree の MBR に対応し，四角で囲まれたエントリ群は R-tree のノードに対応する．各エントリは木構造を構築するために，上位エントリ，下位エントリ，水平エントリへの 3 つの接続を保持する．最上位のエントリ（根ノードエントリ）は下位エントリへの接続のみ保持し，最下位のエントリ（葉ノードエントリ）は下位エントリの代わりに分類レコードへの接続を保持する．

探索木からの検索は以下の手順で実行する．

1. ノードリスト $\mathbf{N}$ を根ノード，分類レコードリスト $\mathbf{R}$ を空とする

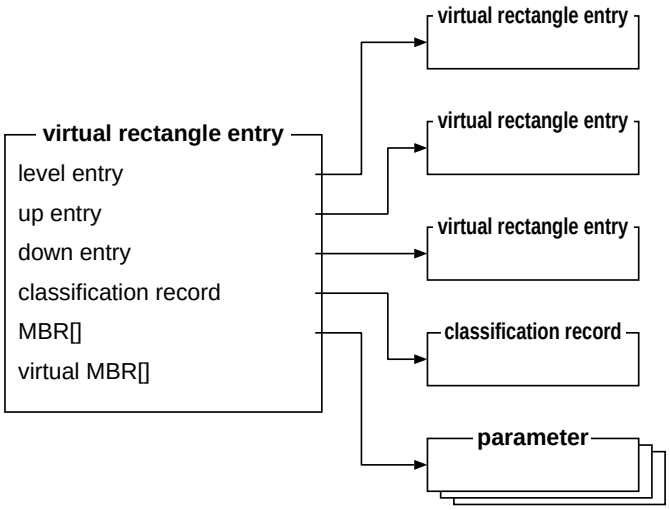

図 6.9: KUPF-VR 探索木ノードエントリ

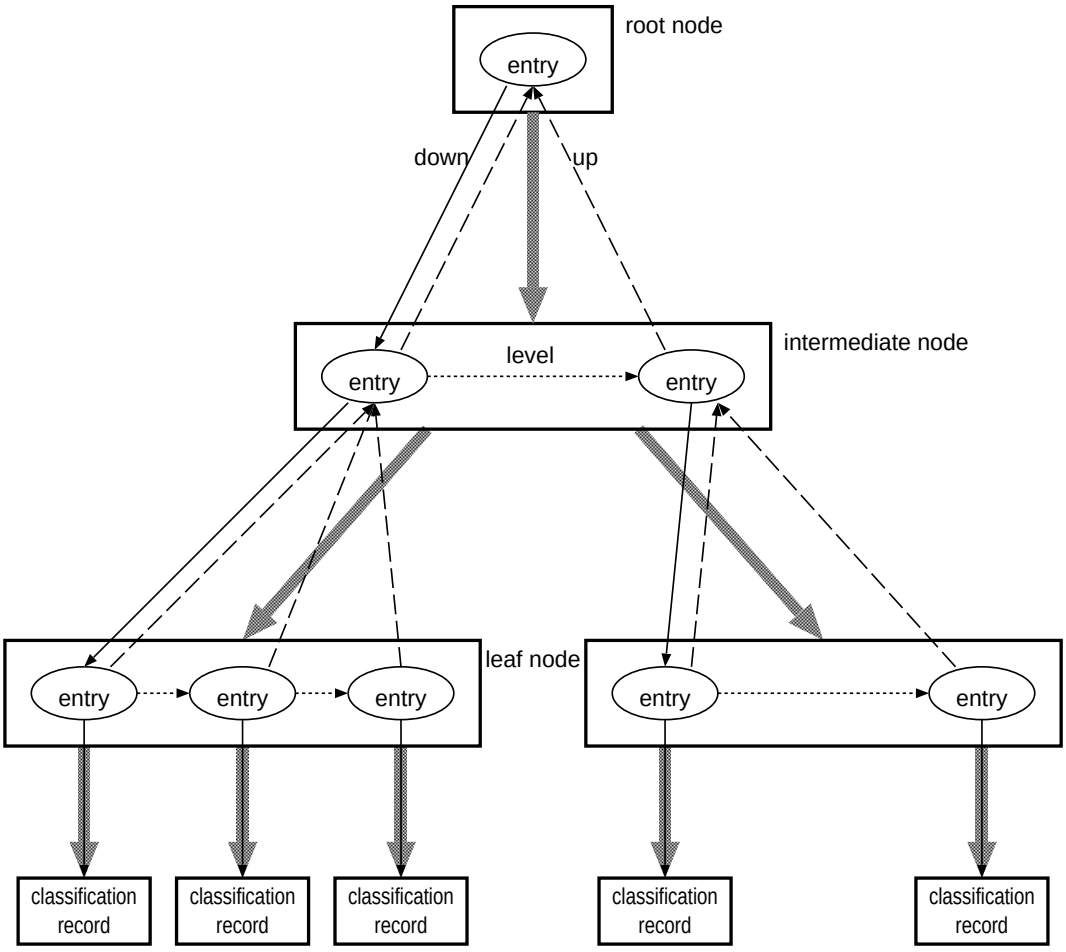

図 6.10: KUPF-VR の木構造

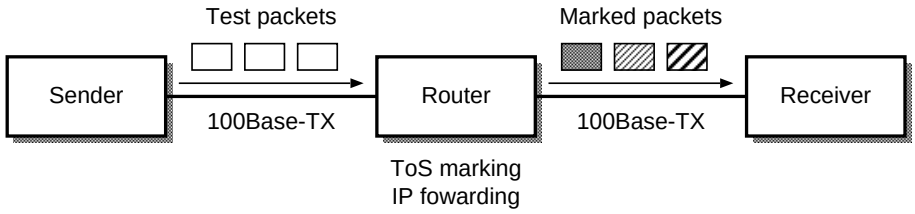

図 6.11: KUPF-VR 実験ネットワーク

2. ノードリスト **N** から 1 ノード取り出し，ノード N とする．ノードがなければ，分類レコードリスト **R** を結果として終了する

3. キーパラメータからノード N を基準とした VBR を算出する

4. ノード N が葉ノードでなければ，キーパラメータとノード N の各エントリを VBR 同士，キーパラメータとレコードパラメータで比較し，適合したエントリの下位ノードをノードリスト **N** に加える

5. ノード N が葉ノードであれば，キーパラメータとノード N の各エントリを VBR 同士，キーパラメータとレコードパラメータで比較し，適合したエントリの分類レコードを分類レコードリスト **R** に加える

6. 2 に返る

6.4. 評　　価

本提案による速度性能向上の効果を検証するために，VBR を用いた KUPF（KUPF-VR）を実装し，従来手法との比較実験を行った．比較対象として，従来の KUPF と IP パケット処理に特化した ALTQ [9] を用いた．

6.4.1. 測定システム

性能測定ネットワークを図 6.11 に示す．Sender, Router, Receiver の環境は表 6.5 の通りである．Sender から送出されたテストパケットは，Router で IP 転送され Receiver に届く．Router 入力時にパケットはフィルタ規則に従って分類され，IP ヘッダ中の ToS フィールドの値が書き換えられる．テストパケットは，毎秒 1,000 パケット送出し，終点アドレス，終点ポートを 1 パケットおきに変化させた．パケットの分類と ToS フィールド書き換えの処理時間を RDTSC (Read Time-Stamp Counter)

表 6.5: KUPF-VR 実験環境

CPU	Pentium III (1 GHz)
メモリ	512 MByts
インタフェイス	Intel i82559
OS	NetBSD 1.6.1

表 6.6: 実験に用いたフィルタ

フィルタ	入力インタフェイス	プロトコル	終点アドレス	終点ポート
IPv4 host	fxp0	UDP	192.168.4.0 –	10,000 – 10,099
IPv4 net	fxp0	UDP	192.168.4.0/28 –	10,000 – 10,099
IPv6 host	fxp0	UDP	fec0:0:0:4::0 –	10,000 – 10,099
IPv6 net	fxp0	UDP	fec0:0:0:4000::/64 –	10,000 – 10,099

命令を用いて測定した．測定結果として，100 回の計測の平均値 14 サンプル中の中間値 10 サンプルの平均値を用いた．1 ノードあたりの子ノード数の上限値は 50，下限値は 20 とした．仮想区間の上限値，下限値の精度は 16 ビット長整数とした．

6.4.2. 測 定 結 果

従来手法との比較を行うため，表 6.6 のフィルタを用いて処理時間を測定した．測定結果を図 6.12, 6.13，6.14，6.15 に示す．グラフの “(M)”, “(N)” は，テストパケットの違いを表し，それぞれいずれかのフィルタに適合する場合と，いずれのフィルタにも適合しない場合を示す．いずれのフィルタにおいても，KUPF-VR は KUPF に比べ性能が改善していることが分かる．KUPF-VR と ALTQ の比較を図 6.16，6.17，6.18，6.19 に示す．ALTQ は，IPv4 の場合にはネットマスク長が 32 の終点アドレス，IPv6 の場合にはフローラベルから生成したハッシュ値毎の線形リストにフィルタを保持する．終点アドレスのネットマスク長が 32 でないフィルタ，もしくは終点アドレス・フローラベルがワイルドカードのフィルタは，ワイルドカード用の線形リストに保持する．検索時には，パケットの終点アドレス，フローラベルからハッシュ値を求め，ハッシュ値に対応する線形リストを逐次検索する．ハッシュ値に対応する線形リストで検索が失敗した場合は，ワイルドカード用の線形リストを逐次検索する．ALTQ は，IPv4 host フィルタにおいてはフィルタ数に関係なくほぼ一定の処理時間を維持して

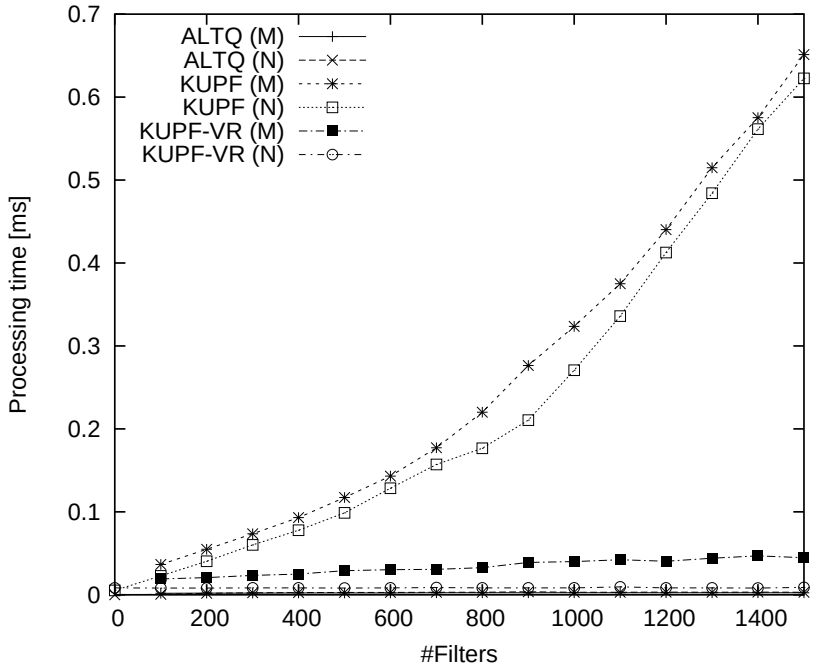

図 6.12: フィルタ数に対する処理時間変動 (IPv4 host)

いるが，IPv4 network, IPv6 host/network においてはフィルタ数の増加につれて処理時間が急激に増加している．ネットマスク長が 32 の IPv4 アドレスのフィルタに対してのみ，ALTQ はハッシュ表による高速化が有効に働いたため，この結果が得られたと考えられる．これに対して，KUPF-VR は IP アドレスに特化しない高速化を行っているため，フィルタの違いに関係なく処理時間が同じ傾向を示している．KUPF-VR は ALTQ に比べ，フィルタ数の増加による処理時間の増加はゆるやかである．そのため，ALTQ 処理時間がほぼ一定の IPv4 host を除き，フィルタ数が多い場合は KUPF-VR が優れた性能を示している．

インターネット上で維持されている IPv4 の全経路数は 2003 年 8 月において約 124,000 経路 [15] である．先の実験で ALTQ が全般に優れていた IPv4 host フィルタにおいて，全経路数にほぼ等しい 100,000 経路までフィルタ数を増やした場合の結果を図 6.20 に示す．この結果は，フィルタ数が 100,000 を越えるとき，フィルタに適合しないパケットの処理時間が逆転し，KUPF-VR の方が優れていることを示す．

6.4.3. メモリ使用量

KUPF-VR は探索木を構築するため，KUPF に比べメモリ使用量が増加する．表 6.7 に KUPF と KUPF-VR のメモリ使用量を示す．IPv4 network フィルタの場合，KUPF-VR は探索木のために，

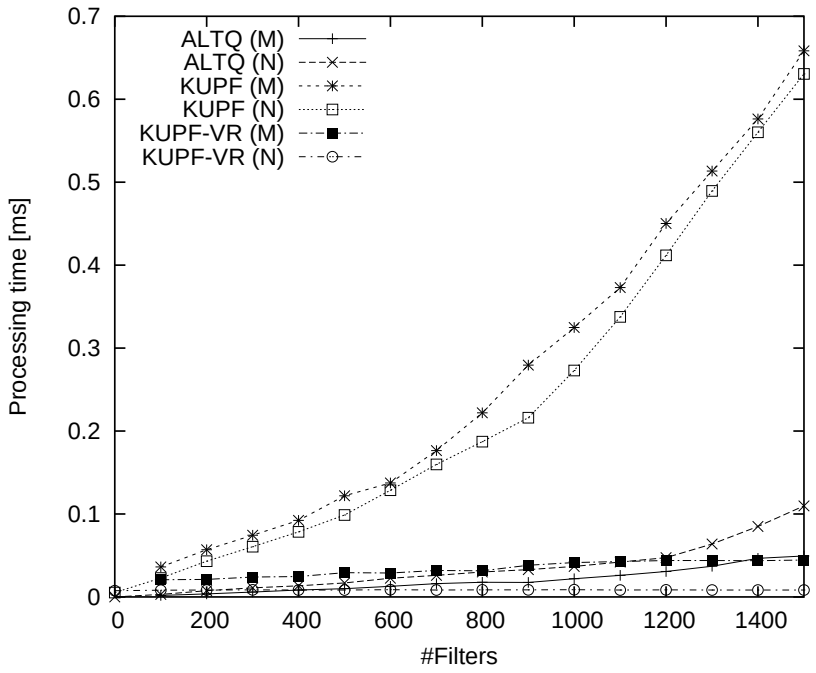

図 6.13: フィルタ数に対する処理時間変動 (IPv4 net)

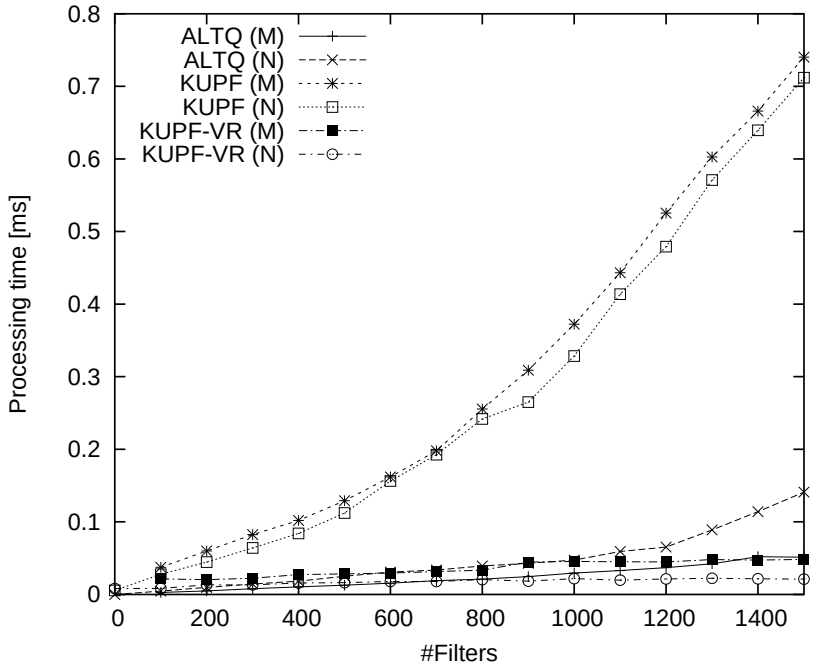

図 6.14: フィルタ数に対する処理時間変動 (IPv6 host)

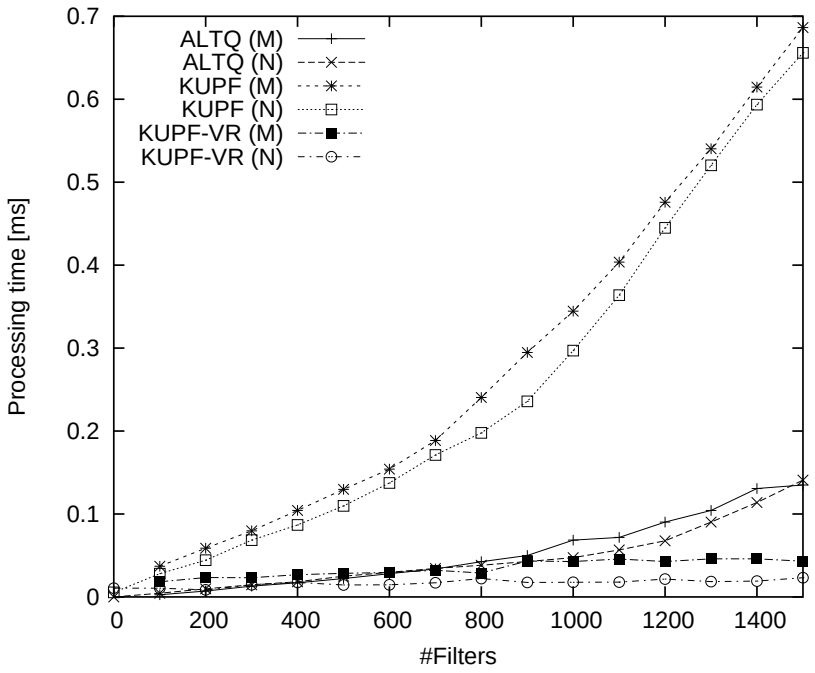

図 6.15: フィルタ数に対する処理時間変動 (IPv6 net)

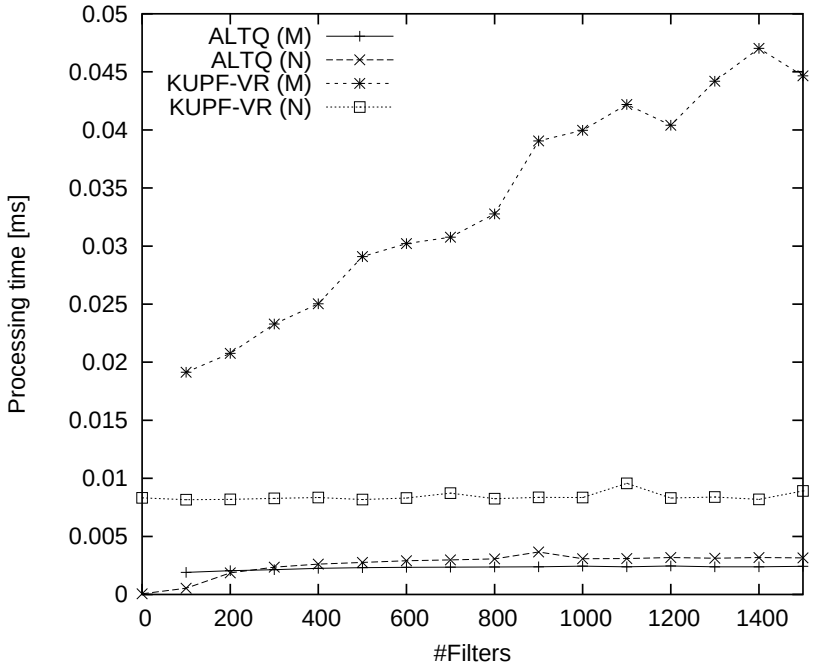

図 6.16: フィルタ数に対する処理時間変動 (ALTQ, KUPF-VR: IPv4 host)

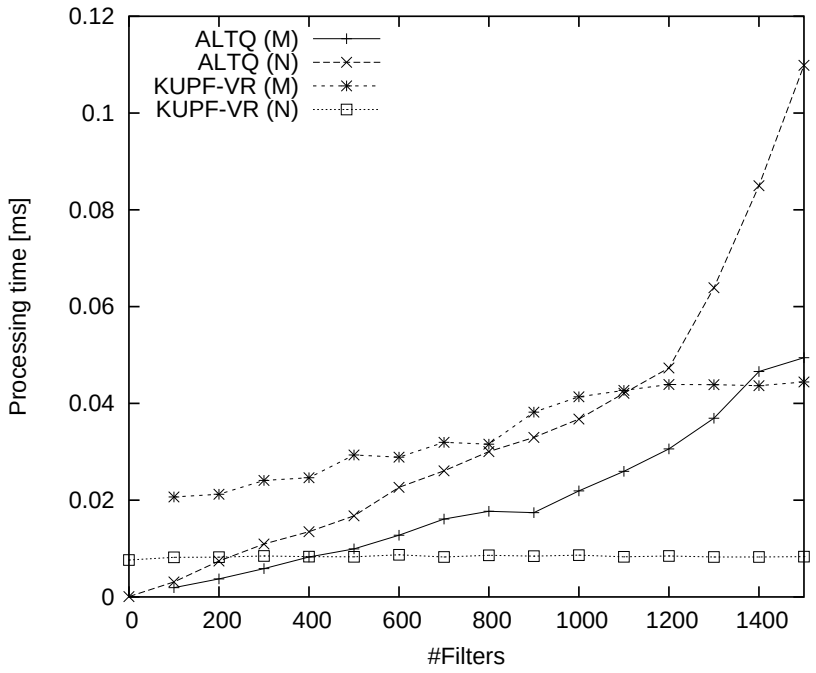

図 6.17: フィルタ数に対する処理時間変動 (ALTQ, KUPF-VR: IPv4 net)

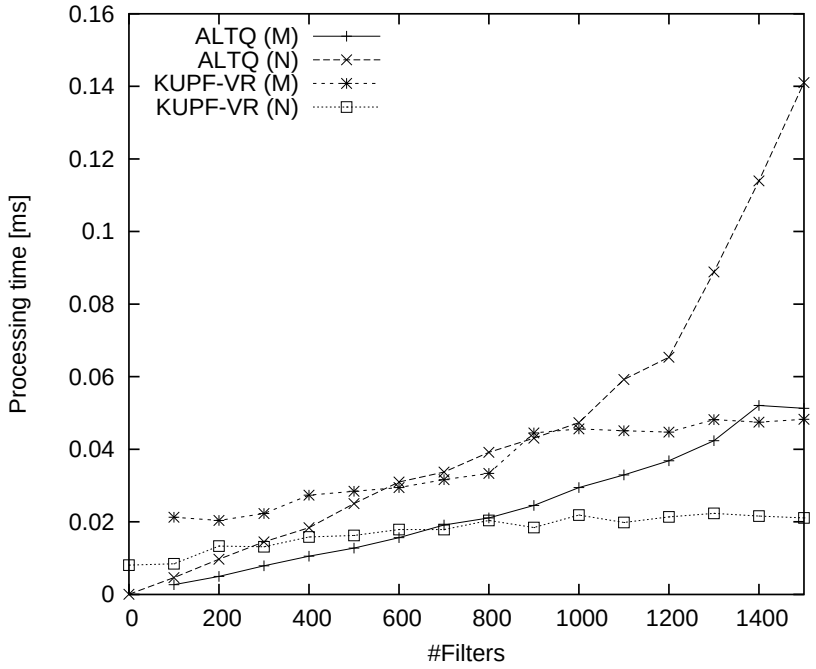

図 6.18: フィルタ数に対する処理時間変動 (ALTQ, KUPF-VR: IPv6 host)

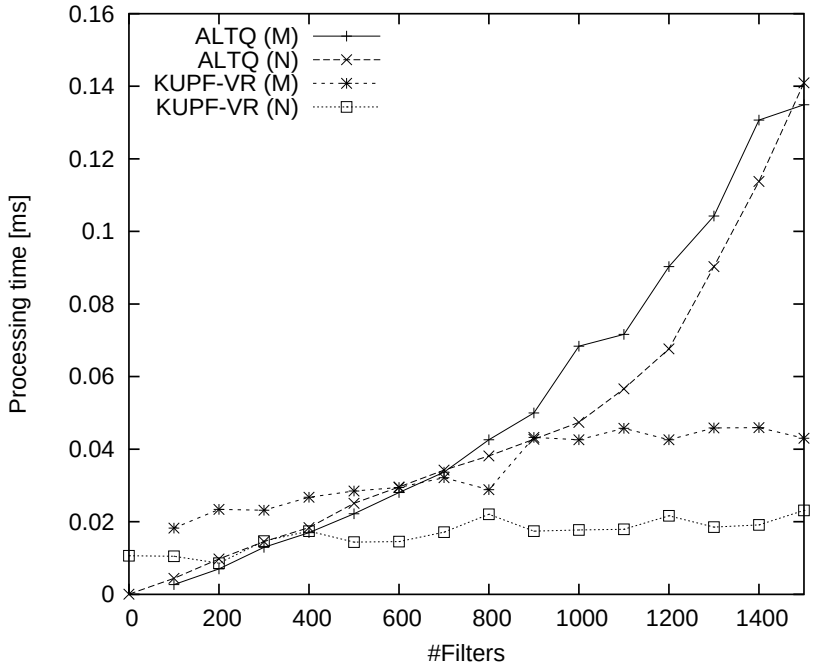

図 6.19: フィルタ数に対する処理時間変動 (ALTQ, KUPF-VR: IPv6 net)

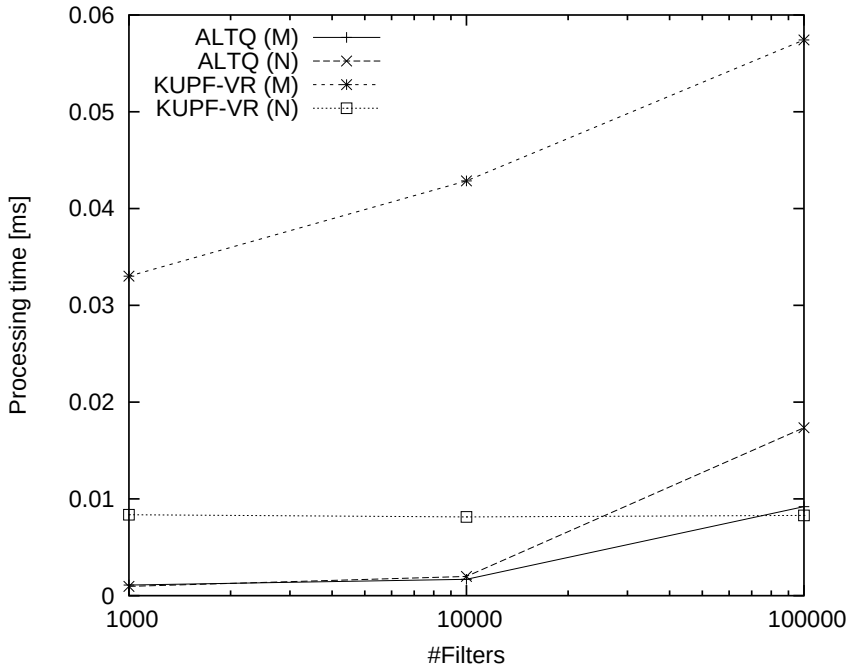

図 6.20: IPv4 ホストフィルタ数に対する処理時間変動（ALTQ, KUPF-VR）

表 6.7: メモリ使用量

フィルタ数	KUPF	KUPF-VR
1	140 バイト	304 バイト
100,000	14 M バイト	200 M バイト

1 ノード当たり 164 バイトを KUPF より多く使用する．100,000 フィルタを登録する場合，探索木の高さ毎のノード数を 100,000 ノード，3,000 ノード，100 ノード，1 ノードとして，KUPF-VR は約 200M バイトを使用する．今日のインターネット上の基幹ルータは百 M バイト程度から数百 M バイトのメモリを実装している．そのため，KUPF-VR を用いてインターネット上ので維持されている全経路数とほぼ同数の 100,000 フィルタを利用することも可能だと考えられる．

6.4.4. 考　　察

KUPF は，2 段階選択モデルを導入することで，サービス非依存の第 1 ステージとサービス依存の第 2 ステージにパラメータフィルタの処理を分離する．第 1 ステージでは，サービス毎のポリシに依存する処理は必要ないが，整数値一致，ビットマスク一致，ビット列プレフィックス一致等，様々なパラメータの比較演算処理を行う．これら個々のパラメータの探索においては，ハッシュ表，二分木，区分木，基数探索などの高速化高速化手法があるが，複数のパラメータを組み合わせた探索には適用できない．この問題を解決するために，KUPF-VR ではパラメータをすべて正規化し，パラメータを仮想的な多次元空間上の区間で表現する．これに多次元空間上の検索手法を適用して，第 1 ステージの高速化を目指した．

探索木は正規化されたパラメータを基に構築されているため，パラメータの種類に関係なく，同じ実装で高速に検索することが可能である．新しい種類のパラメータの KUPF-VR への導入は，パラメータを正規化するモジュールの実装だけで可能である．また，パラメータフィルタは汎用性が高いため，ハードウェア実装を実現できた場合には多様なサービスに適用可能である．KUPF-VR による改善は，複数のサービスポリシに対応可能な一般化した通信データ分類機構の実現を目指す KUPF の目標に適していると考えられる．

もう一つの方向として，特殊化による高速化がある．ALTQ は，特定のパラメータを基にしたハッシュ表で高速化を行っている．また，汎用化のためのオーバヘッドがない．そのため，フィルタがそのパラメータのハッシュ値で適切に分散する場合は，非常に高速に検索が行える．逆に，ハッシュ値が分散しない場合は逐次探索となり，処理が遅くなる．また，新しい種類のパラメータをフィルタに

導入した場合，検索処理の実装を変更する必要があり，そのパラメータに適した効率化をしない限り高速化を望めない．

第7章 ステージ間連携

第6章のKUPF-VRは，サービス非依存の第1ステージに多次元空間検索手法を導入した．これにより，KUPFの汎用性を損なうことなく，検索速度高速化を実現した．

本章では，第1ステージ処理中に第2ステージを先読み的に実行することで，第1ステージと第2ステージの連携により検索処理を高速化した．提案手法では，最善一致検索に着目し，KUPF-VRの探索木の途中ノードに第2ステージ処理で必要とする情報を保持し，この情報を用いて第2ステージの処理を第1ステージ処理中に実行する．第2ステージにより選択される分類レコードを含まない部分木の探索を省略することで，第1ステージの処理を最適化する．本手法の有効性を示すために，パラメータフィルタを実装し，評価実験を行った．

7.1. ポリシの先読み実行

第1ステージ処理中に第2ステージの呼び出しを先読み的に実行することで，ステージ間連携による高速化の余地がある．例えば，最善一致検索において，ある分類レコードが入力データの条件を満たす場合，この分類レコードより一致度が低い分類レコード検索を省略することよる最適化が可能である．

IPアドレスを最長一致検索する場合において，表7.1の分類レコードを考える．この場合のKUPF-VR探索木が図7.1とする．入力データのIPアドレスが10.1.2.3の場合，(1)では10.1.2.0/23, 10.0.0.0/14がともに10.1.2.3に適合する．(2)では10.1.2.0/24，(3)では10.1.0.0/16が適合し，

表 7.1: 最長一致検索分類レコード

分類レコード	最長一致	アクション
R_1	10.1.0.0/16	A_1
R_2	10.1.2.0/24	A_2
R_3	10.1.3.0/24	A_3
R_4	10.2.0.0/15	A_4

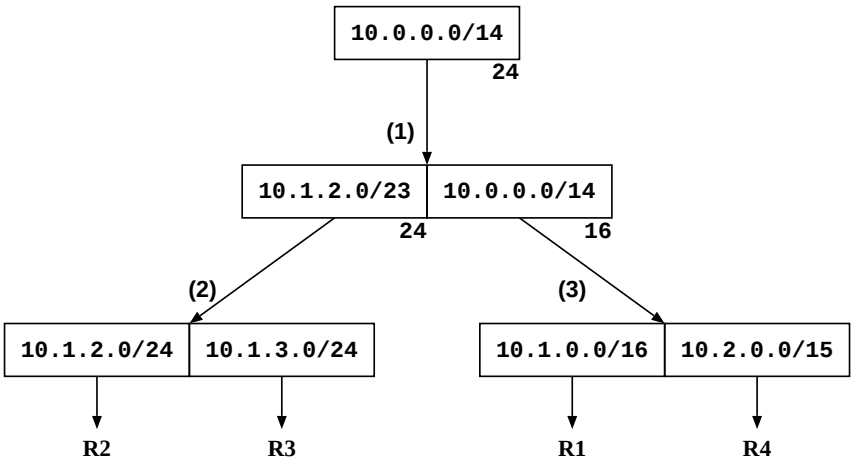

図 7.1: 最長一致検索探索木

第 1 ステージでは R_2, R_1 が抽出される．第 2 ステージでは，最長一致則によりネットマスク長の長い R_2 が選択される．

第 2 ステージの最長一致則で用いるネットマスク長に着目する．中間ノードのエントリ 10.1.2.0/23 の下位ノードのネットマスク長はともに 24，エントリ 10.0.0.0/13 の下位ノードのネットマスク長は最長が 15 である．この下位ノードの最長ネットマスク長を各エントリに保持する．(2) の探索では 10.1.2.0/24 が得られる．次に (3) の探索を行う前に，下位ノードの最長ネットマスク長 15 と (2) で得られたネットマスク長 24 を比較すると，下位ノードのネットマスク長が短い．(3) の探索により得られる結果は，第 2 ステージの最長一致則では選択されないと判断できる．したがって，(3) の探索は必要ない．

最善一致検索において，第 2 ステージの判断基準となる値を分類レコードから抜き出し，下位ノード群から部分木の最善値を選択できれば，部分木の最善値を探索木の各エントリに保持することで，第 1 ステージ処理中に第 2 ステージを先読み的に実行できる．これにより，各ノードにおける探索処理は増えるが，探索ノードを省略することが可能となる．

7.2. 実　　装

最善一致検索に特化したステージ間連携を行う，最善一致 KUPF-VR を NetBSD 1.6 上で C 言語を用いて実装した．第 1 ステージ呼び出し時に第 2 ステージ呼び出しのパラメータが必要があるため，分類操作のプログラムインタフェイスは KUPF-VR と異なる．

最善一致のためのノードエントリの構造を図 7.2 に示す．KUPF-VR 探索木ノードエントリ（図

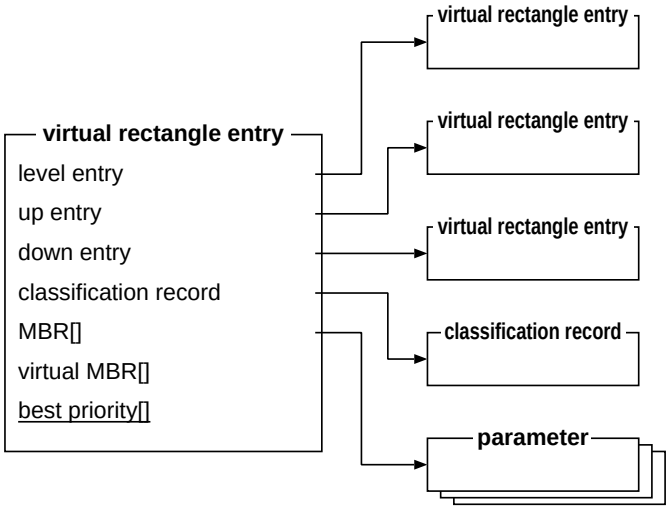

図 7.2: 最善一致 KUPF-VR 探索木ノードエントリ

表 7.2: プログラムインタフェイス

分類操作	
最善一致分類操作	`kupf_comp_table_collect_best`(分類表，キーパラメータ群，優先順位)

6.9）と比べ，下位ノードの比較演算毎の最善優先度を保持する配列（下線部）が増えている．データ分類時にパラメータ間の優先度を指定可能とするため，最善優先度はをパラメータ毎に保持する．この値は，探索木にデータが挿入・削除されるたびに算出する．葉ノードの場合，分類レコードのパラメータ毎に比較優先度関数（表 4.6）で算出した値を保持する．葉ノード以外の場合，下位ノードからパラメータ毎に算出した最大値を保持する．これにより，各ノードは部分木の最善値をパラメータ毎に保持する．

第 1 ステージで第 2 ステージの処理を実行するためには，第 1 ステージ呼び出し時に第 2 ステージ呼び出しのパラメータが必要がある．最善一致検索ではパラメータ間の優先順位が必要であるため，キーパラメータ群とともに優先順位を与える（表 7.2）．返値として，最善一致の分類レコードを返す．

探索木からの検索は以下の手順で実行する．

1. ノードリスト $\mathbf{N}$ を根ノード，分類レコード R を空とする
2. ノードリスト $\mathbf{N}$ から 1 ノード取り出し，ノード N とする．ノードがなければ，分類レコード R を結果として終了する
3. 分類レコード R が空でなければ，R とノード N の優先度を比較し，R の優先度が高ければ 2 に

表 7.3: KUPF-VR 実験環境

CPU	Pentium 4 (2 GHz)
メモリ	512 MBytes
OS	NetBSD 1.6.2_RC1

戻る

4. キーパラメータからノード N を基準とした VBR を算出する

5. ノード N が葉ノードでなければ，キーパラメータとノード N の各エントリを VBR 同士，キーパラメータとレコードパラメータで比較し，適合したエントリの下位ノードをノードリスト $\mathbf{N}$ に加える

6. ノード N が葉ノードであれば，キーパラメータとノード N の各エントリを VBR 同士，キーパラメータとレコードパラメータで比較し，適合したエントリの分類レコードと分類レコード R の内，もっとも優先度が高い分類レコードを分類レコード R とする

7. 2 に返る

7.3. 評　　価

本提案による速度性能向上の効果を検証するために，最善一致検索に特化した KUPF-VR を実装し，KUPF-VR との比較実験を行った．

7.3.1. 測 定 環 境

性能測定環境を表 7.3 に示す．ユーザ空間で測定プログラムを実行した．KUPF-VR における第 1 ステージ，第 1 ステージと第 2 ステージ，最善一致に特化した KUPF-VR における第 1 ステージと第 2 ステージの処理時間を，`getrusage()` システムコールを用いて測定した．測定結果として，1,000 回実行の平均値 12 サンプル中の中間値 10 サンプルの平均値を用いた．1 ノードあたりの子ノード数の上限値は 3，下限値は 8 とした．仮想区間の上限値，下限値の精度は 16 ビット長整数とした．

表 7.4: 実験に用いたフィルタ

入力インタフェイス	プロトコル	終点アドレス	終点ポート
1	UDP	fec0::/128 – fec0::/29	80

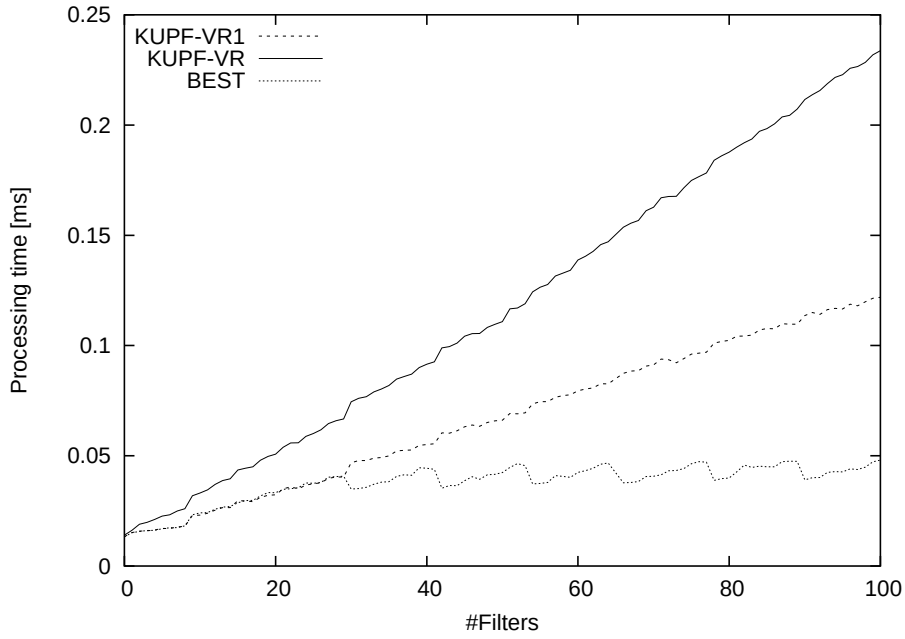

図 7.3: フィルタ数に対する処理時間変動

7.3.2. 測 定 結 果

KUPF-VR との比較を行うため，表 7.4 の終点アドレスのプレフィックス長が異なる 0 から 100 個のフィルタを用いて，処理時間を測定した．キーパラメータ群は，全フィルタに適合するパラメータ群を与えた．測定結果を図 7.3 に示す．グラフの KUPF-VR1，KUPF-VR はそれぞれ，KUPF-VR の第 1 ステージ，第 1 ステージと第 2 ステージの処理時間を示す．BEST は最善一致に特化した KUPF-VR の処理時間を示す．KUPF-VR はフィルタ数にほぼ比例した結果が得られた．フィルタ数の増加につれ探索ノード数が増え，適合レコードも増加したためと考えられる．これに対して，最善一致に特化した KUPF-VR は上下の変動があるものの，フィルタ数に関係なくほぼ一定の結果が得られた．全般に KUPF-VR に比べ優れた性能を示している．探索ノードの省略により，処理時間の増加が抑制されたためと考えられる．

7.3.3. 考　　察

本提案では，第 1 ステージと第 2 ステージ連携による高速化を目指した．第 2 ステージはサービス依存処理となるため，汎用的なステージ連携は困難である．この問題を回避するため，最善一致を行う第 2 ステージに対象を限定した．各エントリは比較演算毎に算出した最善優先度を保持するため，比較順序の変化には対応できるが，最善一致検索以外には適用できない．

問題を解決する方法として，部分木の最善優先度の値を算出する代わりに，部分木毎に第 2 ステージの処理を演算子として取り出すことが考えられる．これが可能であれば，最善一致検索以外にもステージ連携が適用可能となる．

第8章 研究の総括

本研究では，汎用的なパラメータフィルタを構築するために，データ分類について検討した．そして，2段階選択モデルに基づいたパラメータフィルタ・フレームワークの設計と実装を行い，実証実験を行い，さらに速度性能向上のための提案を行った．本章では，本研究で得られた知見と今後の課題を述べ，研究を総括する．

8.1. 本研究によって得られた知見

第5章の適用実験において，従来サービスに対してKUPFフレームワークが適用可能であることを示した．パケットフィルタへの適用では，単一サービス環境下においてKUPFはデータ分類機構を置換可能であることを示した．分類スキーマを定義することにより，KUPFフレームワークはサービス内のデータ分類機構と互換性を維持した入力データに対する処理の選択を実現した．IP入力フィルタへの適用では，サービスから独立したKUPFによりサービスを制御可能であることを示した．初めにKUPFが入力データに対する処理を決定し，その処理に従ってサービスの処理を呼び出した．したがって，サービスを処理する実装を変更せずに，入力データを分類するパラメータを変更できる．また，実験は単一サービス環境で行ったが，複数サービスの処理に対する分類レコードを分類表に登録し，KUPFが決定する処理が示すサービスを呼び出すことで，複数サービス環境下においても同様にKUPFによりサービスを制御可能である．

従来システムにおいて終点アドレスを含めた複数パラメータに基づくIP転送を行う場合，終点アドレスのみに基づく経路は呼び出される通常の経路検索機構の経路表として，その他のパラメータに基づく経路はファイアウォール等のアクセスリストとして経路を管理していた．そのために，IP転送処理がIP転送機構とファイアウォール機構に分離していた．KUPFを用いることで，IP転送機構を使用して複数のパラメータに基づく経路に従ったIP転送が実現できる．

2段階選択モデルに基づいたパラメータフィルタは，汎用性を得た代わりに検索処理速度が犠牲となった．この問題を解決するために，第6章ではデータベース分野において多次元空間探索として代表的なR-treeの派生手法である，R*-treeを適用した．探索木作成アルゴリズムの変更，仮想包囲矩形の導入により実現したKUPF-VRは，KUPFの汎用性を維持したまま高速化を実現した．

従来システムにおいては，データ分類機構はサービスごとに実装されていた．そのため，処理性能向

上は実装ごとの改善が必要であった．パラメータフィルタを汎用的に使用する場合，パラメータフィルタの実装を改善することでサービス全体の性能向上が可能である．

KUPF フレームワークの実装においては，汎用性・モジュール化を目指した設計・実装を行った．これにより，パラメータ型，比較型，分類スキーマの拡張性が得られた．また，モジュール間の独立性により，KUPF-VR の高速化実装を容易にフレームワークに結合することができた．フレームワーク API の変更も分類表作成操作のみであった．

8.2. 今後の課題

8.2.1. ポリシ競合の検出

従来のデータ分類機構では，サービスにおける入力データに対する処理の選択はサービス内のデータ分類機構で行う．多様なサービスをネットワーク上で展開する場合，当然サービス毎のポリシが競合する場合がある．競合は，単一ノード内で発生する場合だけでなく，ネットワーク上の複数ノードが関係することで発生する場合がある．この場合，入力データに対する処理の選択は複数サービス，複数ノードに分散するため，ポリシの競合の発見は困難を伴う．複雑化したネットワークの安定運用には，ネットワーク上の複数ノードを考慮したポリシ競合の検出が必要である．

パラメータフィルタモデルにおいては，サービス内の処理の選択をパラメータフィルタで扱うため，サービスごとの処理はすべてパラメータフィルタ内の分類レコードで表される．そのため，ポリシ競合はパラメータフィルタ内の分類レコード間で発生する．2 段間選択モデルを用いる KUPF 上では，第 1 ステージはポリシ非依存処理，第 2 ステージはポリシ依存処理を行う．したがって，ポリシ競合の要因は第 2 ステージに限定される．

以上のことから，分類レコードが保持するアクションの排他関係を第 2 ステージに与えることで，ポリシ競合の検出が可能であると考える．

8.2.2. ポリシの記述

本研究の提案する 2 段階選択モデルによるパラメータフィルタモデルは，サービス非依存の第 1 ステージについては具体的実装および性能改善の提案が行えた．一方，サービス依存の第 2 ステージについては 1 例として最善一致を示すことに限定している．第 2 ステージは，適合レコード間の関係をサービス依存のポリシにより定義する．パラメータフィルタの動作を記述するためには，このポリシを記述することが必要である．

第 2 ステージについても適用例を積み重ねることで，何らかの系統化が可能であると考えられる．

ポリシは，単純な適合レコード間の優先度であったり，適合レコードが保持するアクションの排他関係であったりする．これらを系統化することで，条件によっては第 2 ステージのモデルが構築できる可能性がある．

8.2.3. 処理速度の向上

KUPF フレームワークはパラメータフィルタを抽象化したモデルで実装しているため，汎用性が高い反面，処理速度性能が犠牲となっている．処理速度性能を向上させる手法としては，KUPF 各ステージ毎の高速化，ステージ間連携による高速化，ハードウェア処理導入による高速化がある．

第 1 ステージ高速化は，第 6 章において多次元空間検索手法を用いることで検索速度高速化を実現した．第 2 ステージはサービス毎に個別実装となるため，KUPF フレームワークとして第 2 ステージ単体での高速化はできない．

これに対し，第 1 ステージ処理中に第 2 ステージの呼び出しを先読み的に実行することで，ステージ間連携による高速化の余地がある．最善一致検索を対象として，第 7 章において KUPF-VR の探索木の途中ノードに第 2 ステージ処理で必要とする情報を保持し，この情報を用いた第 1 ステージ最適化により検索速度高速化を実現した．最善一致検索以外においても，同様の手法を用いて第 1 ステージ最適化が可能だと考えられる．パラメータフィルタ外部の最適化としては，複数のサービス処理機構による個別の分類機構をパラメータフィルタとして KUPF に集約することで，重複する処理の効率化が考えられる．これにより，従来ソフトウェアによりデータ分類機構が実装されていた小規模ネットワーク機器に対し，KUPF フレームワークを利用した従来より高機能・高速なパラメータフィルタを提供可能である．

KUPF-VR は仮想包囲矩形導入によりパラメータ比較を単純化している．そのため，ハードウェア処理導入はパラメータの柔軟性が高いにも関わらず比較的容易だと考えられる．今後，これらの手法を組み合わせることで，KUPF の汎用性を維持しつつ処理速度の向上が達成できると考える．トラフィックが集中する基幹ルータに対しては，KUPF モデルに基づき KUPF フレームワークのインタフェイスを備える，ハードウェア処理によるパラメータフィルタを開発することで，異なる実装が混在するネットワークにおいても単一の制御機構により制御可能となる．

従来，ルーティングスケーラビリティは経路表と経路制御プログラムの処理能力で述べられてきた．経路表に保持可能な経路数と，IP 転送時の終点アドレスからの経路表検索の所要時間が大きな要因であった．しかし，今日では始点アドレス偽装確認のための始点アドレスによる経路表検索も行われ，経路表の処理項目が増えている．さらにこれより複雑なポリシールーティング，パケットフィルタリング等を行う場合，複数のパラメータを扱う必要があるために経路表だけでは機能が足りず，アクセスリスト等のパケット分類機構が必要である．これらの処理を基幹ルータのルーティングスケーラビ

リティと同程度で処理しようとすると，ソフトウェア処理によるアクセスリストでは処理できず，高価なハードウェアが必要となる．このような分野に対応するためには，KUPF においてもルーティングスケーラビリティを考慮した実装の検討が必要である．

8.3. 結　　論

ADSL の低価格化によるインターネットの爆発的普及により，研究機関に属する一部の人々だけが利用していたインターネットは，誰もが利用する社会基盤としての役割を果たすまでに発展した．従来の手紙，電話，新聞，テレビなどの多くのコミュニケーション手段が，今日ではインターネット上で商用サービスとして提供され，他にもさらに多くのサービスが展開されようとしている．IP プロトコルを使用してインターネット上を流れるデータは多様化し，インターネットに求められる要求は，当初の予想を越え，ますます高度化・複雑化している．本研究では，高機能サービスを支える複数のサービスポリシに対応可能な一般化した通信データ分類機構の構築を目標とし，これを実現する技術の設計と開発を行い，実証実験を行った．

ネットワークサービスが通信データに必要な処理を実行するためには，サービス処理機構が通信データを的確に分類し，処理内容を決定することが不可欠である．様々なサービスが並行して提供されているネットワーク内では，各サービス毎のポリシによる矛盾を明らかにし，通信データを正しく分類しなければならない．本研究は，パラメータフィルタの概念の明確化から始めた．サービス毎に個別に動作していた分類機構をサービスから分離し，サービス毎に処理内容別に通信データを分類するモデルから，通信データ毎に複数サービスから処理内容を選択するパラメータフィルタモデルへと転換した．これにより，複数サービスを考慮して通信データに適用する処理内容を決定可能となる．

複数サービスを並行して運用する場合，ポリシの競合問題は避けられない．本研究では，パラメータフィルタをサービス非依存の第 1 ステージとサービス依存の第 2 ステージに分離した 2 段階選択モデル，KUPF により，パラメータフィルタの一般化と汎用性の実現を試みた．サービス非依存部分はデータ比較方法を定義する分類スキーマにより，処理を決定する．ポリシの競合解決は，サービス依存部分に完結させることで，処理を明確化する．

2 段階選択モデルを検証するために，実利用者が使用するネットワークでこのモデルに基づいたフレームワークを適用した QoS 保証サービスを運用し，実証実験を行った．この結果，本研究の提案モデルが実運用にも耐えうることが示せた．パラメータフィルタの処理速度計測実験では，サービスに特化した分類機構に比べ，処理速度性能が劣ることが明らかになった．

さらに，多次元空間上での効率的な検索手法として知られる R*-tree を応用した，仮想包囲矩形に基づく探索木を第 1 ステージに使用した KUPF-VR により，処理速度向上を試みた．この結果，フ

レームワークの汎用性を維持しながら処理速度の大幅な向上を達成し，分類フィルタ数が多い場合にサービスに特化した実装を上回る性能を示した．また，最善一致検索におけるステージ間連携により，さらに高速化を実現した．

以上のように，本研究では複数のサービスポリシに対応可能な一般化した通信データ分類機構の構築を目標とし，パラメータフィルタのモデル構築とフレームワーク開発を行い，実験・運用によりその可能性を示した．しかし，今後のさらなるインターネットへの高度化する要求を満たすためには，解決するべき課題が残っている．今後も研究を継続し，多様化するサービスを支えることができるインターネットを実現するシステムを確立していく必要がある．

謝　　辞

本研究を進めるにあたり，多くの皆様からご協力と助言をいただきました．ここに感謝の意を表します．

WIDE プロジェクト研究会で行った実証実験は，研究会参加者のご理解とご協力により実現できました．WIDE プロジェクトの皆様に心より感謝いたします．この実験の転送層はあやめプロジェクトのご協力をいただきました．制御層はくまプロジェクトのご協力をいただきました．実装・実験をともに行い，数々のアイデアをくださったくまプロジェクト，あやめプロジェクトの皆様に深く感謝いたします．

本研究を進めるにあたり，奈良先端科学技術大学院大学附属図書館研究開発室助手の森島直人先生には，数々の貴重なアドバイスをいただきました．深く感謝いたします．さまざまな助言をいただいた情報科学センター助手の中村豊先生に心より感謝いたします．研究活動を支えていただいた情報科学センターの皆様に感謝いたします．

本研究のご指導をいただいた，奈良先端科学技術大学院大学情報科学センターの藤川和利助教授，インターネット工学講座の山口英教授に心より感謝いたします．

本研究を進めるにあたり，貴重なご指導をいただいた奈良先端科学技術大学院大学情報科学センターの砂原秀樹教授に深く感謝いたします．

参 考 文 献

[1] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An architecture for differentiated service. *RFC 2475*, December 1998.

[2] Eric C. Rosen, Arun Viswanathan, and Ross Callon. Multiprotocol label switching architecture. *RFC 3031*, January 2001.

[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. the 1990 ACM SIGMOD international conference on Management of data*, pp. 322–331, Atlantic City, United States, 1990.

[4] Keith Sklower. A tree-based packet routing table for berkeley UNIX. In *Proc. the Winter '91 USENIX Conference*, pp. 93–99, Dallas, United States, January 1991.

[5] Gary Scott Malkin. RIP version 2. *RFC 2453*, November 1998.

[6] John Moy. Ospf version 2. *RFC 2328*, April 1998.

[7] Vince Fuller, Tony Li, Jessica Yu, and Kannan Varadhan. Classless inter-domain routing (CIDR): an address assignment and aggregation strategy. *RFC 1519*, September 1993.

[8] Philip Almquist and Frank J. Kastenholz. Towards requirements for IP routers. *RFC 1716*, November 1994.

[9] Kenjiro Cho. A framework for alternate queueing: Towards traffic management by PC-UNIX based routers. In *Proc. USENIX 1998 Annual Technical Conference*, pp. 247–258, New Orleans, United States, June 1998.

[10] Yojiro Uo, Satoshi Uda, Nobuo Ogashiwa, Satoshi Ohta, and Yoichi Shinoda. AYAME: A design and implementation of the CoS-capable MPLS layer for BSD network stack. In *Proc. INET 2000*, Yokohama, Japan, July 2000.

[11] Naoto Morishima, Akimichi Ogawa, Hiroshi Esaki, Osamu Nakamura, Suguru Yamaguchi, and Jun Murai. Preliminary field-trial for QoS routing and dynamic SLA. *IEICE: Special Issue on Internet Technology*, Vol. E84-B, No. 8, pp. 2039–2047, August 2001.

[12] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. the 1984 ACM SIGMOD international conference on Management of data*, pp. 47–57, Boston, United States, 1984.

[13] 櫻井保志, 吉川正俊, 植村俊亮. 仮想包囲矩形に基づく多次元データ集合に対する近傍探索. 情処学論, Vol. 40, No. SIG03, pp. 68–79, February 1999.

[14] 堀之口浩征, 黒木進, 牧之内顕文. 時空間データベースインデックス正規化 R*-tree の実装と性能テスト. 情処学論, Vol. 40, No. 3, pp. 1225–1235, March 1999.

[15] CIDR report. `http://www.cidr-report.org/`.

付録 A

研 究 業 績

A.1. 本研究に関する業績

A.1.1. 学 術 論 文 誌

1. 垣内 正年, 森島 直人, 砂原 秀樹. “仮想包囲矩形に基づくパラメータフィルタの設計と実装,” 電子情報通信学会和文論文誌, D-I. (2004 年 5 月掲載予定)

A.1.2. 国際会議（査読あり）

1. Masatoshi Kakiuchi, Noto Morishima, Yutaka Nakamura, Kazutoshi Fujikawa, Hideki Sunahara. “KUPF: 2-Phase Selection Model of Classification Records,” In *2003 Symposium on Applications and the Internet Workshops,* Orlando, Florida, United States, January 27–31, 2003.

A.1.3. 国内研究会等（査読なし）

1. 垣内 正年, 森島 直人, 宇田 仁, 中村 豊, 砂原 秀樹. “汎用的なパラメータフィルタの設計と実装,” 電子情報通信学会技術研究報告, vol.102, no.143, pp.7–14, June 2002.

2. 垣内 正年, 森島 直人, 砂原 秀樹. “KUPF フレームワークの実装と応用,” 情報処理学会研究報告, vol.2003, no.118, pp.61–66 November 2003.

3. 垣内 正年, 森島 直人, 砂原 秀樹. “ポリシ先読みによるパラメータフィルタ高速化の試み,” 情報処理学会研究報告, vol.2004, no.9, pp.89–94 January, 2004.

A.1.4. そ の 他

1. 森島 直人, 小川 晃通, 垣内 正年. “高機能転送層を持つ広域ネットワークの包括的制御システムの開発,” 情報処理振興事業協会平成 13 年度成果報告集,
<http://www.ipa.go.jp/SPC/report/01fy-pro/explorat/hieffic/hieffic.pdf>.

A.2. その他の業績

A.2.1. 国内研究会等（査読なし）

1. 垣内 正年, 馬場 基晴, 原田 章, 中西 通雄. “TeX 操作および辞書検索のための GUI 構築,” ’97 PC カンファレンス予稿集, pp.89–92, August 1997.

2. 米山 清二郎, 垣内 正年, 砂原 秀樹. “Reliable Multicast — Using FEC on IPv6,” インターネットコンファレンス’99 論文集, p.139, December 1999.