卒業論文 2023 年度（令和 5 年度）

Linux Netfilter の SRv6 への統合

慶應義塾大学 環境情報学部
澤田 開杜

卒業論文要旨 - 2023 年度（令和 5 年度）

Linux Netfilter の SRv6 への統合

本論文では，Linux の持つパケット操作機能である netfilter を，トラフィック制御技術の 1 つである SRv6 に統合する手法を提案する．昨今のデータセンタネットワークでは，汎用的なサーバや仮想マシン，コンテナ技術を使ってネットワークの機能 (NF) を仮想化する技術が一般化してきている．一連のルールに沿って NF を適用することを Service Function Chaining (SFC) という．SFC にデプロイされる NF は Service Function (SF) と呼ばれ，SFC ではあるパケットを任意の順番で Service Function へ通過させる必要がある．従来のパケットルーティングでは，あるパケットを任意の順番で指定したノードを通過させる，ということはでききない．よって，SFC の実現のためには従来のパケットルーティングとは別の経路制御機構が必要である．SFC を実現できる経路制御技術の 1 つに Segment Routing over IPv6 (SRv6) という技術がある．SRv6 では，SRv6 header と呼ばれるヘッダで IP パケットをカプセル化する．また，SRv6 header にはパケットが通過するノードが順番に含まれる．これによって，IP 的なベストパスに関係なくパケットが通過するノードを指定可能であるから，任意のルールに従って SF を通る順番を指定できる．また，SRv6 はトラフィック制御だけでなく，トランジットするパケットに対して特定の操作を適用でき，この特定の操作の種類のことを**ビヘイビア**という．

Linux kernel には netfilter というパケット処理フレームワークが実装されている．netfilter を使うことで，パケットのフィルタリングや NAT，NAPT，その他のパケットマングリング操作を適用できる．しかし，SRv6 は SRv6 header でカプセル化されているため，カプセル化されている内部のパケットに対して netfilter を適用することができない．そこで，本論文では End.AN.NF という新しい SRv6 ビヘイビア を提案する．End.AN.NF は Linux netfilter を Linux の SRv6 ルーティングインフラストラクチャへ統合する事ができる．End.AN.NF は，SRv6 を利用したサービスファンクションチェイニング環境において，Linux netfilter を SRv6 に対応した NF として扱えるようにする．End.AN.NF を利用する際，netfilter を利用して作成されたアプリケーションの実装を変える必要はなく，End.AN.NF は SRv6 の基本処理である End ビヘイビアを実行しながら，SRv6 でカプセル化された内部のパケットへ netfilter を適用できる．さらに，End.AN.NF は，パケットにマークを付けることができる．したがって，netfilter を利用して作成されたアプリケーションは End.AN.NF がパケットバッファに付与したマークを照合することで，適用するルールを変更できる．我々は End.AN.NF を Linux kernel に実装し，その性能評価を行った．計測の結果， End.AN.NF は End.DT4 と H.Encaps を使って SRv6 でカプセル化された内部パケットに netfilter を適用する方法に比べ，27% 高いスループット，及び 3.0 マイクロ秒低いレイテンシを実現した．

キーワード:
1. Service Function Chaining 2. Segment Routing 3. SRv6

慶應義塾大学 環境情報学部
澤田 開杜

Abstract of Bachelor's Thesis - Academic Year 2023

Integrating Netfilter into SRv6 Routing Infrastructure of Linux as an SR-Aware Network Function

This paper proposes a new SRv6 End behavior, called End.AN.NF, integrating Linux netfilter as a network function for service function chaining by Segment Routing (SR). End.AN.NF allows netfilter-based applications to be executed as SR-Aware applications without modification, as it applies netfilter to inner packets encapsulated in SRv6 while performing the basic SRv6 End behavior. Furthermore, End.AN.NF utilizes the argument of the segment identifiers to mark packets. Consequently, this enables netfilter-based applications to match the marks on packet buffers and change rules to be applied. We implemented End.AN.NF on the Linux kernel and evaluated its performance. The evaluation shows that End.AN.NF achieves 27% higher throughput and 3.0 microseconds lower latency than applying netfilter to SRv6-encapsulated inner packets by End.DT4 and H.Encaps.

Keywords :
1. Service Function Chaining 2. Segment Routing 3. SRv6

Keio University Bachelor of Arts in Environment and Information Studies
Kaito Sawada

目 次

図 目 次

表 目 次

第1章　序論

1.1　Service Function Chaining

SFC とは，エンドツーエンド通信を提供するために必要なさまざまなサービスファンクション (SF) を決定及び順序付けし，それらを介するようにトラフィックを操作することを指す．SF には，ファイアウォールや IP ネットワークアドレストランスレータ (NAT) などのネットワークサービスファンクションや，アプリケーション固有の機能が含まれる．SFC アーキテクチャは，基礎となるネットワークトポロジから独立したトポロジを前提としている．この基礎となるネットワークトポロジをアンダーレイネットワークといい，独立した SFC のためのネットワークトポロジをオーバレイネットワークという．SFC アーキテクチャでは，パケットは通信の入口となるノードで分類され，そのノードで SFC 対応ドメイン内で適用する SF のセットを決める．その後，任意の順番で各ファンクションで処理されるように転送される．

SFC アーキテクチャは，ネットワークの用途や拡張計画などのコンテキストに依存しない，汎用的な場面で利用可能な技術である．つまり，SFC アーキテクチャは固定ネットワークやモバイルネットワーク，多くのデータセンターアプリケーションに適用できる．SFC の構築においては，すべてのサービス機能 (SF) に適用できる標準の定義や特性は存在せず，各 SF は不透明な処理要素として扱われる．また，特定の管理ドメインで有効な SF のグローバルリストや標準リストは存在せず，SF のセットは現在アクティブなサービスの機能であり，時間やネットワーク環境によって異なる場合がある．SF のチェインとそれらを呼び出す基準は，SF 対応ドメインを運用する各管理エンティティに固有である．

SFC におけるサービス機能 (SF) は，受信したパケットの特定の処理を担当する機能である．SF はプロトコルスタックのさまざまなレイヤで動作し，論理コンポーネントとして仮想要素として実現されるか，物理ネットワーク要素に組み込まれることがあ

1.2　ソースルーティング

[**Service Function Chaining に必要な技術であるソースルーティングについて，ソースルーティングを実現できる技術ついて説明するのだ**]

1.3 SRv6

SRv6 は，IPv6 を使用するソースルーティングアーキテクチャの 1 つである．SRv6 では，SRv6 ヘッダ (SRH) と呼ばれる IPv6 拡張ヘッダに一連の識別子を埋め込むことで，ネットワークオペレータやアプリケーションはパケットが通過する中間地点を指定できる．識別子はセグメント識別子 (SID) と呼ばれ，各 SID はネットワーク内の特定の場所で実行される，特定の機能を表す．リスト状になっている SID の中で，どれが現在有効な SID であるかを指定するために，SRH にはセグメントレフト (segleft) と呼ばれるフィールドがある．segleft は SID リストのインデックスで，(SID の合計) – 1 から始まり，0 で終わる．SID は，ネットワーク内の特定のセグメントに関連付けられた IPv6 アドレスである．SID は `LOC:FUNCT:ARG` という構造になっており，IPv6 アドレスであるため，その長さの合計は 128bit である．**ここから先の文章は前提知識がないと分かりにくいので本綴じまでに修正** [`LOC` はロケータを表し，`FUNCT` は SID に関連付けられた転送動作の識別であり，`ARG` は転送動作に必要な追加情報をエンコードする領域である．また，ロケータは `B:N` と表すこともできる．ここで，`B` は SRv6 SID ブロック (SRv6 SID のために割り当てられた IPv6 プレフィックス) であり，`N` は SID をインスタンス化するノードの識別子である．]

SRv6 ノードは，宛先 IPv6 アドレスがノードに設定されたローカル SID であるパケットを受信すると，SID に関連付けて事前定義された動作を実行する．SRv6 のコンテキストにおいて，SRv6 ノードが実行する振る舞いは End ビヘイビアと称される．現行の RFC8986 [1] では，15 種類の End ビヘイビアが定義されており，その中で最も基本的なものは End である．End ビヘイビアは，受信したパケットの SRH の segleft をデクリメントし，宛先 IPv6 アドレスを次の SID に置換する．続いて，SRv6 ノードは更新された宛先 IPv6 アドレスに基づき，パケットを次のホップに転送する．また，RFC8986 では，SID リストを含む SRH でパケットをカプセル化する動作も定義されており，これは Headend ビヘイビアと呼ばれる．

SRv6 の SID は IPv6 アドレスであるため，既存のルーティングプロトコルを用いて SID を経路情報として広告することにより，SRv6 ベースのネットワーク構築が可能である．例えば，H.Encaps と End.DT4 はそれぞれヘッドエンドビヘイビア及びエンドビヘイビアに該当し，これら 2 つのビヘイビアを組み合わせることで layer-3 VPN を構成できる [2]．H.Encaps は IPv4 または IPv6 パケットを SRH を含んだ IPv6 ヘッダで包み，一方 End.DT4 は SRH でカプセル化されたパケットの外部ヘッダを外す．このようにパケットを新たな外部ヘッダで包むことをカプセル化と称し，包まれた外部ヘッダを取り外して内部のパケットを取り出すことをデカプセル化と称す．入口となる SRv6 ノードでは， H.Encaps が出口側 SRv6 ノードの End.DT4 SID に対応する IPv6 アドレスを宛先として持つ IPv6 ヘッダで，送信するパケットをカプセル化する．該当パケットは SRv6 ルーティングインフラを通じて，End.DT4 SID の `LOC` に沿って出口 SRv6 ノードへと転送される．SID は IPv6 アドレスであるため，通信の途中の経路ではカプセル化された外側の IPv6 ヘッダの宛先アドレスに基づいてルーティングされる．出口 SRv6 ノードがパケットを受信すると，そのノードは End.DT4 を実行する．End.DT4 を実行するとパ

ケットはデカプセル化され，内部パケットは SID の ARG に関連付けられた VRF (Virtual Routing and Forwarding) テーブルに基づいてルーティングされる．

1.4 導入

Service Function Chaining (SFC) は，Software Defined Network (SDN) 及び Network Function Virtualization (NFV) の文脈で研究されているトピックである [3, 4, 5, 6]．SFC では，ネットワーク機能 (NF) を通過する順序や NF のタイプに関する情報を事前に定義し，それらのルールをネットワーク機器に配布する必要がある．SFC ネットワークを構築するネットワーク機器は，事前に決定されたルールに従って受信したパケットを NF に導く．パケットを NF へ導くためのルールは，SDN コントローラやルーティングプロトコルによってネットワーク機器に配布される．ネットワーク機器は IP ルーティング上の最短経路に関係なく，配布された SFC ルールに従ってパケットを転送する次のホップを選択する必要がある．また，パケットのヘッダにこれらのルールに合致させるための特別な情報を埋め込む手法が取られることもある．SFC は，クラウドサービスプロバイダ (CSP)，アプリケーションサービスプロバイダ (ASP) 及びインターネットサービスプロバイダ (ISP) にとって，現在の静的な環境に代わる柔軟かつ経済的な選択肢を提供する [7]．

SFC を実現可能な技術には，いくつかの候補が存在する．例えば OpenFlow [8]，Network Service Header (NSH) [9]，MPLS [10] などである．これらの技術はどれも，最短経路に関係なく，ルールに基づいて受信したパケットを意図した NF に導く，という要件を満たすことができる．OpenFlow では，経路情報を管理する中央のコントローラが，実際にパケットを転送する OpenFlow スイッチに対して明示的にパケット転送ルールを設定する．OpenFlow スイッチは，コントローラによって適切に管理されたルールに従い，パケットを意図した NF に転送する．OpenFlow のもつこのアーキテクチャは，従来のルーティングプロトコルに基づかない柔軟な経路制御を可能にする．NSH は Service Path Identifier (SPI) と Service Index (SI) によって NF を識別する．NSH ノードは，パケットに付与された NSH 内の SPI，SI に基づいてパケットを転送する．NSH は，サービスプレーンと呼ばれる専用のオーバーレイネットワークを作成し，そのオーバレイネットワーク内でサービスを転送する．このオーバレイネットワークを構築する，というアーキテクチャにより，NSH では基礎となるネットワークトポロジを変更することなくサービス転送を可能にする．一方，MPLS では，直接 NSH を使用する代わりに，MPLS ラベルスタックを利用する．このラベルスタックには，パケットが通過すべきノードの順序がホップバイホップで含まれている．ラベルスタック内で表現されるノードはルータだけでなく，NF も含まれるため，そのラベルスタックに基づいてパケットを転送する事で SFC を実現できる．このアプローチもまた，基礎となるネットワークトポロジを変更せずに SFC を実現するために必要な，最短経路によらないパケット転送を達成する．

Segment Routing (SR)，特に Segment Routing over IPv6 (SRv6) もまた，SFC を実装するために使用される技術の 1 つである．SR では，リンク，ノード，サービスといったネットワーク内の各エンティティを**セグメント**として表現する．SRv6 パケットのヘッダ

(SRH) には，セグメントリストと呼ばれる，そのパケットが通過すべきセグメントの順序を示したリストが含まれている．SRv6 では，セグメントを識別するための ID (SID) として，IPv6 アドレスを使用する．言い換えれば，SRv6 は IPv6 ルーティングインフラをその基盤として利用し，SRH 内で定義された順序に従って，任意のセグメントを経由してパケットを転送する．SRv6 は，NF が実行されるノードをセグメントとして表現し，SID を割り当て，任意の順序で NF を通過するようにパケットを転送することで SFC を実現する．

SRv6 では，NF を SID で表し，セグメントリストに基づいて適切にパケットを転送をすることで，SRv6 を基盤とした SFC ネットワークを実現できる．しかし，SRv6 レイヤよりも上位にある NF の振る舞いと，基盤となる IPv6 ルーティングインフラをどのように統合するかは明確でない．例えば，IPv4 パケットの Network Address Translation (NAT) を SRv6 ネットワーク内の NF として考慮する場合を考える．SRv6 ネットワーク内において，IPv4 パケットは，SR Header (SRH) を含む外部 IPv6 ヘッダでカプセル化される．NF で動作する NAT の実装が SRv6 に対応していない場合，SR プロキシ [11] が必要となり，ネットワーク構成や運用における複雑さが増加してしまう [12]．実装が内部パケットへの NAT と SRv6 に則した転送動作を同時に実行できる場合，それはレイヤバイオレーションとなる．Linux には，SERA [13] という iptables を拡張したファイアウォールアプリケーションが存在する．SERA は SRH でカプセル化されたパケットについて，カプセル化された内部パケットのヘッダ情報にマッチする iptables のフィルタールールを適用できる．SRv6 での基本的な転送動作として，End と呼ばれる動作がある．SERA は iptables を拡張することで，この End 動作を処理する機能も実装されている．ただし，既に Linux カーネルには IPv6 ルーティング，及び SRv6 End 動作に関する処理が実装されている．SERA は，Linux kernel に実装されている SRv6 機能を使わずに，独自に改良した iptables アクションによって End 動作を処理する．つまり，SERA は Linux kernel 内で統合されている IPv6 ルーティングインフラと SRv6 処理機能を使わずに，独自に拡張した iptables によって SRv6 とフィルタリングサービスとしての NF を統合している．

本論文では，既存の netfilter ベースアプリケーションの実装を変更することなく SRv6 対応 NF として扱えるようにする，End.AN.NF 提案する．End.AN.NF は Linux netfilter を NF として扱えるようにしつつ，Linux に実装されている IPv6 ルーティングインフラを活用する．End.AN.NF は受信した SRv6 内部パケットに対して，netfilter のフックポイントを透過的するように設計されている．本論文では End.AN.NF を Linux カーネル上で実装し，スループットとレイテンシを評価した．評価の結果，End.AN.NF は End.DT 4 と H.Encaps の組み合わせによる SRv6 内部パケットへの netfilter 適用と比較して 27% 高いスループットと 3.0 マイクロ秒低いレイテンシを実現した．さらに，End.AN.NF のレイテンシは，End.DT4 と H.Encaps の組み合わせよりも 3.0 マイクロ秒低い．また，End.AN.NF のレイテンシはマイクロ秒解像度で End 動作と同じである．

1.5 本論文の目的と構成

本論文における以降の構成は次の通りである．1 章では，本論文の構成，及び本論文の概要を述べる．2 章では，サービスファンクションチェイニングに関する前提知識，及びそれを実現する技術について解説し，本論文の概要について述べる．3 章では，本論文の提案する新たな SRv6 End behavior である End.AN.NF についての詳細な動作，及び実装について述べる．4 章では，実装した End.AN.NF について，レイテンシ及びスループットの性能を特定のを変化させながら性能の計測する．5 章では，本研究における結論と今後の展望について述べ，End.AN.NF に必要なネットワーク制御プレーンについて検討する．

第2章　先行研究と問題提起

本章では，Service Function Chaining (SFC)，及びそれを実現する技術について解説する．SFC を実現するための技術は複数存在する．本論文では，複数ある技術の中で Segment Routing over IPv6 (SRv6) における SFC を前提としているため，SRv6 の概念やその知識についても解説する．

2.1　SRv6 と NF の統合手法

1.3 章で示した End.DT4 及び H.Encaps は，パケットをカプセル化，及びデカプセル化するビヘイビアである．一方で，いくつかのビヘイビアのもつ機能はパケットのカプセル化，及びデカプセル化に限定されていない．SRv6 では，パケットに適用される NF (Network Function) も SID で表現可能である．NF がトランジットパケットに対して，segleft をデクリメントし，宛先 IPv6 アドレスを次の SID で更新する End ビヘイビアとしての動作をしながらネットワークサービスを適用する場合，その NF は SR-Aware ファンクションと呼ばれる．SERA [13] は，Linux iptables に統合された SR-Aware ファンクションの実装である．SERA は Linux iptables を拡張し，SRH のフィールドと iptables のルールをマッチさせて，ファイアウォール用のフィルタリングルールを適用する．SERA はまた，End ビヘイビアのように，パケットを次の SID に転送する機能も持つ．しかしながら，SERA の採用した iptables を拡張する，というデザインでは SERA に関連する SID を既存のルーティングインフラに統合することは困難である．iptables 内のフィルタリングルールとして利用するための SID の情報は，1.3 章で解説した layer-3 VPN の例とは異なり，既存のルーティングプロトコルを通じて広告することはできない．このように，NF の一形態と基盤となる IPv6 ルーティングインフラをどのように統合するかについては，検討の余地がある．

SR-Aware ファンクションと対照的に，従来の SR-unaware NF を SRv6 ベースの SFC に統合するための様々な方法論が提案されている．SR プロキシ [11] は，SR-unaware NF を SRv6 ネットワークに接続するための重要なコンポーネントである．SR プロキシはローカル SID 宛のパケットを受信し，SRH を持たないインナーパケットを関連する NF に渡した後，NF から返されたパケットに適切な SRH を再度付加し，次の SID にパケットを転送する．Linux における SR プロキシの実装もいくつか提案されている [14, 15, 16]．しかし，SR プロキシは根本的にネットワークにさらなる複雑さをもたらす．例えば，SR プロキシは NF から返されるパケットに付加する適切な SID リストを決定する必要がある．内部パケットは任意の宛先と送信元を持つ可能性があり，そのため SR プロキシが付

けるべき SID リストは内部パケットによって異なる可能性がある．SR プロキシは，適切な SID リストを決定するための独自のメカニズムを実装する必要がある．例えば，静的な SID リストをアタッチする End.AS か，プロキシの内部で状態をキャッシュする必要がある [14] ．さらに，SR プロキシをデプロイするためにはいくつかの問題が存在する．例えば特定の SR プロキシタイプと共存できないサービスのタイプ，サービスの有効性の検出，SR プロキシの背後のサービスに対する SID 広告の問題など [12] が既にインターネットドラフトとして挙げらている．

[**この段落では End.eBPF について述べる**]

- eBFP とはなにか
  - eBPF を使って NF を作ることができる
  - 最近では LKM に変わって Linux kernel を拡張する方法としての側面に注目が集まっている
  - eBPF は Virtual Machine で動作するものの Linux kernel に依存する機能も多い
- End.eBPF とはなにか
- eBPF に対して，本研究では netfilter を NF に統合することを目的としている
  - ここで Linux kernel に依存する End ビヘイビアを提案することの妥当性を述べる

2.2 問題提起

現在，Linux の持つ SRv6・IPv6 ルーティングインフラストラクチャを活用しながら Linux カーネルに実装されている netfilter という多機能なパケット処理機能 を NF として利用する手法は，確立されていない．現在提案されている手法では，Linux の持つ IPv6 ルーティングインフラストラクチャを活用した SR-Aware NF をシンプルに実現することは難しい．また，Linux カーネルには netfilter という多機能なパケット処理機能が実装されているものの，SRv6 上で netfilter を直接 NF として扱う方法も確立されていない．SRv6 は NF を SID として表すことで SFC を実現可能なアーキテクチャであるものの，SID として表現された SRv6 上のノードとしての NF と，実際のアプリケーションとしての NF を統合するの方法は自明ではない．セクション 1.3 で述べた SR プロキシを利用する方法では，SR プロキシを導入することで生まれるオーバヘッドや運用上の問題が指摘されている．また，SRH でカプセル化されているパケットに対して，パケットをカプセル化したまま netfilter を適用する手法も確立されていない．本論文では，Linux netfilter を SRv6 NF として活用するための手法を提案する．

2.3 Linux netfilter

[Linux netfilter について説明する． netfilter によってできることや hook point の場所やパケットの流れについて説明する．] [なぜ本研究では NF として netfilter を選んだのかについても述べる．]

第3章　設計と実装

本章では，前提となる Linux カーネルにおけるパケットフォワーディング処理，netfilter に関する知識を解説し，本論文の提案手法についての設計と実装について述べる．

3.1　Linux カーネルにおけるパケットフォワーディング

[Linux カーネルにおけるパケットフォワーディングの流れや，その実装について説明する．]

3.2　設計

SR-Aware NF の新しい実装として，Linux のルーティングインフラに netfilter によるパケットのフィルタリングとマングリング機能を統合した SRv6 End ビヘイビアである End.AN.NF を提案する．End.AN.NF は，End behavior of SR-Aware Native function for NetFilter の略である．End.AN.NF の実装は，Linux カーネルの IPv6 ルーティングスタックを活用するように設計されている．End.AN.NF の SID は IPv6 アドレスとして表現され，Linux 上では IPv6 ルーティングテーブルエントリとして扱われる．End.AN.NF を示す SID は，通常の IPv6 経路として既存のルーティングプロトコル，及びその実装を介して他のノードに透過的に広告される．さらに，End.AN.NF では，SRv6 でカプセル化されたインナーパケットに対して，netfilter のルールを適用できる．End.AN.NF は，nftables[17] や iptables [18] を介して設定されたトラフィックに対する選択的なパケット破棄や NAT の適用などを SRH でカプセル化された内部のパケットに対して適用できる．また，nftables や iptables に限らず，その他の netfilter ベースのアプリケーションについても，それらの実装を変更することなく SR-Aware NF として使用可能にする．

Netfilter は，Linux のレイヤ 3 パケット転送フローに 3 つのフックポイントを持ち，異なるタイミングでパケットに netfilter ルールを適用する．図 3.1 は，トランジットパケットに適用される netfilter のフックのフローを示している．受信したあるパケットに対して End.AN.NF が動作する際，そのパケットは 2 段階の netfilter フックが適用される．1 つ目の適用では，SRH を含む外部 IPv6 ヘッダのついたカプセル化されたパケットに対して，その外部 IPv6 ヘッダをターゲットにして実行される．2 つ目の適用では，SRH を含まない，カプセル化された内部パケットの IP ヘッダをターゲットにして実行される．まず，End.AN.NF を実装した Linux ベースの SRv6 ノードが IPv6 パケットを受信すると，そのカーネルは受信したパケットに prerouting フックを適用し，通常通り宛先 IPv6 ア

ドレスの最長プレフィックスマッチングを行う．宛先アドレスがローカルの End.AN.NF の SID である場合，カーネルはパケットを End.AN.NF の実装に渡し，そうでない場合，カーネルは forward フックと postrouting フックを適用しながら，IPv6 パケットを対応するネクストホップに転送する．一方，End.AN.NF は，SRH でカプセル化されたインナーパケットに対して，再度，prerouting フック，forward フック，及び postrouting フックを適用する．End.AN.NF の段階で netfilter が適用されている間，SRH は End.AN.NF によって隠されるので，netfilter は SRH の処理を考慮する必要がない．End.AN.NF が終了すると，外部 IPv6 ヘッダの宛先アドレスは次の SID に置き換えられ，カプセル化されたパケットは Linux の IPv6 パケットフォワーディングプロセスにおける通常の転送パスに戻る．

End.AN.NF は，パケットをマーキングするために SID の `ARG` フィールドを利用する．ARG は SID の下位ビットである [1]．SRv6 の仕様では，ある End ビヘイビアがその End ビヘイビア固有の用途で `ARG` を利用することを許可している．End.AN.NF では，SID の ARG がマークとして Linux カーネル空間におけるパケットバッファに付加される．netfilter ベースのアプリケーションは，パケットバッファ上のマークを照合することで，適用するルールを変更することができる．したがって，オペレータが，単一の End.AN.NF SID しか定義されていない場合であっても，NF は ARG に基づいてトラフィックのルールを調整することが可能である．

アルゴリズム 1 は，End.AN.NF がパケットを netfilter のフックポイントに渡す方法を示した擬似コードである．まず，End.AN.NF は，ARG の長さがこの End.AN.NF SID に指定されている場合，受信したパケットの宛先アドレスから ARG 値を抽出する．抽出された ARG 値は，マークとしてパケットバッファに付加される．次に，End.AN.NF はパケットバッファの先頭を外側の SRH から内側のパケットに切り替え，バッファを netfilter フックに渡す．フックにインストールされたルールが内側のパケットに適用された後，End.AN.NF は，パケットバッファの先頭を内側のパケットから外側の SRH に復元し，パケットを次のプロセスに渡す．この手順は，図 3.1 の赤い長方形で示した 3 つのフックポイント，prerouting，forward，postrouting に対してそれぞれ適用する．

Linux カーネルは，End ビヘイビアを関数の SID を宛先とするルーティングテーブルエントリとして実装している．このメカニズムは seg6local と呼ばれる．End.AN.NF は End ビヘイビアの 1 つであるため，その実装も seg6local を利用している．図 3.2 に示すように，カーネルは他の End ビヘイビアと同様に End.AN.NF を表す SID をルーティングテーブルエントリとして扱っていることが確認できる．ルーティングソフトウェアや iproute2 を用いて SID をルーティングテーブルエントリとして追加すると，従来のルーティングプロトコルを用いてカーネルのルーティングテーブルにインストールされた経路を広告することが可能となる．実際に FRRouting [19] を使用し，カーネル内の End.AN.NF に関連付けられた SID を BGP 経由で他のルータに IPv6 経路として広告できることを確認した．End.AN.NF のアーキテクチャは，ルーティング制御に既存のルーティングプロトコルを使用できるため，既存の NF との互換性が高い．このアーキテクチャは，Linux netfilter を用いた SR-Aware NF の実現方法の一つである．

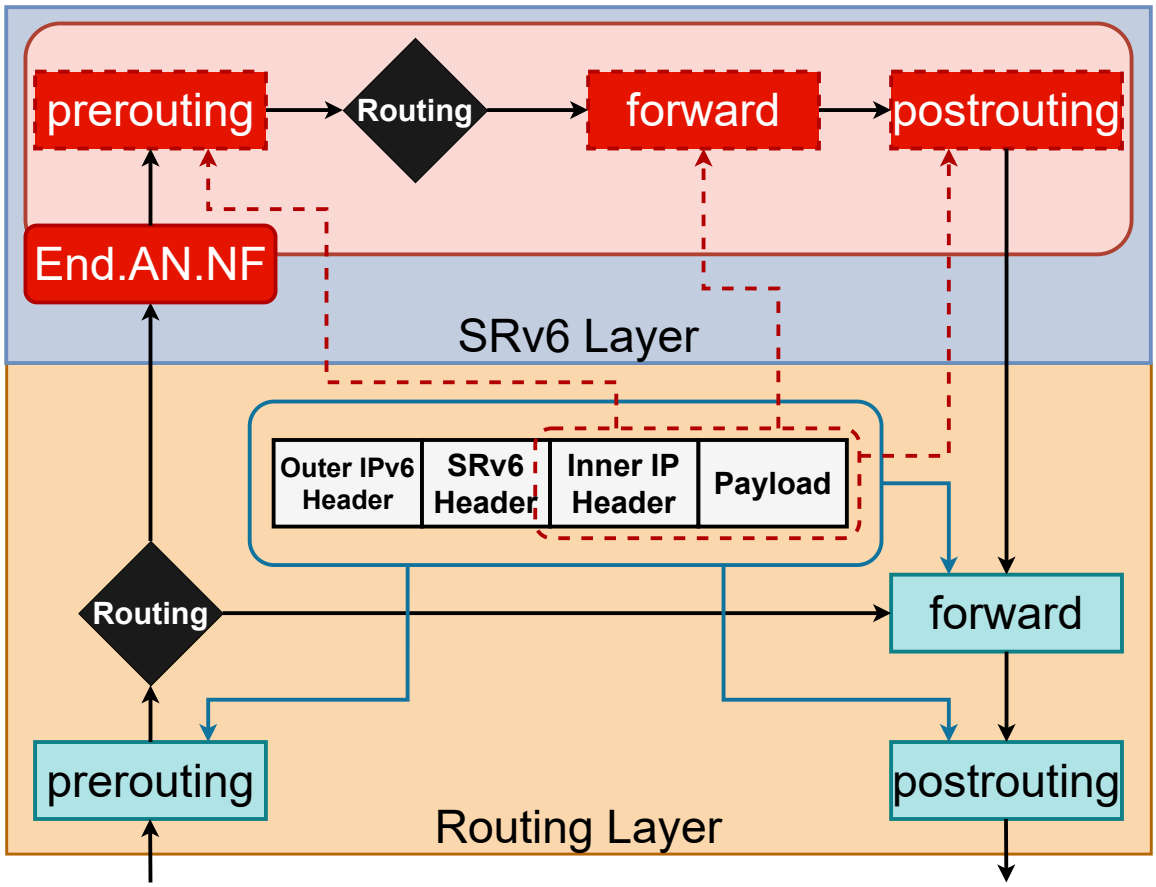

図 3.1: `End.AN.NF` applies three netfilter hook points, prerouting, forward, and postrouting, to inner packets encapsulated in SRv6.

Algorithm 1 Pseudo code of passing a packet to a netfilter hook point in `End.AN.NF`

```
function PassPacketToHook(packet)
    if the length of ARG is specified for this End.AN.NF SID then
        Extract the ARG value from the destination address of outer SRH
        Mark the ARG value on the packet buffer packet
    end if
    Switch the head of packet buffer packet from the outer SRH to the inner packet
    Pass packet to a netfilter hook
    Switch the head of packet buffer packet from the inner packet to the outer SRH
end function
```

```
$ ip -6 route | grep End
2001:db8:1::/96  encap seg6local action End.AN.NF arglen 32 dev eth0 metric 1024 pref medium
2001:db8:2::200  encap seg6local action End dev eth0 metric 1024 pref medium
2001:db8:3::300  encap seg6local action End.DX4 nh4 192.168.99.1 dev eth1 metric 1024 pref medium
```

図 3.2: The modified Linux kernel treats an `End.AN.NF` SID as an IPv6 routing table entry. We can manage the `End.AN.NF` routes with the existing tools such as iproute2.

第4章　評価

我々が実装した End.AN.NF の性能を評価するために，3 つの実験を行った．本章では，3 つの計測実験で得られた結果から，提案手法が実用上十分なスループット性能を持っているか，及び実用的なレイテンシに収まっているのかを確認するために Linux に実装されている既存のパケット転送メカニズムと比較し評価する．このうち 2 つはスループットについて，もう 1 つはレイテンシについて焦点を当てたものである．最初の実験では，パケットサイズに基づくスループットを計測し，2 つ目の実験では，netfilter ベースのアプリケーションにおけるフィルタールール数を増加させた際のスループットの変化を評価した．3 つ目の実験では，異なるパケット転送メカニズムに関連するレイテンシを計測した．また，計測用パケットの送信に利用したトラフィックジェネレータ，及び評価の際に考慮したレシーブサイドスケジューリングについても解説する．

4.1　計測の概要と予想

End.AN.NF の性能を，3 つの転送メカニズムと比較する．比較対象は，End，End.DT4 と H.Encaps の組み合わせ，及び IPv4 である．IPv4 は Linux のパケットフォワーディング性能におけるベースラインとして参照する．図 3.1 に示すように，End.AN.NF が動作する場合，受信パケットは End と比較して 2 倍の数のフックポイントを通過する．したがって，End.AN.NF の性能は End に劣ることが予想される．一方で，End.AN.NF の性能は End.DT4 と H.Encaps の組み合わせよりも高いと予想される．SRv6 でカプセル化されたパケットに netfilter のルールを適用する場合，バニラ Linux カーネルでの実用的なアプローチは End.DT4 と H.Encaps の組み合わせである．しかし，バニラ Linux カーネルには SRv6 でカプセル化された状態の内部パケットに対して netfilter を適用する手法が実装されていない．よって，パケットを一度カプセル化解除し，再度カプセル化する必要がある．End.DT4 によってデカプセル化されたパケットは IPv4 パケットとして netfilter のフックポイントを通過し，H.Encaps は新しい SRH を格納する．この方法では End.DT4 がパケットをデカプセル化し，その後 H.Encaps がパケットを再度カプセル化するため，オーバーヘッドが発生する．したがって，このオーバーヘッドが性能の低下につながることが予想される．

3 つの実験はすべて同じ構成，同じ環境で行った．100Gbps のリンクで直結された 2 台のマシンを用意した．2 台のマシンは同一仕様で，CPU には Intel(R) Xeon(R) Silver 4310 12 コア x2，メモリは 64GB DDR4-2666，NIC には Intel E810 100Gbps を搭載している．CPU のハイパースレッディング機能は無効に設定した．1 台はトラフィック・ジェネレー

タとして，もう 1 台は SUT (System Under Test) として使用する．トラフィック生成マシンには Ubuntu 22.04 と TRex [20] をインストールし，テストトラフィックの生成に使用した．一方，SUT マシンにはカスタマイズした Linux カーネル 5.15.106 をインストールし，End.AN.NF と，End.AN.NF の SID を設定するために独自に拡張した iproute2 コマンドを実装した．また，2 台のマシン間のリンクには 2 つの VLAN を設定し，テストトラフィックを送信するためのリンクと End.AN.NF 動作後に送信されるトラフィックが論理的に別のリンクになるようにした．

4.2 TRex

[**TRex について解説する**]

4.3 レシーブサイドスケジューリング

[**RSS について解説する**]

4.4 パケットサイズ毎のスループット性能

End.AN.NF，End，IPv4，及び End.DT4 と H.Encaps の組み合わせについて，パケットサイズを増加させながらスループットを測定した．この実験により，各パケット転送メカニズムにおけるパケットサイズによるスループットの変化が明らかになった．この実験では，netfilter のルールは使用しなかった．netfilter のフックポイントは通過するものの，実際に適用されるルールを何も定義せずに計測を行った．End.AN.NF の，End に対するスループットの低下，及び End.DT4 と H.Encaps の組み合わせに対する性能の向上を評価した．

4.4.1 計測内容

トラフィック生成マシンで TRex によって生成されたトラフィックを，最小パケット長 126 バイトから最大パケット長 1518 バイトまでパケットサイズを変化させ，SUT マシンに送信した．測定時のパケット長は次のように計算した: $l = 174n + 126$．ここで l はパケット長，n は測定回数である．$n = 0$ から $n = 10$ まで，合計 10 回の測定を行った．

最小パケット長として 126 バイトを選択した理由は，SID リストの長さが 2 である際のタグ付き VLAN を持つ UDP パケットの最小長が 126 バイトだからである．End.AN.NF は，パケットの segleft をデクリメントするため，SID リスト長は少なくとも 2 である必要がある．これは SID リストの長さが 1 の場合，segleft は 0 から始まり，End.AN.NF でデクリメントすると負の値になってしまうからである．一方，End.DT4 は，segleft が 0 であることを必要とする．End.DT4 は SRv6 ネットワークの終点で SRH をでカプセ

ル化するビヘイビアである．つまり，End.DT4 が動作するのは SID リストによって指定された最後のノードであるため，segleft はそれ以上デクリメントできない 0 である必要がある．そこで，End.DT4 と H.Encaps の組み合わせの測定では，TRex は SID リスト長が 2 のパケットを生成し，segleft を 0 に設定した．また，レシーブサイドスケジューリング (RSS) の仕組みを効果的に使用するため，TRex でパケットを生成する歳に内側の IPv4 パケットの宛先アドレスと送信元アドレスの両方をインクリメントした．IPv4 パケットの計測の際は，SRv6 パケット長に合わせて UDP ペイロードにダミーデータを埋め込み，最小パケット長が 126 バイトから始まるようにした．End.DT4 と H.Encaps の組み合わせの計測と同様，RSS を効果的に活用するため，パケット生成時に宛先アドレスと送信元アドレスをインクリメントした．最大パケット長については，タグ付き VLAN ヘッダを含むイーサフレームの最大サイズが 1518 バイトであることから，今回の測定ではパケットサイズの上限を 1518 バイトに設定した．

4.4.2 評価

図 4.1 に，この実験の結果を示す．End.AN.NF のスループットは，すべてのパケット長において End と比較して 6% 以上の低下は見られない．パケット長が 1518 バイトのとき，End.AN.NF は End と比べた際のスループットの低下が最も少なく，その低下は約 1.7% である．対して，パケット長が 478 バイトのとき，End と比較した際の End.AN.NF のスループットの低下は最も大きく，その低下は 約 5.6% である．パケット長とスループットには相関がなく，大きな変動が見られた．このスループットの低下は，End.AN.NF のパケットが End のパケットに比べて 2 倍の netfilter のフックポイントを通過することが原因として挙げられる．ただし，そのスループット低下のレベルは許容範囲内に留まっている．

End.AN.NF のスループットを End.DT4 と H.Encaps の組み合わせと比較した場合，End.AN.NF は予想通り，パケット長に関係なく一貫して優れた性能を示している．具体的には，End.AN.NF は End.DT4 と H.Encaps の組み合わせよりも，最大で 26.7% 高いスループットを達成している．グラフから，End.AN.NF と End.DT4 と H.Encaps の組み合わせとの間のスループットの差はパケット長の影響を受けていることが読み取れる．短いパケットでは相対的な性能格差が大きくなり，長いパケットではその差は縮まる．パケットサイズが小さくなるにつれて，1 秒あたりのパケット転送レート (pps) は増加する．結果として，パケットサイズが小さいほど，パケット転送のオーバーヘッドが顕著になる．

4.5 netfilter にインストールされたルール毎のスループット

次に，End.AN.NF，IPv4，および End.DT4 と H.Encaps の組み合わせについて，netfilter にインストールするフィルタールールの数を変更しながらスループットを測定した．フィルタールールのインストールには，netfilter ベースのアプリケーションとして nftables を使用した．nftables では，ルールはチェインの集合として表現され，チェインにはベー

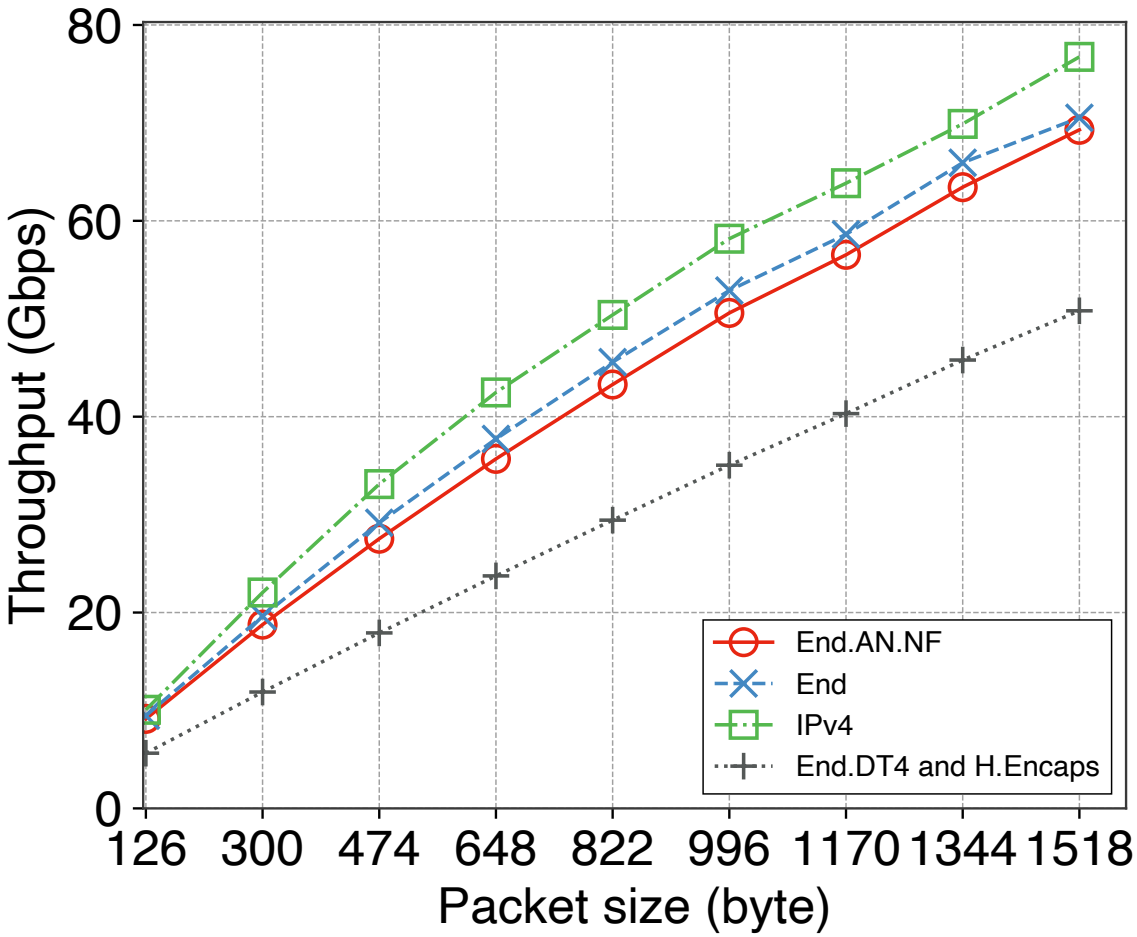

図 4.1: Throughput per SRv6 End behaviors and IPv4

スチェインとレギュラーチェインの 2 種類がある．nftables は，他のチェインがレギュラーチェインを参照している場合のみ，レギュラーチェインを使用する．実験では，チェインの種類ごとにカウントを増やしながらスループットを測定した．パケット転送メカニズムに関わらず，フィルタールールの追加によりスループットが低下することが予想される．この実験は，フィルタールールによる各パケット転送メカニズムのスループット低下の特徴を明らかにすることを目的とする．

4.5.1 nftables

[nftables について解説する． 特に regular chain と base chain について詳しく記す．]

4.5.2 計測内容

トラフィック生成マシンで TRex が生成したトラフィックを SUT マシンに送信した．この測定では，パケット長を一貫して 126 バイトに設定した．パケット長を 126 バイトに設定した理由はセクション 4.4.1 で説明したものと同じで，SID リスト長が 2 の場合のタグ付き VLAN の UDP パケットの最小長が 126 バイトだからである．

4.5.3 評価

図 4.2 は，ベースチェインのルール毎のスループットを示している．全てのチェインルールがベースチェインのみで構成されるこれらのチェインルールは，nftables のチェインルールの定義の中でも最も性能の出ないルール定義の 1 つである．この測定では，netfilter のフォワードフックポイントにフィルタールールを設定した．netfilter は 1 つのフックポイントに複数のルールを設置できる．実験を通して，このフックポイントで適用される同一のカスケードルールの数を増加させた．すべてのパケット転送メカニズムにおいて，スループットはルール数の増加と共に低下する．ルール数が増加するにつれて，3 つのパケット転送メカニズムすべてのスループットは約 0.4 Mbps に収束する．End.AN.NF と IPv4 のスループットを比較すると，スループット低下における顕著な特性の違いは見られず，End.AN.NF は IPv4 に対して大きく劣るスループット低下特性を示さない．一貫して，End.AN.NF は End.DT4 と H.Encaps の組み合わせのスループットを上回る．しかし，このスループットの差はルール数が増えるにつれて縮小し，128 ルールではわずか 9% の差まで減少した．よって，レギュラーチェインのルール数が増加するにつれて，End.AN.NF の End.DT4 と H.Encaps の組み合わせに対する優位性は低下すると言える．

図 4.3 は，ベースチェインのルール数毎のスループットを示している．注目すべき点は，ルール数を増加させてもスループットの低下が認めれず，かつ End.AN.NF が一貫して End.DT4 と H.Encaps の組み合わせを上回っていることである．レギュラーチェインのフィルタールールは，ベースチェインで測定した際と同じ構成である．通過するパケットをすべてアクセプトするルールが定義されており，一度受け入れるルールが適用されたあとも，事前に決めた回数同じ内容のルールが適用され続ける．しかし，レギュラーチェインのみから成るこのようなルール構成では，定義されたレギュラーチェインが他のチェインから参照されていないため，実際にはルールがパケットへ適用されることはない．その結果，パケットが netfilter のフックポイントを通過する際に実際に適用されるルールの数は変わらない．

4.6 レイテンシ

スループットと同様に End.AN.NF，End，IPv4，及び End.DT4 と H.Encaps の組み合わせについて，レイテンシを測定した．この計測では特に，End.DT4 と H.Encaps の組み合わせと比較して，End.AN.NF のレイテンシがどれだけ改善されたのか評価することを目的としている．この評価では，ベースラインとして IPv4 のレイテンシを用いた．

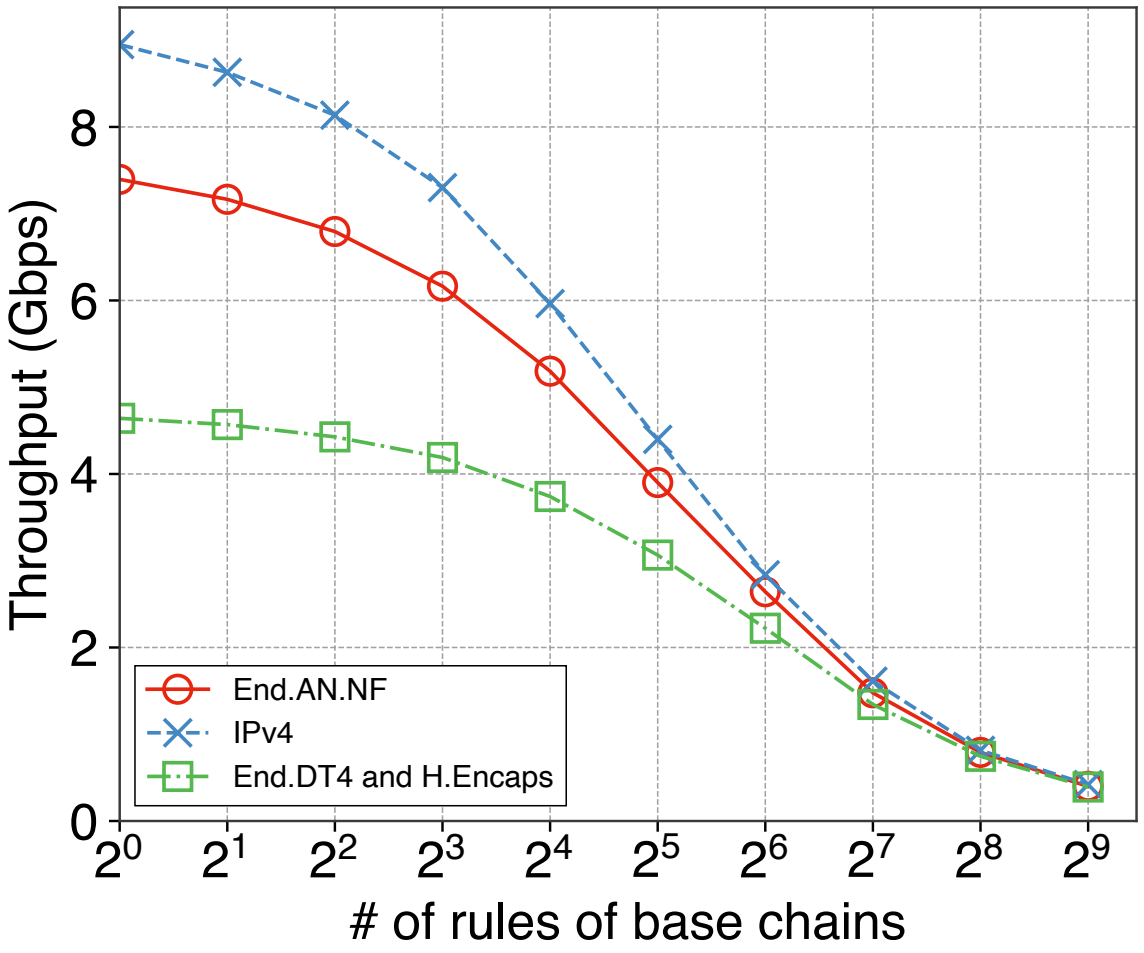

図 4.2: Throughput per number of rules of base chains

4.6.1 計測内容

スループットと同様に，計測には TRex を使用し，パケット転送のレイテンシを測定した．TRex はパケットの送受信時間間隔をマイクロ秒単位で測ることが可能である．今回の測定は，パケット長を 142 バイトに設定した．142 バイトの内訳について，先頭 126 バイトはセクション 4.4.1 で説明した通りで，End.AN.NF がの動作要件を満たす最小のパケットとして必要だからである．追加の 16 バイトは，TRex がよるレイテンシを測定する際に利用するメタデータの埋め込みに使用される．実験中，トラフィック生成マシンは SUT に対して毎秒 10000 個のパケットを 10 秒間送信した．今回のレイテンシ測定では，RSS を無効化するために送信元アドレスと宛先アドレスを変更しなかった．なぜなら，この規模の pps で RSS を使用してパケットを処理する CPU コアを分散させてしまうと，かえってレイテンシが悪化し，余分なジッタが発生することがあるからである．

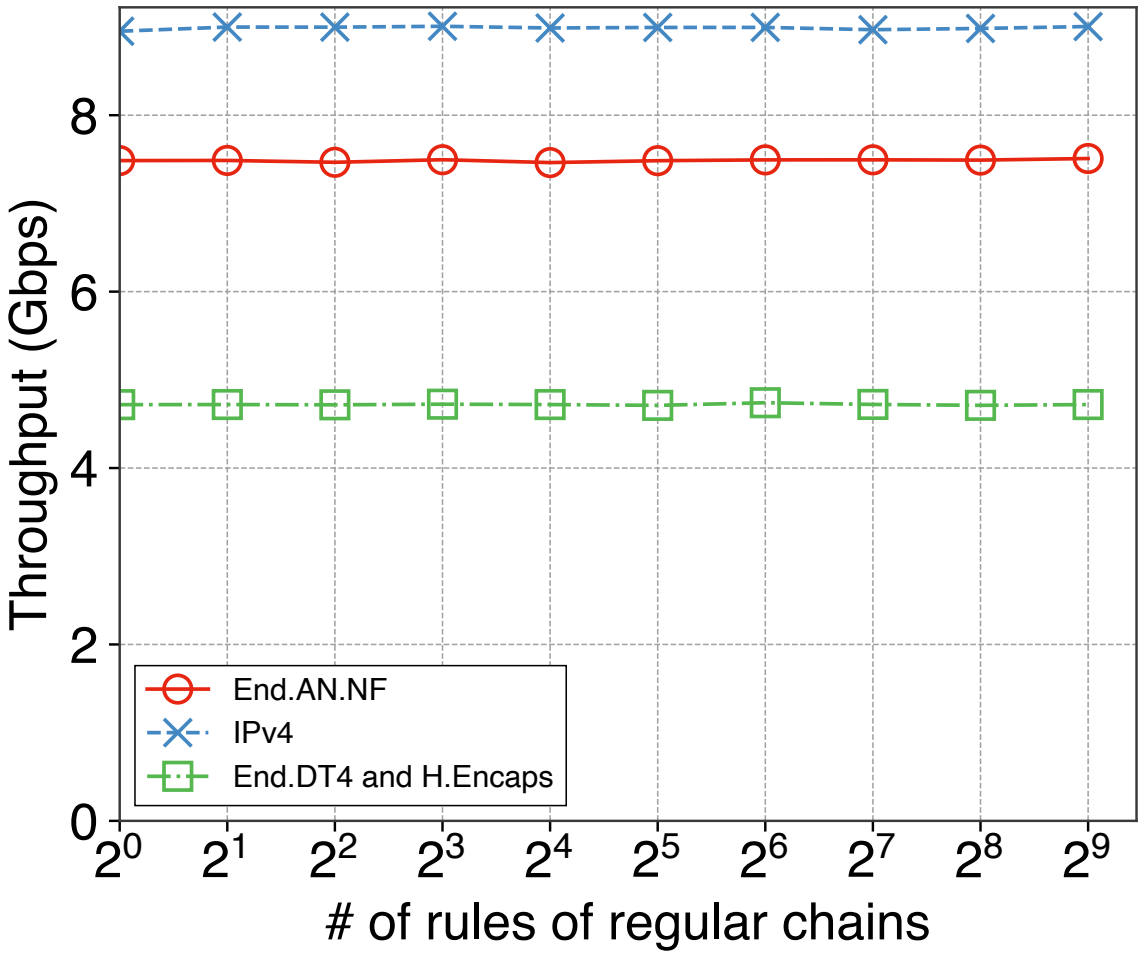

図 4.3: Throughput per number of rules of regular chains

4.6.2 評価

計測結果を図 4.4 に示す．これらのグラフの各データポイントは，100000 回のレイテンシ測定の平均値を表している．End.AN.NF，End，IPv4 のレイテンシはどれも 16.0 マイクロ秒である．対照的に，End.DT4 と H.Encaps の組み合わせは 19.0 マイクロ秒のレイテンシである．マイクロ秒単位での測定では，End.AN.NF のレイテンシは End と IPv4 のレイテンシと一致し，End.DT4 と H.Encaps の組み合わせのレイテンシよりも約 15.8% 高速であることが分かる．

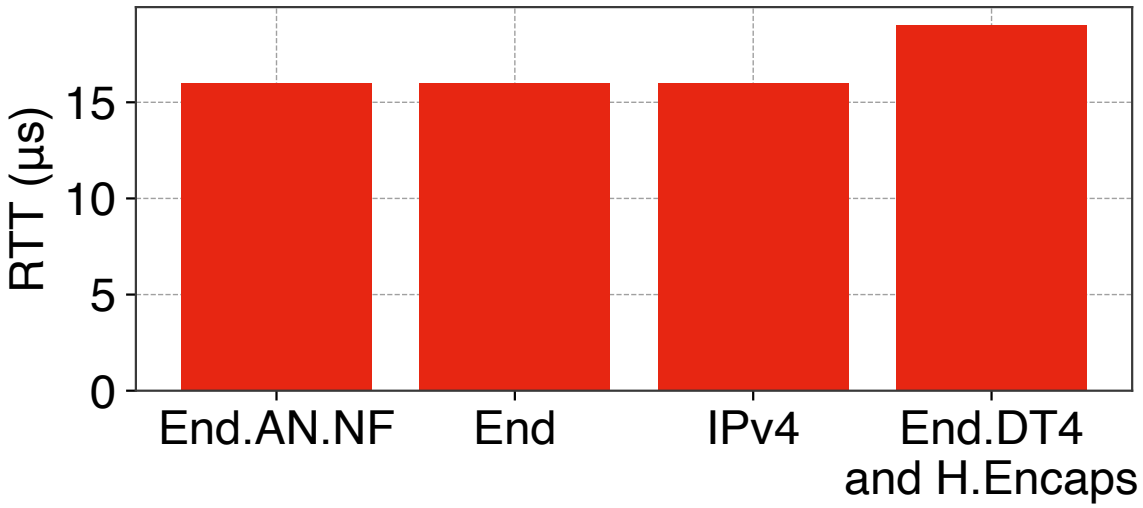

図 4.4: Latency per SRv6 End behaviors and IPv4

第5章　結論

本論文では，Linux netfilter を統合し，BGP などの既存のルーティングプロトコルとの共存を実現する新しい SRv6 End ビヘイビア，End.AN.NF を提案した．End.AN.NF は，SRv6 インナーパケットに対して netfilter の 3 つのフックポイント prerouting，forward，postrouting を透過的に動作させることができる．netfilter のフックポイントを透過する事により，netfilter ベースのアプリケーションはその実装を変更せずに SR-Aware アプリケーションとして機能させることができる．また，End.AN.NF はパケットをマークするために SID の `ARG` フィールドを利用する．このアプローチにより，netfilter ベースのアプリケーションはパケットバッファ上のマークをマッチングさせることによるダイナミックなルール調整が可能となる．我々は End.AN.NF を Linux カーネルに実装し，その性能を評価した．評価の結果，我々の実装は，SRv6 インナーパケットに netfilter のルールを適用する方法である End.DT4 と H.Encaps の組み合わせと比較して，27% 高いスループットと 3.0 マイクロ秒低いレイテンシを達成した．さらに，End と End.AN.NF のスループットの差は 6% 未満であり，End.AN.NF のオーバーヘッドは最も基本的な End の動作と比較して許容範囲内であることを示している．

謝辞

感謝.

参考文献

[1] Clarence Filsfils, Pablo Camarillo, John Leddy, Daniel Voyer, Satoru Matsushima, and Zhenbin Li. Segment Routing over IPv6 (SRv6) Network Programming. RFC 8986, February 2021.

[2] Gaurav Dawra, Ketan Talaulikar, Robert Raszuk, Bruno Decraene, Shunwan Zhuang, and Jorge Rabadan. BGP Overlay Services Based on Segment Routing over IPv6 (SRv6). RFC 9252, July 2022.

[3] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[4] Karamjeet Kaur, Veenu Mangat, and Krishan Kumar. A comprehensive survey of service function chain provisioning approaches in sdn and nfv architecture. *Computer Science Review*, 38:100298, 2020.

[5] Irena Trajkovska, Michail-Alexandros Kourtis, Christos Sakkas, Denis Baudinot, João Silva, Piyush Harsh, George Xylouris, Thomas Michael Bohnert, and Harilaos Koumaras. Sdn-based service function chaining mechanism and service prototype implementation in nfv scenario. *Computer Standards & Interfaces*, 54:247–265, 2017. SI: Standardization SDN&NFV.

[6] Gianluca Davoli, Walter Cerroni, Chiara Contoli, Francesco Foresta, and Franco Callegati. Implementation of service function chaining control plane through openflow. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–4, 2017.

[7] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[8] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, mar 2008.

[9] Paul Quinn, Uri Elzur, and Carlos Pignataro. Network Service Header (NSH). RFC 8300, January 2018.

[10] Adrian Farrel, Stewart Bryant, and John Drake. An MPLS-Based Forwarding Plane for Service Function Chaining. RFC 8595, June 2019.

[11] Francois Clad, Xiaohu Xu, Clarence Filsfils, Daniel Bernier, Cheng Li, Bruno Decraene, Shaowen Ma, Chaitanya Yadlapalli, Wim Henderickx, and Stefano Salsano. Service Programming with Segment Routing. Internet-Draft draft-ietf-spring-sr-service-programming-08, Internet Engineering Task Force, August 2023. Work in Progress.

[12] Ryo Nakamura, Yukito Ueno, and Teppei Kamata. An Experiment of SRv6 Service Chaining at Interop Tokyo 2019 ShowNet. Internet-Draft draft-upa-srv6-service-chaining-exp-00, Internet Engineering Task Force, October 2019. Work in Progress.

[13] Ahmed Abdelsalam, Stefano Salsano, Francois Clad, Pablo Camarillo, and Clarence Filsfils. Sera: Segment routing aware firewall for service function chaining scenarios. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 46–54, 2018.

[14] Marco Haeberle, Benjamin Steinert, Michael Weiss, and Michael Menth. A caching sfc proxy based on ebpf. In *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, pages 171–179, 2022.

[15] Andrea Mayer, Stefano Salsano, Pier Luigi Ventre, Ahmed Abdelsalam, Luca Chiaraviglio, and Clarence Filsfils. An efficient linux kernel implementation of service function chaining for legacy vnfs based on ipv6 segment routing. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 333–341, 2019.

[16] Baosen Zhao, Yifang Qin, Wanghong Yang, Pengfei Fan, and Xu Zhou. Sra: Leveraging af_xdp for programmable network functions with ipv6 segment routing. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 455–462, 2022.

[17] The Netfilter's webmasters. The netfilter.org "nftables" project. Accessed: 2023-09-18.

[18] The Netfilter's webmasters. The netfilter.org "iptables" project. Accessed: 2023-09-18.

[19] FRRouting Project, a Linux Foundation Collaborative Project. Frrouting. Accessed: 2023-08-23.

[20] TRex Team. Trex: Realistic traffic generator. Accessed: 2023-08-25.