(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2005-130502
(P2005-130502A)
(43)公開日　平成17年5月19日(2005.5.19)

(51)Int.Cl.[7]	F I		テーマコード（参考）
H04L 29/06	H04L 13/00	305Z	5K034

審査請求 未請求　請求項の数 23　ＯＬ　外国語出願　（全 26 頁）

(21)出願番号　特願2004-306402 (P2004-306402)
(22)出願日　平成16年10月21日 (2004.10.21)
(31)優先権主張番号　691515
(32)優先日　平成15年10月24日 (2003.10.24)
(33)優先権主張国　米国 (US)

(71)出願人　391030332
アルカテル
フランス国、７５００８　パリ、リュ・ラ・ボエティ　５４
(74)代理人　100062007
弁理士　川口　義雄
(74)代理人　100113332
弁理士　一入　章夫
(74)代理人　100114188
弁理士　小野　誠
(74)代理人　100103920
弁理士　大崎　勝真
(74)代理人　100124855
弁理士　坪倉　道明

最終頁に続く

(54)【発明の名称】加速されたパケット処理のための方法

(57)【要約】
【課題】パケットフローの上位層ヘッダにアクセスする方法を提供すること。
【解決手段】ＩＰｖ６は、ＩＰｖ４の進化的前進として開発された。ＩＰｖ６は、アドレッシングやルーティングなど、特定の領域で大きな改善をもたらすが、インターネットヘッダ長フィールドを排除している。その結果、拡張ヘッダが追加された上位層ヘッダ情報を得るためにパケットを処理することが、処理速度の低速化の結果となる場合がある。本発明は、拡張ヘッダとしても知られるヘッダオプションフィールドの長さに関する情報をキャッシュすることによって、この問題に対処する。ＩＰｖ６パケットのフローが受け取られたときに、パケットヘッダが拡張ヘッダを含む場合、キャッシュされた情報のレビューは、処理ステップを回避することができ、したがってパケット処理を加速させることができる。
【選択図】　図２

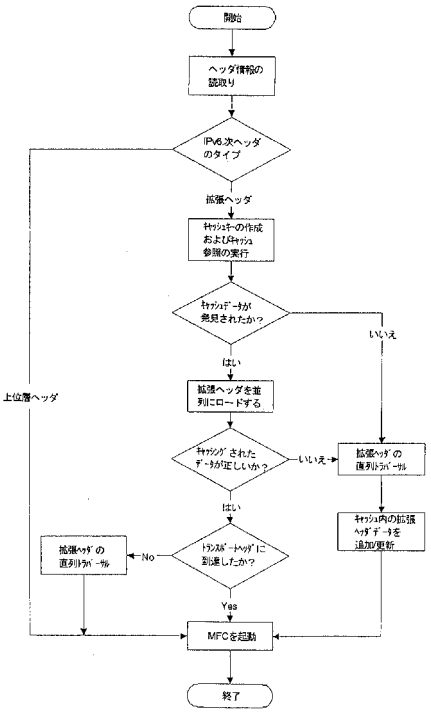

Figure 2 - 処理フローチャート

【特許請求の範囲】
【請求項１】
ａ）拡張ヘッダを含むパケットヘッダに応答して、キャッシュキーを作成し、キャッシュエントリについてキャッシュ参照を実行するステップと、
ｂ）上位層ヘッダに到達し、その上位層ヘッダ内のフィールドを読み取るために、対応するキャッシュエントリの発見に応答して、キャッシュエントリを使用して拡張ヘッダを並列に読み取るステップとを含む、パケットフローの上位層ヘッダにアクセスする方法。
【請求項２】
パケットフローがＩＰｖ６パケットを含む請求項１に記載の方法。
【請求項３】
キャッシュ参照が、ＩＰｖ６ヘッダ内に存在するフィールドに基づいて構成するタプルを使用して、拡張ヘッダの長さを含むテーブルで実行される請求項２に記載の方法。
【請求項４】
パケット処理時間をさらに短縮するために、キャッシュ参照が実行されている間に拡張ヘッダのグループ内の第１拡張ヘッダを読み取るステップを含む請求項３に記載の方法。
【請求項５】
キャッシュエントリが発見されない場合、拡張ヘッダの直列読取りを実行し、同一フローの後続パケットを処理するためにこれらの拡張ヘッダの長さに関する情報をキャッシュする請求項１に記載の方法。
【請求項６】
ホップバイホップおよびルーティング拡張ヘッダを備えたパケットを検出し、これらのパケットのオプション処理が必要かどうか決定するステップを含む請求項１に記載の方法。
【請求項７】
拡張ヘッダを含まないパケットに応答して、キャッシュ参照を実行せずに上位層ヘッダを読み取る請求項６に記載の方法。
【請求項８】
拡張ヘッダが常に変化していて、誤りのあるキャッシュされたデータとなる場合には、拡張ヘッダが直列にトラバースされる請求項１に記載の方法。
【請求項９】
常に変化している拡張ヘッダが、手動設定を使用して検出される請求項８に記載の方法。
【請求項１０】
常に変化している拡張ヘッダが、パケットフローの監視に基づいて動的に決定される請求項８に記載の方法。
【請求項１１】
キャッシュされたデータがパケットに一致しない場合には、キャッシュエントリが更新されない請求項１に記載の方法。
【請求項１２】
キャッシュがパケットに一致しないことの検出が、手動でイネーブルにされる請求項１１に記載の方法。
【請求項１３】
キャッシュがパケットに一致しないことの検出が、フロー監視に基づいてイネーブルにされる請求項１１に記載の方法。
【請求項１４】
上位層ヘッダからの情報がキャッシュされる請求項１に記載の方法。
【請求項１５】
キャッシュされる情報が、プロトコルならびに送信元および宛先ポート識別を含む請求項１４に記載の方法。
【請求項１６】

通常のパケット処理の間に実行される分類および参照の結果がキャッシュされる請求項１に記載の方法。

【請求項１７】

拡張ヘッダを含むパケットヘッダに応答して、キャッシュキーを作成し、キャッシュエントリについてキャッシュ参照を実行する手段と、

上位層ヘッダに到達し、その上位層ヘッダ内のフィールドを読み取るために、対応するキャッシュエントリの発見に応答して、キャッシュエントリを使用して拡張ヘッダを並列に読み取る手段とを含む、パケットフローの上位層ヘッダにアクセスするシステム。

【請求項１８】

パケットフローがＩＰｖ６パケットを含む請求項１７に記載のシステム。

【請求項１９】

キャッシュ参照を実行する手段が、第１拡張ヘッダを読み取る際に使用するためにヘッダからの情報を並列に検査する請求項１８に記載のシステム。

【請求項２０】

キャッシュエントリが発見されない場合、拡張ヘッダを読み取る手段がヘッダを直列にトラバースする請求項１９に記載のシステム。

【請求項２１】

特定の状況で、パケット処理の加速を試行するのにキャッシュされたデータを使用すべきでないことを指定する手段を含む請求項１７に記載のシステム。

【請求項２２】

上位層ヘッダ情報をキャッシュする手段をさらに含む請求項１７に記載のシステム。

【請求項２３】

上位層ヘッダ情報が、プロトコル及び送信元及び宛先ポートを含む請求項２２に記載のシステム。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、通信システムに関し、特に、キャッシュを使用してパケットフローの上位層ヘッダに迅速にアクセスするパケット処理スキームに関する。

【背景技術】

【０００２】

以下の議論では、特にインターネットプロトコルバージョン６（ＩＰｖ６：Ｉｎｔｅｒｎｅｔ ｐｒｏｔｏｃｏｌ ｖｅｒｓｉｏｎ ６）およびマルチフィールド分類（ＭＦＣ：ｍｕｌｔｉ－ｆｉｅｌｄ ｃｌａｓｓｉｆｉｃａｔｉｏｎ）に言及する。しかし、本発明の概念がＩＰｖ６およびＭＦＣだけに制限されず、記載されるインプリメンテーションが単に例示を目的としたものにすぎないことを理解すべきである。

【０００３】

ＩＰｖ６は、ＩＰｖ４のいくつかの制限を克服するための進化的発展を表す。特に、ＩＰｖ４のルーティングおよびアドレッシングの制限が、３２ビットのフィールド制限に基づいて利用可能なアドレス構成を現実的に制限している。ＩＰｖ６は、アドレスフィールドを１２８ビットに増加させた。これは、より多くのレベルのアドレッシング階層、ならびに非常に多数のアドレス可能ノードをサポートする。さらに、パケットハンドリングの処理コストを削減し、ＩＰｖ６ヘッダの帯域幅コストを制限するために、いくつかのＩＰｖ４ヘッダフィールドが廃止された。ＩＰｖ６には、送信側がサービス品質またはリアルタイムハンドリングに基づいて特別なハンドリングを要求する特定のトラフィックフローに属するパケットのラベリングを可能にする、新しい機能が追加された。さらに、パケットにオプションのインターネット層情報を追加するメカニズムが変更された。ＩＰｖ４では、これは、ＩＰｖ４ヘッダの一部であるオプションフィールドを通じて達成される。ＩＰｖ４ヘッダ内のインターネットヘッダ長（ＩＨＬ）フィールドは、これら追加のフィールドと基本ヘッダとの長さを示す。しかし、ＩＨＬフィールドの長さの理由から、パケッ

トには限られた数のオプションのみ追加することができる。ＩＰｖ６では、インターネット層情報は、拡張ヘッダを使用して追加される。これらのヘッダは、ＩＰｖ６ヘッダと上位層ヘッダとの間でＩＰｖ６パケットに追加される。各拡張ヘッダは、次ヘッダのタイプを識別するフィールドを含んでおり、ＩＰｖ６は、ＩＰｖ４のＩＨＬに類似したフィールドを使用する代わりに、拡張ヘッダ自体の中で各拡張ヘッダの長さを規定する。これで、無限の数の拡張ヘッダをＩＰｖ６パケットに加えることが可能になる。

【０００４】

図１は、フローラベルフィールドを含めたＩＰｖ６ヘッダ形式を示す。

【０００５】

通常のパケット処理は、典型的には、上位層ヘッダからフィールドを要求する。マルチフィールド分類（ＭＦＣ）は、そのようなデータを処理する多数の例のうちの１つであるが、通常、５－タプル＜送信元ＩＰ、宛先ＩＰ、送信元ポート、宛先ポート、プロトコル＞に基づいてパケットを分類する。オプションまたは拡張ヘッダが使用されるときには、パケット処理に必要なデータを検索するために上位層ヘッダに到達するのが困難でコストのかかるものになる場合がある。

【０００６】

ＩＰｖ４の場合、前述のＩＨＬフィールドを使用して、任意の付加オプションを含めたＩＰｖ４ヘッダ全体をスキップオーバーして上位層ヘッダに到達できるので、オプションを備えたパケットの処理が単純である。

【０００７】

ＩＰｖ６の場合、ＩＰｖ６ヘッダが上位層ヘッダに即時に到達するために使用できるＩＰｖ４のＩＨＬのようなフィールドをもたないので、拡張ヘッダを含むパケットでは、上位層ヘッダに到達するのが著しく困難である。先行技術の一解決策は、拡張ヘッダを直列にトラバース（ｔｒａｖｅｒｓｅ）することである。各拡張ヘッダの長さが拡張ヘッダ自体に格納されているので、これが最も自明の解決策であるように思われる。このように、上位層ヘッダに到達して通常のパケット処理を実行できる前に、いくつかの拡張ヘッダをトラバースする必要がある場合、ＩＰｖ６パケット処理は、コストのかかるものになる。ＩＰｖ６拡張ヘッダの存在は、特別なハンドリングがノードから必要とされることを必ずしも示さないので、このコストは、さらに不要に思われる。幾つかの拡張ヘッダは、宛先ノードだけに関する情報を含み、宛先に行く途中のすべてのノードが完全に無視できるものがある。実際、ホップバイホップ（ｈｏｐ－ｂｙ－ｈｏｐ）およびルータ拡張ヘッダだけがすべてのノードに関係があり、このルータ拡張ヘッダは、到達するパケットがノードに向かっているときにだけ利用可能である。幸いなことに、ＲＦＣ２４６０は、ホップバイホップ拡張ヘッダをＩＰｖ６ヘッダの後の第１拡張ヘッダに指定している。したがって、ノードは、単純にＩＰｖ６ヘッダの宛先アドレスおよび次ヘッダフィールドを調査することによって、パケットに特別なハンドリングが必要かどうか迅速に判定することができる。

【０００８】

従来のアプローチに代わる代替方法は、上位層ヘッダを調査する必要がないように、パケット分類で使用されるフィールドをＩＰヘッダからのフィールドだけに単純に制限することである。ＭＦＣの場合、このことが、標準の５－タプル分類を３－タプル分類に低減する。ＩＰｖ６のためのＭＦＣキーで使用されるフィールドを制限することは、ノードのセキュリティが制御される方式を変更することを意味するので、これは、あまり好評な選択肢ではない。この場合、ＩＰｖ６について新しいセットのルールを開発しなければならないが、これは単純に、上位層ヘッダからすべてのフィールドを読み取るのが不都合な場合があるからである。この理由から、パケット分類で使用されるフィールドを制限することは、対抗する解決策とは見なされない。

【発明の開示】

【発明が解決しようとする課題】

【０００９】

簡潔に言えば、本発明は、フローの後続パケットのハンドリングを加速させるための、パケットフロー内で使用される拡張ヘッダの長さに関する情報のキャッシュに関する。例示的な一実施形態では、パケットフローがＩＰｖ６パケットを含む。

【課題を解決するための手段】

【００１０】

したがって、本発明の第１の態様によれば、拡張ヘッダを含むパケットに応答して、ヘッダ内に存在するフィールドに基づいてキャッシュキーを作成し、キャッシュエントリについてキャッシュ参照を実行するステップと、上位層ヘッダに到達し、その上位層ヘッダ内のフィールドを読み取るために、対応するキャッシュエントリの発見に応答して、キャッシュエントリを使用して拡張ヘッダを並列に読み取るステップとを含む、パケットフローの上位層ヘッダにアクセスする方法が提供される。

【００１１】

本発明の第２の態様によれば、拡張ヘッダを含むＩＰｖ６パケットヘッダに応答して、ヘッダ内に存在するフィールドに基づいてキャッシュキーを作成し、キャッシュエントリについてキャッシュ参照を実行する手段と、上位層ヘッダに到達し、その上位層ヘッダ内のフィールドを読み取るために、対応するキャッシュエントリの発見に応答して、キャッシュエントリを使用して拡張ヘッダを並列に読み取る手段とを含む、パケットフローをフィルタリングするためのＭＦＣを実行するシステムが提供される。

【００１２】

ここで、本発明について、添付図面に即してさらに詳細に説明する。

【発明を実施するための最良の形態】

【００１３】

ノードがパケットを受け取ると、ＩＰｖ６ヘッダが検査される。拡張ヘッダをもたないパケットは、通常のパケットハンドリング技術を使用してハンドリングされる。これは、これらのパケットを処理するための唯一の追加コストが、拡張ヘッダの検査であることを意味する。ホップバイホップ拡張ヘッダを備えたパケット、またはルーティングヘッダを含むノードに向かうパケットの場合、ヘッダによって指定される特別なハンドリングをこの時点で起動することができる。

【００１４】

拡張ヘッダを備えたパケットの場合、ノードが、ＩＰｖ６ヘッダに含まれる情報を使用して、キャッシュ参照を実行するためのキーを作成する。このキーの形式は、すべてのパケットについて同一にすることができ、またはＩＰｖ６ヘッダ内のフィールドの存在もしくは値に基づいたものにすることができる。例えば、非ゼロのフローラベルを備えたパケットの場合、タプル＜ＩＰｖ６．送信元ＩＰ、ＩＰｖ６．フローラベル＞、または＜ＩＰｖ６．送信元ＩＰ、ＩＰｖ６．フローラベル、ＩＰｖ６．次ヘッダ＞を使用することができる。ゼロのフローラベルを備えたパケットの場合、タプル＜ＩＰｖ６．送信元ＩＰ、ＩＰｖ６．宛先ＩＰ＞または＜ＩＰｖ６．送信元ＩＰ、ＩＰｖ６．宛先ＩＰ、ＩＰｖ６．次ヘッダ＞を使用することができる。

【００１５】

キャッシュ参照を実行している間、ヘッダからの情報を使用して、リスト内の第１拡張ヘッダを読み取ることができる。キャッシュ内にエントリが発見されない場合、ノードは、拡張ヘッダの拡張ヘッダ長および次ヘッダのフィールドを使用して、直列に拡張ヘッダのリストのトラバースを進める。直列トラバーサルの後、拡張ヘッダをトラバースするのに使用されたデータは、類似の拡張ヘッダを備えた同一フローの他のパケットを予想してキャッシュに置かれる。しかし、キャッシュされたエントリが発見された場合、ノードは、第１拡張ヘッダおよびキャッシュされたデータからの情報を使用して、残りの各拡張ヘッダを並列に読み取る。これで、ノードが拡張ヘッダを迅速にトラバースできるようになり、ノードは、上位層ヘッダに到達し、または十分なデータがキャッシュされていない場合、最後の拡張ヘッダの直列トラバーサルを継続する。拡張ヘッダの並列トラバーサルのいずれかのポイントで、キャッシュされたデータが誤りであることがわかった場合、ノー

ドは、データが誤りであったポイントから、残りの拡張ヘッダを直列にトラバースしなければならない。この場合、類似の形式を備えた同一フローの他のパケットを予想して、現在のパケット内の拡張ヘッダを反映するようにキャッシュが更新される。

【００１６】

図２は、前述のアルゴリズムを要約している。ＩＰｖ６ヘッダを読み取った後にパケット内に拡張ヘッダが存在しないことがわかった場合、キャッシュ参照が実行されないことに留意すべきである。したがって、拡張ヘッダを使用するパケットが存在しない場合、これらの最適化をインプリメントするペナルティは、たとえあったとしてもわずかである。

【００１７】

図３は、拡張ヘッダ連結リストのトラバースと、加速されたアルゴリズム使用との間の、メモリアクセスの相違を示す。図示した事例では、パケット処理待ち時間が、１つのメモリアクセスの分だけ短縮される。図３Ａは、ヘッダの直列トラバーサルで使用されるメモリアクセスを示す。図３Ｂに示したように、並列処理は、第１拡張ヘッダのレビューと並列にステップ２Ｂでキャッシュを検査するものである。

【００１８】

本発明にはいくつか代替的な実施形態がある。第１の実施形態は、「予測不能」フローを識別して、それに対処する能力である。これは、常に拡張ヘッダが変化しているフローを検出するために使用される。そのようなフローが識別された場合、キャッシュされたデータが誤りの可能性があり、そして、並列トラバーサルを試行するのでなく拡張ヘッダを直列にトラバースすることがより効率的であることが明らかである。さらに、そのようなフローについては、現在のパケットの形式を反映するようにキャッシュを更新する必要がない。フローを「予測不能」としてマーキングすることは、手動設定によって達成することができ、またはフローの監視に基づいて動的に決定することができる。

【００１９】

第２の実施形態は、キャッシュエントリを「動かない（スティッキー）」として指定する能力である。これは、フロー内の大部分のパケットがキャッシュされたデータを使用して正しく解釈されるべきであり、また、キャッシュされたデータがパケットと一致しない場合には、キャッシュが更新されるべきでないことを示す。このことが、単一パケットがキャッシュされたデータを変更するのを防ぐ。前述した拡張機能のように、この機能は、手動でイネーブルにすることができ、またはフロー監視の結果としてイネーブルにすることができる。この拡張機能は、キャッシュ障害の場合に、キャッシュ内のデータが特定数の障害の後にだけ更新されることを指定することによって、向上させることができる。

【００２０】

第３の実施形態は、やはり、プロトコルや送信元及び宛先ポートなどの、いくつかの上位層ヘッダ情報をキャッシュすることである。キャッシュされた情報が受け取られると、プロトコル情報をＩＰｖ６ヘッダに含まれる情報と組み合わせて使用して、ＭＦＣなどのパケット分類を即時に開始することができる。分類が完了するときに、キャッシュされた残りの情報を使用して、拡張ヘッダを迅速にトラバースすることができ、そして、キャッシュされたプロトコル情報がパケットの情報と一致することを確認することができる。２セットの情報が一致する場合、分類が有効である。しかし、キャッシュからの情報が誤りの場合、上位層ヘッダからの正しい情報を用いて分類を繰り返す必要がある。最悪の場合、さらにもう１つ分類コストが存在するが、最良の場合、これで、パケット分類を拡張ヘッダのトラバーサルと並列に実施できるようになり、パケット分類に必要な待ち時間の分、パケット処理の合計待ち時間を短縮することができる。

【００２１】

他の実施形態は、通常のパケット処理の間に実行された他の分類および参照の結果をキャッシュする能力である。例えば、ＭＦＣおよび転送参照（ｆｏｒｗａｒｄｉｎｇ　ｌｏｏｋｕｐｓ）の結果を、キャッシュに追加することができる。この実施形態を使用すると、フローの第１パケットだけを完全に分類することのみが必要であり、これは、フローの後続パケットが、キャッシュからのこの分類の結果を使用できるからである。

【００２２】
本発明は、前述した先行技術の解決策に勝る利点を提供する。拡張ヘッダの大部分がほとんどのノードによる特別処理を必要としないので、本発明は、拡張ヘッダを含むＩＰｖ６パケットの処理に関連するコストを削減する。これで、パケット処理を継続できるように、上位層ヘッダに迅速に到達することを可能とする。
【００２３】
既存のＭＦＣタプルの使用に関して、上位層ヘッダの場所を迅速に見つけ出すことができるので、パケット分類で使用されるフィールドを、ＩＰｖ４に使用されたフィールドに対して変更せずにおくことができる。これは、既存のセキュリティポリシーを、ＩＰｖ４の場合と同一の方式でＩＰｖ６に適用できることを意味する。
【００２４】
拡張ヘッダ情報を並列にロードするコストは、キャッシュからデータを読み取る必要があるので、全メモリ帯域幅の少々の増加をもたらす場合があるが、この増加は、拡張ヘッダを含むパケットについてのみ発生する。
【００２５】
図４は、６個までの拡張ヘッダを並列にロードするタイムラインを示す。実線のバーは、拡張ヘッダのトラバーサルが最良の場合を示す。破線のバーは、キャッシュされた情報が誤りのときの、拡張ヘッダのトラバーサルが絶対的に最悪の場合を示す。図４Ａ～図４Ｇから、拡張ヘッダのトラバーサルが最良の場合、破線によって示される直列ヘッダ参照に関わる処理よりも、かなりの時間が節約されることがわかる。
【００２６】
キャッシュは、任意の数の方法でインプリメントすることができる。可能なインプリメンテーションには、コンテキストアドレス可能なメモリ（ＣＡＭ：ｃｏｎｔｅｘｔ　ａｄｄｒｅｓｓａｂｌｅ　ｍｅｍｏｒｙ）の使用、またはハッシュテーブルの使用を通じた方法が含まれる。おそらく最も明らかな方法は、既存のＣＡＭを使用することである。キャッシュとしてＣＡＭを使用することには、利点と欠点とがある。主な利点は、ＣＡＭを用いた場合、キーからのビットをすべてではなくてもほとんど使用して、正確なインプリメンテーションに応じて、キーを一致させるときに衝突が起こる可能性をほとんどまたは全くもたらさない参照を実行できることである。これは、ＣＡＭから間違った情報を取得することに起因した不一致がほとんどないことを意味する。ＣＡＭのインプリメンテーションのマイナス面は、ＣＡＭが高価でスペースに制約されるという事実を別にして、ＣＡＭ参照が、キャッシュ参照を実施するのに２回のメモリアクセスを必要とする場合があることである。第１のアクセスである書込みは、ＣＡＭにキーおよび参照を実行する命令を提供する。第２のアクセスである読取りは、ＣＡＭから結果を検索する。
【００２７】
ＣＡＭの使用に代わる代替方法として、ハッシュテーブルを使用してキャッシュをインプリメントすることができる。貴重でコストのかかるＣＡＭスペースを消費しないので、このインプリメンテーションには特定の利点がある。ハッシュ参照を実行できる速度の理由から、このインプリメンテーションは、恐らくは少なくともＣＡＭインプリメンテーションと同程度に速く、確実にＣＡＭインプリメンテーションよりもコストがかからない。ハッシュインプリメンテーションのマイナス面は、ＣＡＭインプリメンテーションに比べ、キーを一致させるときに衝突が起こる可能性が大きいことである。
【００２８】
正確にキャッシュがどのようにインプリメントされるかに応じて、キーを一致させるときに衝突が起こる可能性がある。これは、キャッシュへの索引としてとしてキーのビットすべてを使用しなかった結果である。結果として、この衝突を解決するためにさらなる処理が必要になる場合がある。特定のインプリメンテーションでは、これらの衝突を解決するコストが、キャッシュ情報の誤りが原因で拡張ヘッダを直列にトラバースするコストよりも大きくなることがある。これらの場合、単純に衝突が起こらないと仮定することが有益である。最終的には、キャッシュとパケットデータとの不一致の頻度を増加させるが、

処理を全体的に削減することになる。ハッシュインプリメンテーションの場合、衝突を解決するのに必要であった情報がもはや必要ないので、ハッシュテーブルエントリのサイズが縮小されるという追加的な利益がある。ハッシュテーブルエントリサイズの縮小は、テーブル内のエントリ数を効果的に増加させ、またはメモリの節約を可能にする。

【００２９】

前述したように、キャッシュする方法は、キャッシュへの索引を付けるためのキーを作成するために、ヘッダからのフィールドを使用する。キャッシュに格納される情報は、パケット内に存在すると思われる情報の複製である。この複製の理由から、キャッシュされたデータがパケット内のデータと確実に一致するように注意しなければならない。この目的で、拡張ヘッダがキャッシュされたデータと一致することを確認するためにロードされることが可能なように、最低でも拡張ヘッダの長さをキャッシュしなければならない。この検証が実行されなければ、悪意のあるホストが有効なフローを確立することが可能であるが、いくつかのセキュリティメカニズムを迂回するためにフローの後続パケットを修正することが可能である。一例として、単純にすべての拡張ヘッダの全長をキャッシュするインプリメンテーションを仮定する。多くの拡張ヘッダを含む、フローの第１パケットが到達すると、拡張ヘッダが直列にトラバースされ、エントリがキャッシュに加えられる。ＭＦＣは、パケットに関して実行され、パケットが受け入れられる。フローの別のパケットが到達し、キャッシュされたデータが読み取られる。拡張ヘッダの合計オフセットが使用されて、上位層ヘッダであると思われるものが読み取られるが、これは、実際には第１パケットのヘッダと同一の偽のヘッダである。ＭＦＣが実行され、パケットが受け入れられる。拡張ヘッダが直列的に読み取られたので、明らかに、この第２パケットは、第１パケットよりも拡張ヘッダ数が少なく、実際の上位層ヘッダは、キャッシュされたデータによって指される場所よりもパケット内の早い場所で見付かる。

【００３０】

本発明の特定の実施形態について記載し説明したが、本発明の基本概念から逸脱することなく多くの変更を実施できることが当業者には明らかであろう。しかしながら、そのような変更が、特許請求の範囲によって定義される本発明の全範囲内に入ることを理解すべきである。

【図面の簡単な説明】

【００３１】

【図１】ＩＰｖ６ヘッダ形式を示す図である。

【図２】ＩＰｖ６パケットの上位層ヘッダに到達するのに必要な動作シーケンスを示す図である。

【図３】直列処理および複数の処理を使用するメモリアクセスの比較を示す図である。

【図４】オプションを使用する拡張ヘッダ参照のタイムラインを示す図である。

【図１】

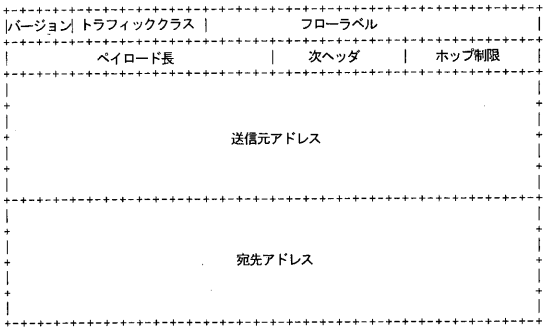

Figure 1 - IPv6ヘッダ

【図２】

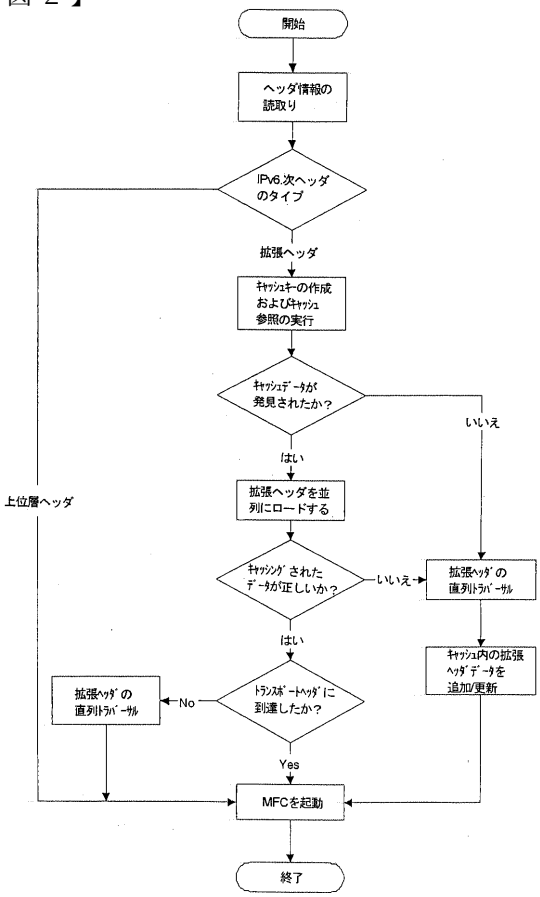

Figure 2 - 処理フローチャート

【図３】

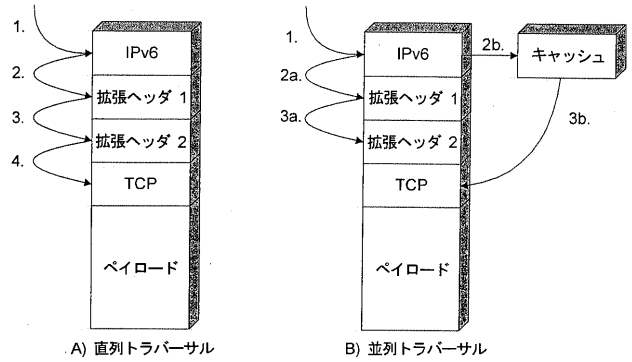

Figure 3 - メモリアクセスの比較

【図４】

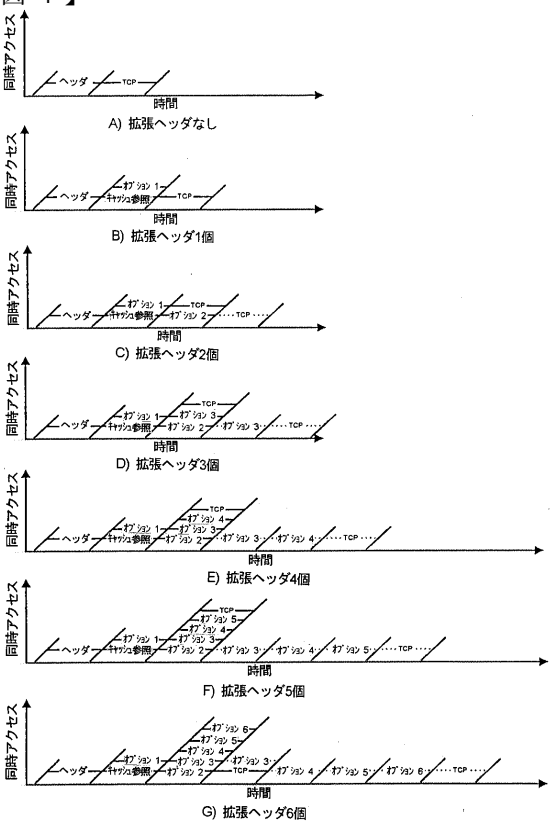

Figure 4 - キャッシュを使用する拡張ヘッダ参照のタイムライン

フロントページの続き

(72)発明者　デイビツド・ジエームズ・ウイルソン
カナダ国、オンタリオ・ケー・２・エム・２・ケー・６、カナタ、タマラ・ウエイ・４

Ｆターム(参考) 5K034 AA02　DD01　EE11　HH61

【外国語明細書】

Specification

Title of Invention

METHOD FOR ACCELERATED PACKET PROCESSING

Field of the Invention

This invention relates to communication systems and in particular to a packet processing scheme in which a cache is used to quickly access upper-layer headers in packet flows.

Background of the Invention

In the following discussion specific reference is made to Internet protocol version 6 (IPv6) and multi-field classification (MFC). It is to be understood, however, that the concepts of the present invention are not limited to IPv6 and MFC but the described implementation is intended as exemplary only.

IPv6 represents an evolutionary development to overcome some limitations in IPv4. In particular, the routing and addressing limitations of IPv4 place a real limit on the address configurations available based on a 32 bit field limit. IPv6 has increased the address field to 128 bits. This supports more levels of addressing hierarchy as well as a much greater number of addressable nodes. Additionally, some IPv4 header fields have been dropped to reduce the processing cost of packet handling and to limit the bandwidth cost of the IPv6 header. A new capability has been added to IPv6 which enables labeling of packets belonging to particular traffic flows for which the sender requests special handling based on quality of service or real time handling. Additionally, the mechanism to add optional internet-layer information to a packet has changed. In IPv4, this is achieved through the option field which is a part of the IPv4 header. The Internet Header Length (IHL) field in the IPv4 header indicates the length of these extra fields and the basic header. However, due to the length of the IHL field, only a limited number of options could be added to a packet. In IPv6, internet-layer information is added using extension headers. These headers are added to an IPv6 packet between the IPv6

header and the upper-layer header. Each extension header contains a field identifying the type of the next header, and instead of using a field similar to the IHL of IPv4, IPv6 specifies the length of each extension header in the extension headers themselves. This allows an unlimited number of extension headers to be added to an IPv6 packet.

Figure 1 shows the IPv6 header format including the flow label field.

Regular packet processing typically requires fields from upper-layer headers. Multi-Field Classification (MFC), one of many examples of processing such data, usually classifies packets based on the 5-tuple <Source IP, Destination IP, Source Port, Destination Port, Protocol>. When options or extension headers are used, it can become difficult and costly to reach the upper-layer header in order to retrieve the data necessary for packet processing.

For IPv4, processing packets with options is simple as the aforementioned IHL field can be used to skip over the entire IPv4 header, including any appended options, to reach the upper-layer header.

For IPv6, reaching the upper-layer header is significantly more difficult for packets containing extension headers as the IPv6 header does not have a field like the IHL in IPv4 that can be used to immediately reach the upper-layer header. One prior art solution is to traverse the extension headers serially. Since the length of each extension header is stored in the extension header itself, this appears to be the most obvious solution. Thus, IPv6 packet processing becomes costly if a number of extension header need to be traversed before the upper-layer header can be reached and regular packet processing performed. This cost seems even more unnecessary since the presence of IPv6 extension headers does not necessarily indicate that special handling is required from the node. Some extension headers

contain information relevant only to the destination node, and can be completely ignored by all nodes en-route to the destination. In fact, only hop-by-hop and router extension headers are relevant to all nodes and the latter is only applicable when the arriving packet is destined for the node. Fortunately, RFC2460 specifies that hop-by-hop extension headers be the first extension header after the IPv6 header. Thus a node can quickly determine if special handling for a packet is required simply by examining the destination address and the Next Header field of the IPv6 header.

An alternative to the previous approach is to simply limit the fields used in packet classification to those from the IP header such that the upper-layer header does not need to be examined. For MFC, this reduces the standard 5-tuple classification to a 3-tuple classification. This is not a very popular option as limiting the fields used in the MFC key for IPv6 means changing the way security of the node is controlled. Thus, a new set of rules must then be developed for IPv6 simply because it is sometimes inconvenient to read all the fields from the upper-layer header. Because of this, limiting the fields used in packet classification is not considered a competing solution.

Summary of the Invention

Simply stated, the present invention relates to the caching of information about the length of extension headers used within a packet flow in order to accelerate the handling of subsequent packets in the flow. In an exemplary embodiment the packet flow comprises IPv6 packets.

Therefore in accordance with a first aspect of the present invention there is provided a method of accessing upper-layer headers in a packet flow, comprising the steps of: responsive to a packet containing extension headers, building a cache key based on the fields present in the header and performing a cache lookup for a

cache entry; and responsive to finding a corresponding cache entry, reading extension headers in parallel using the cache entry to arrive at and read fields in the upper-layer header.

In accordance with a second aspect of the present invention there is provided a system for performing MFC for filtering a packet flow, comprising: means responsive to an IPv6 packet header containing extension headers, for building a cache key based on the fields present in the header and performing a cache lookup for a cache entry; and means responsive to finding a corresponding cache entry, for reading extension headers in parallel using the cache entry to arrive at and read fields in the upper-layer header.

The invention will now be described in greater detail with reference to the attached drawings.

Detailed Description of the Invention

When a node receives a packet, the IPv6 header will be examined. Packets which have no extension headers will be handled using regular packet handling techniques. This means that the only additional cost for processing these packets is the check for extension headers. For packets with the hop-by-hop extension header, or packets destined to the node containing routing headers, the special handling specified by the header can be launched at this point.

For packets with extension headers the node will use the information contained in the IPv6 header to build a key in order to perform a cache lookup. The format of this key could be the same for all packets, or could be based on the presence or value of fields in the IPv6 header. For example, for packets with non-zero flow labels the tuple <IPv6.srcIP, IPv6.flowLabel>, or <IPv6.srcIP, IPv6.flowLabel, IPv6.nextHeader> could be used. For packets with a zero flow label the tuple <IPv6.srcIP, IPv6. dstIP> or <IPv6.srcIP, IPv6. dstIP, IPv6.nextHeader> could be used.

While the cache lookup is executing, the information from the header can be used to read the first extension header in the list. If no entry is found in the cache the node proceeds to traverse the list of extension headers serially using the extension header length and next header fields of the extension headers. After a serial traversal, the data used to traverse the extension headers is placed in the cache in anticipation of more packets in the same flow with similar extension headers. However, if a cached entry is found, the node uses the information from the first extension header and the cached data to read each of the remaining extension headers in parallel. This allows the node to quickly traverse the extension headers and it will either arrive at the upper-layer header, or, in the event that not enough data was cached, continue with a serial traversal of the final extension headers. If the cached data is found to be incorrect at any point in the

parallel traversal of the extension headers, the node must traverse the remaining extension headers serially from the point at which the data was incorrect. In this case the cache would be updated to reflect the extension headers in the current packet in anticipation of additional packets in the same flow with similar format.

Figure 2 summarizes the algorithm described above. It should be noted that, if after reading the IPv6 header it is known that there are no extension headers in the packet then no cache lookup is performed. Thus, in cases where there are no packets using extension headers there will be only a small, if any, penalty for implementing these optimizations.

Figure 3 shows the difference in the memory accesses between traversing the extension header linked list and using the accelerated algorithm. In the case shown, the packet processing latency is reduced by one memory access. Figure 3A shows the memory accesses used in a serial traversal of the header. As shown in Figure 3B the parallel processing involves at step 2B an examination of the cache in parallel with the review of the first extension header.

There are several alternate embodiments to the invention. The first embodiment is the ability to identify, and react to "unpredictable" flows. This would be used to detect flows in which the extension headers were constantly changing. If such a flow were identified, it would be clear that the cached data would likely be incorrect, and that it would be more efficient to traverse the extension headers serially, instead of attempting a parallel traversal. Additionally, for such a flow, updating the cache to reflect the format of the current packet would be unnecessary. Marking a flow as "unpredictable" could be the result of manual configuration or determined dynamically based on observing the flow.

The second embodiment is to have the ability to specify a cache entry as "sticky". This would indicate that the majority of the packets in a flow should be correctly interpreted using the cached data, and that in the event that the cached data does not match the packet the cache should not be updated. This would prevent a single packet from changing the cached data. As in the previous extension, this feature could be manually enabled or enabled as a result of flow observations. This extension could be enhanced by specifying that in the case of cache failures, the data in the cache would only be updated after a certain number of failures.

A third embodiment is to also cache some of the upper-layer header information such as the protocol and the source and destination ports. When the cached information is received, the protocol information, combined with the information contained in the IPv6 header, can be used to immediately begin packet classification, such as MFC. While the classification is completing, the remaining cached information can be used to quickly traverse the extension headers and to verify that the cached protocol information matches that of the packet. If the two sets of information match then the classification is valid. If, however, the information from the cache is incorrect, classification needs to be repeated with the correct information from the upper-layer header. In the worst case, there is a cost of one additional classification, but in the best case this allows for packet classification to be done in parallel with the extension header traversal and can reduce the total packet processing latency by the latency required for packet classification.

A still further embodiment is the ability to cache the results of other classifications and lookups performed during regular packet processing. For example, the result of the MFC, and forwarding lookups could be added to the cache. Using this embodiment, only the first packet in a flow needs to be fully

classified as subsequent packets in the flow are able to use the results of this classification from the cache.

The present invention provides advantages over the previously discussed prior art solutions. As a large portion of extension headers require no special processing by most nodes, the present invention reduces the cost associated with processing IPv6 packets containing extension headers. This allows the upper-layer headers to be quickly reached such that packet processing can continue.

With respect to the use of existing MFC tuples, since the upper-layer header can be quickly located, the fields used in packet classification can remain unchanged with respect to those which were used for IPv4. This means that existing security policies can be applied to IPv6 in the same manner as they were in IPv4.

The cost of loading extension header information in parallel may results in a small increase in total memory bandwidth due to the need to read the data from the cache, but this increase will only occur for packets which contain extension headers.

Figure 4 shows the timelines for the parallel loading of up to six extension headers. The solid bars indicate the best case traversal of the extension headers. The dashed bars show the absolute worst case traversal of the extension headers in the event that the cached information is incorrect. It can be seen from Figures 4A to 4G that in the best case traversal of the extension headers considerable time is saved from the process involving serial header lookups as indicated by the dashed bars.

The cache can be implemented in any number of ways. Possible implementations include the use of a context addressable memory (CAM) or through the use of hash tables. Perhaps the most obvious method is to use an existing CAM. Using the CAM as a cache has its advantages and disadvantages. The main advantage is that with a CAM is that most, if not all, the bits from the key can be used to perform a lookup resulting in very low or no chance, depending on the exact implementation, for there to be a collision when matching a key. This means there will be a very few mismatches due to obtaining the wrong information from the CAM. The downside to a CAM implementation, apart from the fact that CAMs are expensive and constrained by space, is that the CAM lookup may require two memory accesses to do a cache lookup. The first access, a write, provides the CAM with the key and the instruction to perform a lookup. The second access, a read, retrieves the result from the CAM.

As an alternative to using the CAM a hash table can be used to implement the cache. This implementation has certain advantages since it does not consume precious and costly CAM space. Due to the speed that a hash lookup can be performed, this implementation is likely at least as fast, and certainly less costly, as the CAM implementation. The downside to the hash implementation is that there is a greater probability for a collision when matching a key compared to the CAM implementation.

Depending on exactly how the cache is implemented, it may be possible for a collision to occur when matching a key. This is the result of not using all of the bits of the key as an index into the cache. The effect is that further processing may be required to resolve this collision. It may be that in certain implementations, the cost of resolving these collisions is greater than the cost of serially traversing the extension headers due to incorrect cache information. In these cases, it would be beneficial to simply assume that collisions do not occur. The end result would be

an increase in the frequency of cache and packet data mismatch, but an overall reduction in processing. For hash implementations, this has the additional benefit of reducing the size of the hash table entries as the information that was needed to resolve collisions is no longer needed. The reduction in hash table entry size effectively increases the number of entries in the table, or allows for memory savings.

As discussed previously, the caching method uses fields from the header in order to build a key to index into the cache. The information stored in the cache is a duplication of the information believed to be in the packet. Because of this duplication, care must be taken to ensure that the cached data matches the data in the packet. In order to do this, at a minimum the lengths of the extension headers must be cached so that the extension headers can be loaded to confirm they match the cached data. If this validation were not performed it would be possible for a malicious host to establish a valid flow, but modify subsequent packets in the flow in order to bypass some security mechanism. As an example, assume an implementation that simply caches the total length of all extension headers. When the first packet, containing many extension headers, in the flow arrives the extension headers are traversed serially, and an entry is added to the cache. MFC is performed on the packet, and the packet is accepted. Another packet in the flow arrives, and the cached data is read. The total extension header offset is used to read what is believed to be the upper-layer header, but which actually a fake header identical to that of the first packet. MFC is performed, and the packet is accepted. Had the extension headers been read serially it would have been clear that this second packet had fewer extension headers that the first packet, and that the real upper-layer header was located earlier in the packet than the one pointed to by the cached data.

Although particular embodiments of the invention can be described and illustrated it will be apparent to one skilled in the art that numerous changes can be made without departing from the basic concept of the invention. It is to be understood, however, that such changes will fall within the full scope of the invention as defined by the appended claims.

Brief Description of Drawings

Figure 1 illustrates the IPv6 header format.

Figure 2 illustrates the sequence of operations needed to reach the upper-layer header of an IPv6 packet.

Figure 3 shows a comparison of memory accesses using serial and plural processing.

Figure 4 illustrates time lines of extension header lookups using options.

Claims

1. A method of accessing upper-layer headers in a packet flow, comprising the steps of:

   a) responsive to a packet header containing extension headers, building a cache key and performing a cache lookup for a cache entry; and

   b) responsive to finding a corresponding cache entry, reading extension headers in parallel using the cache entry to arrive at and read fields in the upper-layer header.

2. The method as defined in claim 1 wherein the packet flow comprises IPv6 packets.

3. The method as defined in claim 2 wherein the cache lookup is performed on a table containing lengths of extension headers using a tuple comprising based on the fields present in the IPv6 header

4. The method as defined in claim 3 including the step of reading the first extension header in the group of extension headers while the cache lookup is being performed to further reduce packet processing times.

5. The method as defined in claim 1 wherein if no cache entry is found, performing a serial read of the extension headers and caching information on the lengths of these extension headers for processing subsequent packets in the same flow.

6. The method as defined in claim 1 including the step of detecting packets with hop-by-hop and routing extension headers and determining whether options processing of those packets is required.

7. The method as defined in claim 6 wherein, responsive to a packet not containing extension headers, read the upper-layer header without performing a cache lookup.

8. The method as defined in claim 1 wherein if the extension headers are constantly changing resulting in incorrect cached data, the extension headers are traversed serially.

9. The method as defined in claim 8 wherein the constantly changing extension headers are detected using a manual configuration.

10. The method as defined in claim 8 wherein the constantly changing extension headers are determined dynamically based on observations of the packet flow.

11. The method as defined in claim 1 wherein if the cached data does not match the packet the cache entry is not updated.

12. The method as defined in claim 11 wherein detection that the cache does not match the packet is enabled manually.

13. The method as defined in claim 11 wherein detection that the cache does not match the packet is enabled based on flow observations.

14. The method as defined in claim 1 wherein information from the upper-layer header is cached.

15. The method as defined in claim 14 wherein the cached information includes protocol and source and destination port identification.

16. The method as defined in claim 1 wherein results of classification and lookups performed during regular packet processing are cached.

17. A system for accessing upper-layer headers in a packet flow, comprising:

means responsive to a packet header containing extension headers, for building a cache key and performing a cache lookup for a cache entry; and

means responsive to finding a corresponding cache entry, for reading extension headers in parallel using the cache entry to arrive at and read fields in the upper-layer header.

18. The system as defined in claim 17 wherein the packet flow comprises IPv6 packets.

19. The system as defined in claim 18 wherein the means for performing a cache lookup examines, in parallel, information from the header to use in reading a first extension header.

20. The system as defined in claim 19 wherein if no cache entry is found means for reading extension headers traverses the headers serially.

21. The system as defined in claim 17 including means to specify, in certain situations, that cached data should not be used to attempt to accelerate packet processing.

22. The system as defined in claim 17 further including means to cache upper-layer header information.

23. The system as defined in claim 22 wherein the upper-layer header information includes protocol, and source and destination ports.

1. Abstract

IPv6 has been developed as an evolutionary advance of IPv4. Although IPv6 offers considerable improvement in certain areas such as addressing and routing it has eliminated the Internet header length field. As a result processing of packets to obtain the upper-layer header information in which extension headers have been added can result in slower processing rates. The present invention addresses this issue by caching information relating to the length of header option fields also known as extension headers. When a flow of IPv6 packets is received and if the packet header includes extension headers a review of the cached information can avoid processing steps and hence accelerate packet processing.

2. Representative Drawing

Fig. 2

Fig. 1

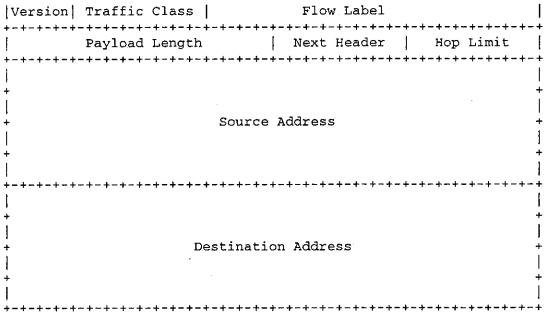

Figure 1 - IPv6 Header

Fig. 2

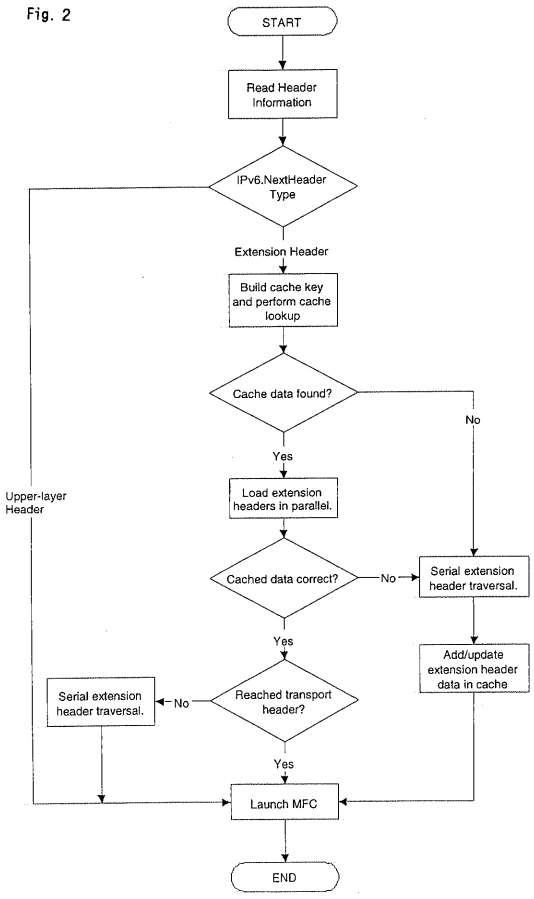

Figure 2 - Processing Flow Chart

Fig. 3

Figure 2 - Processing Flow Chart

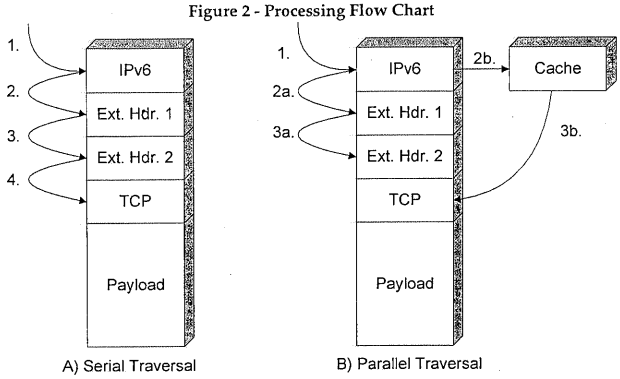

Figure 3 - Comparison of Memory Accesses

Fig. 4

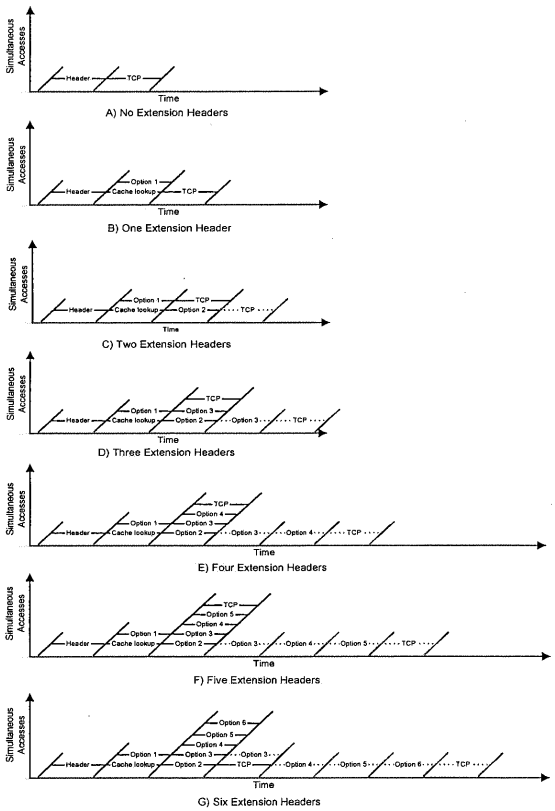

Figure 4 – Timeline of Extension Header Lookups Using Cache