Linux IPv6 Stack Implementation Based on Serialized Data State Processing

吉藤英明
東京大学
hideaki@yoshifuji.org

宮澤和紀
横河電機 (株)
Kazunori.Miyazawa@jp.yokogawa.com

中村雅英
(株) 日立コミュニケーションテクノロジー
masahide_nakamura@hitachi-com.co.jp

関谷勇司
東京大学
sekiya@wide.ad.jp

江崎浩
東京大学
hiroshi@wide.ad.jp

村井純
慶應義塾大学
jun@wide.ad.jp

2003 年 10 月 28 日

概 要

IPv6 は現在の IPv4 インターネット環境を後継する次世代インターネット基盤技術である。著名な OS の一つである Linux は 1996 年より IPv6 に対応しているが、その品質は実運用するには不十分なものであった。

本稿では、経路表管理、XFRM 機構におけるパケット処理における、我々の IPv6 実装の設計について紹介する。本実装は単純なオブジェクト管理と安定、柔軟かつ拡張性のある IPv6 スタックの実現を図る、Serialized Data State Processing と名付けられた設計に基づいている。

TAHI プロトコル適合性試験の結果より、我々の実装が十分な品質を持っていることを示す。

1 はじめに

インターネットは 1960 年代の終わり頃から IPv4 と呼ばれるプロトコルで動作してきた。

インターネットの急激な成長に追いつき、パケット転送性能、プロトコル拡張性、セキュリティ及びプライバシといった、伝統的な IPv4 の種々の欠点を解決するための次世代 IP(IPng) に関する全面的な技術的議論は IETF で 1992 年から開始された。その議論の結果、IPv6 の基本規格は 1994 年に制定され、実験的実装や 6bone[1] などによるネットワーク運用を経て IPv6 技術は実用段階に突入してきている。インターネットサービス事業者 (ISP) による商用接続サービスや、IPv6 で動作する種々のアプリケーションも出現してきている。これは、IPv6 を実装する全ての機器が高い品質を持っていることが求められていることを意味する。

Linux も他の OS 同様 IPv6 技術に対応しており、1996 年のバージョン 2.1.x から利用できる。しかしながら、IPv6 スタックが統合された後は積極的に開発・保守されてこなかった。

従来の Linux IPv6 実装についての検討の結果、その設計は複雑で適切に整理されていないことが判明した。本稿では、Serialized Data State Processing 手法に基づく実装について説明する。本設計は、連続したオブジェクトおよび状態処理構造の導入によって、単純で拡張可能となっている。本稿で説明する USAGI プロジェクト [12] の IPv6 スタックに統合されている。

本稿では、我々の行った IPv6 スタックの実装、設計および仕様適合性評価の結果について説明する。2 章では背景、関連研究および設計思想について概観する。3 章では経路表管理について説明し、4 章では XFRM コンセプトに基づく IP パケット処理について説明する。5 章は簡単なまとめである。

2 Serialized Data State Processing

2.1 背景

大規模システムにおける状態管理が簡単でないことはよく知られている。一般的には、システムを「モジュール」と呼ばれるいくつかの部分に分割し、モジュール内の値への直接操作を禁止することがよく行われる。特に、自律分散オブジェクトとメッセージ交換を用いてこの目的を達成する手法はオブジェクト指向 (OOP) と呼ばれる。他のモジュールの状態に依存せずに自動的に実行されるような、複数モジュールに分割することは効率的なマルチプロセッシングにも重要である。

Linux も一種のオブジェクト指向である、モジュール化の概念が適用されている。たとえば、オブジェクトの生存期間を管理するための参照カウンタの実装は、オブジェクト指向の思想と同様である。しかしながら、オブジェクト間の状態依存性はよく整理されていない。そのため、依存性と状態遷移の衝突を回避するため、より複雑な排他制御や、やや例外的な処理が必要とされている。また、新たな機能を追加することも難しい。それは、オブジェクト間の複雑な状態依存のある既存のシステムに新しいオブジェクトや状態を導入する場合、正確な状態管理と実装が困難であることに起因する。

本稿では、Linux IPv6 スタックにおける状態遷移を再

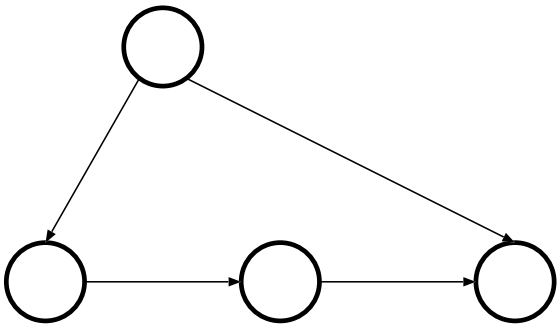

図 1: Data Object Dependency Graph

構成する。従来の Linux IPv6 スタックでは図 1 のように、ループの存在する多くの状態遷移系が存在する。状態遷移系にループがなければ状態管理および実装はループがある場合に比べてはるかに容易になる。我々はこの設計原理を “Serialized Data State Processing” と命名する。この手法により、安定、柔軟かつ拡張性のある実装を実現することができる。

2.2 Serialized Data State Processing

大きな処理タスクは複数の小さな処理タスクとオブジェクトに分割される。これらのタスクとオブジェクトは、状態遷移系にループを含まないよう、基本的に可能な限り直線的に接続される。この Serialized Data State Processing 設計の基本構造は自律したデータオブジェクトであり、それは以下の要素からなる：

排他制御
: オブジェクトへのアクセスの順序化には、ロックやセマフォと呼ばれる機構を用いて実装される。一般に、データアクセスの順序化の観点からは、そのデータを含む構造がそれについての排他制御機構を備える場合には、データ内部の排他制御機構は不要である。しかしながら、可能な限り独立したオブジェクトを実現するため、データ内部にロック機構を導入する。この設計により、複数のデータ構造が同一のデータを共有することができ、マルチプロセッシング環境における性能向上を図ることができる。

参照カウンタ
: 参照カウンタは、オブジェクトの被参照回数をもとにして生存期間を管理する。オブジェクトは、被参照回数が 0 になると自分で消滅することができる。しかしながら、複雑な状態遷移においては、もはや参照されていないオブジェクトを廃棄するために、外部的なガーベージコレクション (GC) が必要となる場合がある。参照回数管理を各オブジェクト毎およびそれを含む構造において行えば、これを分離しない場合よりもオブジェクトを容易かつ確実に管理することができる。

これらの要素に加え、オブジェクトの管理タイマのために内部タイマを必要とすることがある。

Serialized Data State Processing は、1 つあるいは複数のデータオブジェクトが有向グラフによって結合されており、それらが全体として働くような、上記のような自律

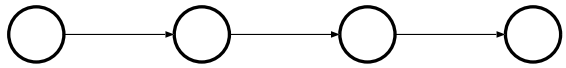

図 2: Serialized Data State

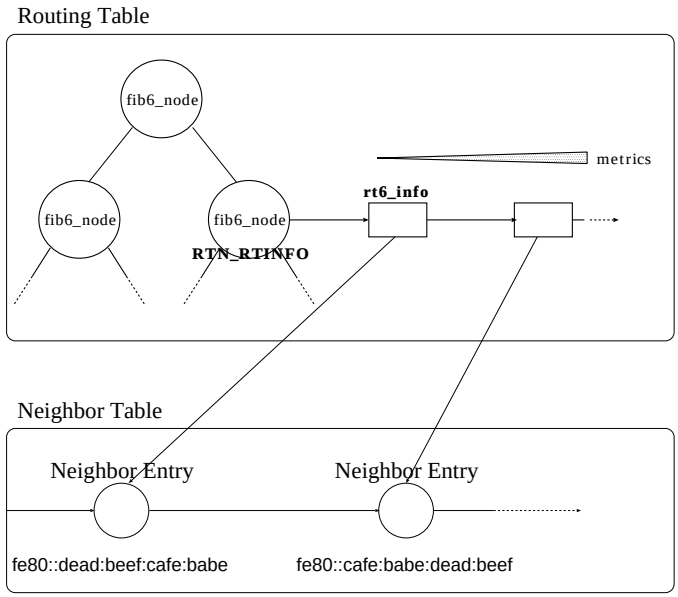

図 3: Linux Routing Table and Neighbor Table

データオブジェクトの一種の組み合わせである。そのオブジェクトとその結合形態を適切に定義することにより、オブジェクトの独立性を保ち、不要な（外部）参照を排除することができる。特に、図 2 に示されるように線形的な結合形態を定義できれば、デッドロックを引き起こす資源依存性の衝突を避けることができる。

3 経路表における状態管理

ルーチングとは、IP パケットを宛先ノードに届けるプロセスである。パケットを次ホップルータに転送したり宛先に伝達するのに必要な全ての情報は経路表が保持している。

Linux の IPv6 経路表は “Forwarding Information Base (FIB)” として知られ、Radix Tree[11] で構成される。図 3 は Linux における経路表と近隣探索管理を示す。

FIB において、各ノードはテストするビット位置を示している。また、`RTN_RTINFO` フラグの立った各ノードには、次ホップやメトリックといった実際の経路情報を表現する “葉” のリンクリストがあり、メトリック順に並んでいる。葉にある次ホップ情報は、実際には対応する NCE (Neighbor Cache Entry; 近隣の到達性を管理する) を参照している。伝統的な BSD 派生物と異なり、Linux は単一宛先に対して複数の経路をサポートしている。

本章では、Serialized Data State Processing 手法がいかに Linux システムにおける経路表管理に適用されるかについて説明する。経路表管理には、(1) デフォルトルータ管理 及び (2) ルータプリファレンス管理 が含まれる。

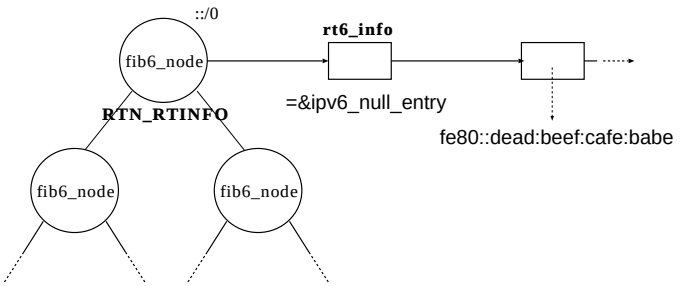

図 4: Linux IPv6 Routing Table Structure

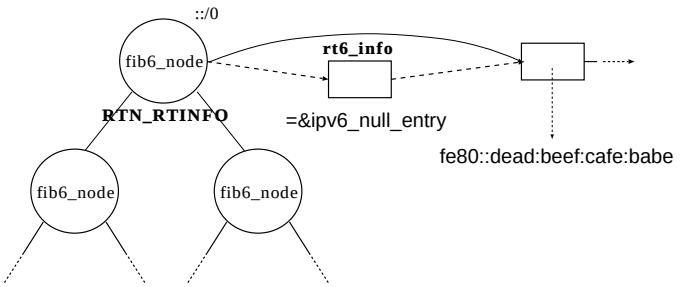

図 5: USAGI IPv6 Routing Table Structure

3.1 デフォルトルータ管理

ND は “Default Router List” と呼ばれる新しいデータ構造を導入している。それは近隣ルータからのルータ広告に含まれる “デフォルトルータ” の情報を含んでいる。

KAME [5] はデフォルトルータの情報を個別の “Default Router List” により管理している。一方、従来の Linux 実装においては、Default Router List に相当するものは経路表木のトップレベルの根に、あたかも一種の ::/0 への経路 (“デフォルト経路”) であるかのように保持されている。::/0 に対する最初のエントリは常に `ipv6_null_entry` で、これは通常の経路としては決して使われない。そして、“デフォルトルータ” がそれに続く (図 4). それらは特別なデフォルトルータ選択アルゴリズムに例外的に使われる。一方、手動でデフォルト経路が追加されるときは、関係する情報は `ipv6_null_entry` を含む `rt6_info{}` 構造の次に追加される。この手法では、追加された ::/0 への経路は参照されない。これは、特別なデフォルトルータ選択は通常の ::/0 への経路には適用されないからである。

経路表の原理及び “Default Router List” の思想を考慮すると、::/0 への経路を特別扱いする必要は全くないことがわかる。図 5 に示すように、提案手法では新しい経路エントリをトップレベルの根に追加する際には `ipv6_null_entry` を新しいエントリと取り替え、逆に、もし最後の経路が削除されようとするときは、再び `ipv6_null_entry` を追加する。これは、実際に “Default Routers” に関する情報を含む “デフォルト経路” を “通常の” 経路として扱うことを意味し、経路表から “default route” エントリを、通常の経路同様、自然に追加、削除することができる。

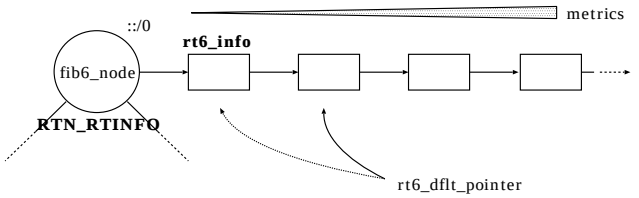

図 6: Default Routers in Linux

3.2 Router Precedence Management

ある宛先に対する次ホップルータのリストからひとつのルータを選択することを “Router Selection” といい、“Default Router Selection” はこのデフォルト経路に対する特別な場合と考えることができる。

前節で述べたように、デフォルトルータは経路表木のトップレベルの根に保持されている。Default Router Selection では、現在選ばれているデフォルトルータが不到達になった場合には、ラウンドロビンで次の利用可能なデフォルトルータを選択する。従来の Linux では、この方法を実現するため、現在選ばれているデフォルトルータは、大域ロックである `rt6_dflt_lock` で守られた、`rt6_dflt_pointer` により示されており、それにより示されたルータが到達できなくなると更新されていた (図 6)。

この実装では、以下のような問題点が存在する。

不公平性
: `rt6_dflt_pointer` は経路が変わるとリセットされる。これは比較的頻繁に起こり、全てのルータがうまく均等に選択されない原因になる。

汎用性の欠如
: `rt6_dflt_pointer` は単一かつ静的な変数である。これはデフォルト経路 (::/0) だけのために用いられ、同一の手法は一般の経路選択に適用することができない。一般の経路に対しも、複数の経路から一つを選択する適切なメカニズムを導入する必要があると考えられる。そのようなポリシあるいは戦略としては、“Default Router Preferences, More-Specific Routes, and Load Sharing” [2] がある。

メトリック
: Linux はデフォルト経路のメトリックの扱いについて、通常の経路と違う方法をとっていた。しかし、デフォルトルータを通常の経路同様に扱うことを提案しており、デフォルト経路に対するメトリック特別な扱いも廃止すべきと考えられる。

この設計の目標を達成するため、我々は新しい一般的ラウンドロビンのメカニズムを導入した。我々は、ある経路が使われると、それと同一メトリックをもつ経路との間でラウンドロビンを適用する (図 7)。つまり、経路が探されるときは、次ホップにある NCE と協調して、“probably reachable” な状態にある最初のエントリを用い、それを同一メトリックのエントリの最後にリンクし直す。これにより、`rt6_dflt_pointer` 及び `rt6_dflt_lock` を削除することができ、またこの手法はデフォルト経路だけでなく他の通常の経路に対しても適用できるものである。

“Router Selection” 仕様草案 [2] には、経路の “プリファレンス” という考えがある。ルータはルータ広告中に経路

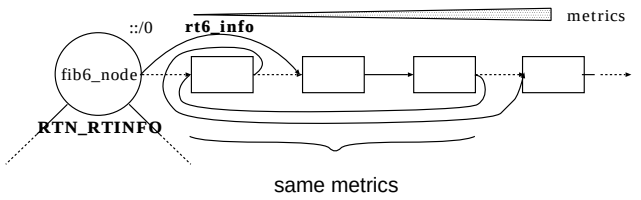

図 7: New Method for Route Round-robin

に関する 2 ビットのプリファレンスを広告し、ホストは最も好ましいルータを選択する。

この機能を実現するため、我々は (2 ビットの) 経路プリファレンスを、経路情報のフラグに導入している。経路に対するメトリックに反映しないのは、その他の、たとえば RA メッセージの受信に伴う経路表の更新といった処理を簡単にするためである。

最終的に経路が検索されると、最も高いプリファレンスを持ち、次ホップにある NCE と協調して、“probably reachable” な状態にある最初のエントリを用い、それを同一メトリックのエントリの最後にリンクし直す。

3.3 Conclusion

本章では、Serialized Data State Processing の経路表管理への適用について説明した。

デフォルト経路に関しては、我々は、デフォルトルータに関係する情報を::/0 への経路情報として表現することにより、“Default Router List” を廃止した。Default Router Selection and Load Sharing between Routers については、ラウンドロビンの状態表現のためにリンク構造を用いている。この機構を用いることにより、現在の優先情報を管理するための外部からのリンクを廃止できる。また、一般的な経路に関するルータ間のロードシェアリングが可能となる。

4 IP パケット変換における状態処理

IP の役割は、データグラムを発信元から宛先に届けることである。従来の利用法では、あるがままの形で転送する既存の機能で十分であった。しかし、こんにち、種々の新しい機能を実現するため、IP パケットを途中で変換 (“transform”) する必要が出てきた。認証や暗号化といった、セキュリティ機能が典型的な例である。種々の変換に対して、効果的で柔軟かつ拡張性のある実装の方法論が必要となっている。

本章では、Serialized Data State Processing の汎用 IP パケット変換への適用について説明する。

4.1 XFRM と Stackable Destination

IP パケット処理のための新しい枠組みが、認証ヘッダ (AH)[6] や暗号化ペイロード (ESP)[7] といった IPsec の実装のために Linux 2.5.x に導入された。それらは “XFRM” と “Stackable Destination” と呼ばれている。

XFRM は変換器 (transformer) を意味する。その基本的なデータ構造は `xfrm_policy{}` 及び `xfrm_state{}` である。それぞれ IPsec Policy (SP) と Security Association (SA) を表現する。従って、Security Policy Database (SPD) は `xfrm_policy{}` から構成される。同様に、Security Association Database (SAD) は `xfrm_state{}` で構成される。`xfrm_state{}` は、パケット変換のためのテンプレート（あるいは変換手順）を表現する `xfrm_tmpl{}` を介して `xfrm_policy{}` と関連づけられている。

Stackable Destination (SD) は出力時のパケット変換のための基盤であり、プロトコル非依存な宛先キャッシュである `dst{}` の一種のリンクリストの形を取っている。このリストはポリシに従い一時的に生成され、キャッシュされる。`dst{}` はそれぞれ独自の出力メソッド `output` をもっており、それがパケット変換の状態を表現する `xfrm_state{}` と協調してパケットの変換を行う。

netlink[9] 基盤が、SAD と SPD を管理するための基本的なユーザインタフェースとして用いられている。この本来のインタフェースに加え、SAD には標準化されている `PF_KEY`[8] が、また、SPD には `PF_KEY` の KAME 拡張がサポートされている。

4.2 パケット処理の詳細

本節ではパケット処理の詳細について説明する。

4.2.1 出力系

出力処理はこのアーキテクチャが全般に用いられている。関数呼び出しは `xfrm_lookup()`, `xfrm_tmpl_resolve()`, `xfrm_bundle_create()`, `dst_output()` の順である。

まず、経路解決後、`xfrm_lookup()` が SPD 内に `xfrm_policy{}` を検索する。ここで、スタック中のパラメータ `dst{}` はもとの `dst{}` 構造を示している。`xfrm_tmpl_resolve()` は `xfrm_policy{}` 内にある、パケットをどのように処理すべきかを示す `xfrm_tmpl{}` を解決し、そのパケットのための `xfrm_state{}` の組を見つけるために `xfrm_lookup()` の中で呼ばれる。この処理は、IPsec ポリシに適合する IPsec SA (あるいは SA bundle) を検索することに対応する。

次に、`xfrm_bundle_create()` が Stackable Destination を作成する。これは IPsec SA (あるいは SA bundle) を作成することに対応する。

最後に、パケット生成後、`dst_output()` が呼ばれる。それぞれの出力ルーチンは `dst{}` にある関数ポインタで指定され、`dst{}` を次々に取り出していくに従ってリンクに沿って次々と呼ばれていく。この関数ポインタは、たとえば、暗号化処理をする `esp6_output()` などである。出力関数は `sk_buff{}` 構造中の `dst{}` ポインタを介して `xfrm_state{}` を使うことができる。最終的には、もとの `dst{}` の出力関数が呼ばれる。

4.2.2 入力系

入力処理は出力よりも簡単である。

他の全ての拡張ヘッダやプロトコルヘッダがその初期化段階で `inet6_protos[]` に登録されているのと同様、AH や ESP の処理ルーチンもその初期化段階で `inet6_protos[]` に登録される。

パケットが入力ハンドラに到達すると、カーネルはその

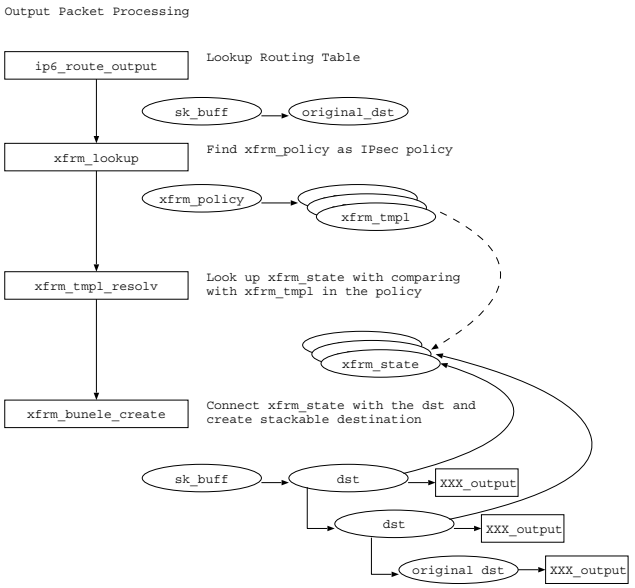

図 8: IPsec output process with XFRM and SD

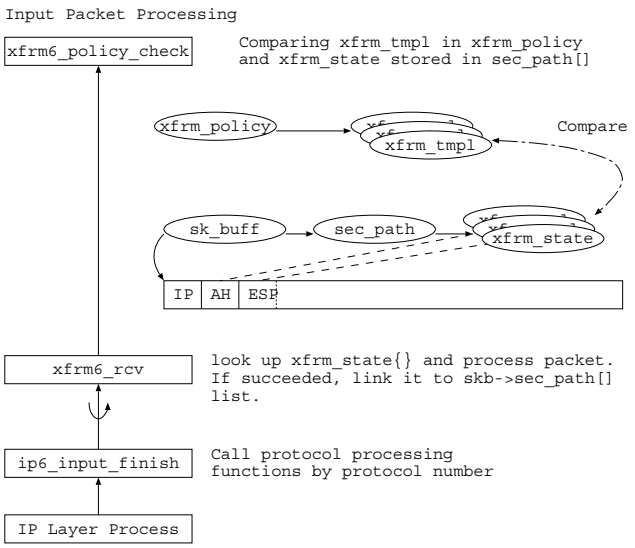

図 9: IPsec input process with XFRM

先頭から解析を開始し、登録テーブルに従ってハンドラを呼び出す。

IPsec のそれぞれのハンドラは xfrm_state{} を検索し、パケットを処理する。それが成功すれば、用いられた xfrm_state{} へのポインタが、パケットを記述する sk_buff{} 中の sec_path{} に保持される。

最後に、上位層処理の入力の直前で xfrm_policy_check() が呼ばれる。それは、xfrm_policy{} 中の xfrm_tmpl{} と sec_path{} 中の xfrm_state{} を比較し、パケットの上層への配送の可否を決定する。

4.3 試験結果

2003 年 4 月 24 日、Tom Lendacky 氏は netdev メーリングリストに対し、Linux 2.5.x IPsec に対する試験結果はきわめて良好であることを報告している (表 1)。これは、Serialized Data State Processing を用いた IPsec 実装が可能であることを示している。

表 1: Summary result of TAHI Conformance Test (Linux 2.5.58, %)

Test Series	Pass	Warn	Fail
ipsec	95	2	3
ipsec4	98	2	0
ipsec4-udp	96	4	0

4.4 Mobile IP への応用

XFRM 及び Stackable Destination は将来性のある枠組みであり、たとえば、Mobile IPv6[4] が実現可能である。

4.4.1 概略

図 10 に XFRM / Stackable Destination 機構を用いた Mobile IP 実装のプロトタイプ図を示す。

このプロトタイプでは、Mobile IP のために別の組のポリシを導入する。ポリシは Binding Update に基いてユーザ空間にあるモビリティデーモンと呼ばれるプログラムにより制御される。それは、変換の種類 (例えば、宛先オプションヘッダ中のホームアドレスオプションの付加) と、それに関連する、例えば、気付アドレスとホームアドレスといった binding data を記述する。

xfrm_bundle_create() は、処理の流れに応じて複数のテンプレートを一つの Stackable Destination にまとめる。例えば、AH, ESP 及び Mobile IP が共存している場合には、AH は全パケットを必要とする。しかし、パケット中のソースアドレスはホームアドレスでなければならない。従って、Stackable Destination は、例えば、以下のようになる。

- Mobile-IP Dest1 が気付アドレスをパケットに挿入
- Mobile-IP Rthdr が経路ヘッダ (type 2) をパケットに挿入
- ESP が、パケットの、ホームアドレスオプションのための宛先オプションヘッダ以降を暗号化
- AH が、パケットに対する署名データを生成し、宛先オプションヘッダの後ろに挿入
- Mobile-IP Dest2 がパケット中の気付アドレスとホームアドレスを交換

4.4.2 XFRM の Mobile IP 対応

従来の XFRM 基盤は IPsec のために設計、実装されたため、Mobile IP に対応するにいくつかの拡張が必要である。以下にその概要を示す。

検索条件項目の拡張

Mobile IPv6 では、ソースアドレスを用いて Binding を確認する必要がある。そのため、従来の宛先アド

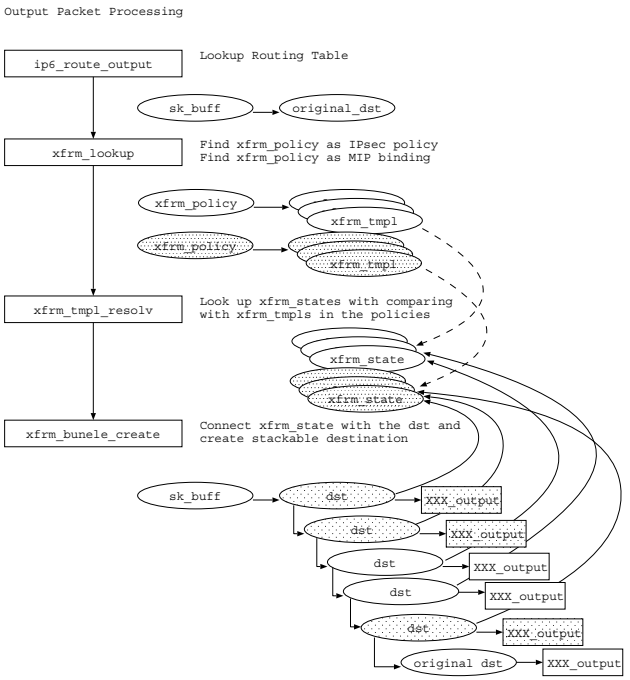

図 10: Mobile IP using XFRM / Stackable Destination Scheme

レス、Security Parameter Index (SPI) に加え、ソースアドレスを検索条件項目を加えている。

検索の拡張

Mobile IPv6 では必ずしも全ての条件項目を必要としない。また、ノードの役割と状態により、必要な項目は変わるため、条件に “don’t care” を与えることができるようにしている。

4.4.3 XFRM 方式の利点

Mobile IPv6 の実装では、ポリシールーチングの一種である、ソースアドレスによる経路制御を用いる方式が HUT[3] により提案されている。XFRM では、経路表で可能なアドレスに加え、ポート番号その他によってより精緻なポリシー管理が可能である。これにより、アドレス単位の移動透過性のサポートに加え、より粒度の細かい移動透過性のサポートなど、将来の拡張性を確保することができる。

4.5 まとめ

本章では、Serialized Data State Processing がいかに IP パケット変換に適用されるかについて説明した。この場合、AH, ESP or IPComp (IP Compression) [10] といった、それぞれの変換についての状態処理が線形化されている。Linux における IPsec 処理は、Serialized Data State Processing の一つの例あるいは実装法である IP パケット変換 (つまり XFRM) の枠組みに昇華されている。最後に、この方法が、いかに他の機能、例えば Mobile IP に自然に適用できるかを述べた。

5 まとめ

本稿では、従来の Linux システムに種々の IPv6 機能を実装するために Serialized Data State Processing を導入した。Serialized Data State Processing は、1 つあるいは複数のデータオブジェクトが有向グラフによって結合されており、それらが全体として働くような、自律データオブジェクトの一種の組み合わせである。

そのオブジェクトとその結合形態を適切に定義することにより、オブジェクトの独立性を保ち、不要な（外部）参照を排除することができる。特に、図 2 に示されるように線形的な結合形態を定義できれば、デッドロックを引き起こす資源依存性の衝突を容易に避けることができる。

本稿では、まず、デフォルトルータリストをその他の一般の経路と同じに扱う管理構造について説明した。

次に、提案手法である Serialized Data State Processing の一例である、XFRM に基づく IPsec および Mobile IP の実装について説明した。

本稿では、提案手法が単純で拡張性のあること示した。また、TAHI IPv6 仕様適合性試験により、我々の実装が実用段階の品質であることを示した。本稿で説明した実装は広く利用可能な Linux 頒布物である USAGI プロジェクトの IPv6 スタックに統合されている。

参考文献

[1] 6bone. 6bone Home Page. http://www.6bone.net.

[2] R. Drave and R. Hinden. Default router preferences, more-specific routes, and load sharing. Work in Progress, June 2002.

[3] GO/Core Project. MIPL Mobile IPv6 for Linux. http://www. mipl.mediapoli.com/.

[4] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. Work in Progress, June 2003.

[5] KAME Project. KAME Project Web Page. http://www. kame.net.

[6] S. Kent and R. Atkinson. IP Authentication Header. RFC2402, November 1998.

[7] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). RFC2406, November 1998.

[8] D. McDonald, C. Metz, and B. Phan. PF_KEY Key Management API, Version 2. RFC2367, July 1998.

[9] J. H. Salim, H. Khosravim, A. Kleen, and A. Kuznetsov. Netlink as an IP Services Protocol, July 2003.

[10] A. Shacham, B. Monsour, R. Pereira, and M. Thomas. IP Payload Compression Protocol (IPComp). RFC3173, September 2001.

[11] Keith Sklower. A tree-based packet routing table for berkeley unix. In *USENIX Winter*, pages 93–104, 1991.

[12] USAGI Project. USAGI Project Web Page. http://www. linux-ipv6.org.