

東京大学
情報理工学系研究科 電子情報学専攻
修士論文

ネットワーク機器のパケットレベルでの
動作比較システムの設計と実装

SPLAND: Proposal for Design and Implementation of
the System to Compare Network Devices ' Behavior in Packet Level

東角 比呂志
Hiroshi Tokaku

指導教員 江崎 浩 教授

2016年2月

概要

インターネットをはじめとする情報通信のネットワークは、重要な社会基盤の一つとなり、インフラとして高い信頼性や可用性が求められるようになった。そのため、災害時や機器雇用などの障害発生時であっても、通信品質の劣化やサービスの停止といった事態を起こさない構成が必要である。そこで情報通信ネットワークでは、回線や通信機器などで単一障害点をなくした、冗長構成をとることが一般的である。

冗長化されたネットワークでは、障害時であってもネットワークの稼働率を落とさないために、障害時は冗長系統へと自動で切り替わる。自動で切り替わった機器は、切り替わる以前に稼働していた機器と同じ様に動作をしなければいけない。また、機器の入れ替えやセキュリティ対策のためにネットワーク機器を更新する必要がある。こうした場合にも、更新前と更新後の機器は同じ動作をしなければならない。このように、ネットワークにおいて、異なる複数の機器が同一の動作をすることが重要である。

しかし、実際には同じ動作を行うように設定したはずの機器が異なる動作をする場合がある。ネットワーク機器はベンダー毎に設定の形式が異なり、設定に使用する用語も異なる。また、同一の機器を用いても、そのファームウェアのバージョンが異なれば、異なる挙動を示す場合がある。また、そもそも実装にバグが存在し、挙動がおかしい場合も存在する。このように、同一動作をするように設定したにも関わらず、ネットワーク機器が異なる挙動を示し、障害の原因となることがある。

そこで、本研究では、同一動作をするように設定した異なる機器間の挙動の差を検出することを目的とし、各機器の挙動によって生じる、パケットの差分情報を収集するシステムを提案する。本システムは、2台のネットワーク機器に対して同じパケットを同時に送信し、各機器から送信されてくる出力パケットを比較する。本研究では、各機器に対して同一のパケット列を送信した時に、各機器から送信されてくる出力されてくるパケット列により、ネットワーク機器の挙動を分類する。この分類に従って、機器の挙動の差を検出するシステムの設計・実装を行った。その上で、提案するシステムが、異なるネットワーク機器の挙動の差を検出可能であることを示すために、ルータの誤転送及びファイアウォールによるトラフィックの誤制御の2つのシナリオに対して、本システムの実験を行い、評価した。この評価によって、本システムはインターネットの一般的なMTUサイズである1500バイトパケットにおいて、約8 Gbpsの速度で検証可能なことを示した。そのため、本システムは十分に実用的なものであるといえる。

目次

第 1 章	序論	1
1.1	背景	1
1.2	本論文の目的	2
1.3	本論文の構成	2
第 2 章	ネットワークの運用	3
2.1	ネットワークの冗長化	3
2.2	ネットワーク機器の更新による変更	5
2.3	ネットワーク機器における挙動の差	6
第 3 章	ネットワーク機器におけるトラブルシューティング	9
3.1	ネットワーク機器の挙動	9
3.2	既存技術	11
3.3	ネットワーク機器の動作比較	13
第 4 章	提案手法	17
4.1	想定環境	17
4.2	要件	17
4.3	システム概要	18
4.4	仮想アドレスを用いた通信	19
4.5	パケットの比較部分	21
第 5 章	実装	25
5.1	実装環境	25
5.2	SPLITTER	25
5.3	ANDER	26
第 6 章	評価	31
6.1	評価方針	31
6.2	ルータによる誤転送の性能評価	31
6.3	ファイアウォールによる誤制御の性能評価	36

iv 目次

6.4	機能評価	39
6.5	考察	40
第 7 章	結論	43
7.1	まとめ	43
7.2	今後の課題	43
	参考文献	45

目次

2.1	ファイアウォール機器の入れ換え	6
3.1	データパス	10
3.2	出力パケット列と機器の挙動分類	15
4.1	SPLAND の概要	19
4.2	仮想アドレスを用いた IP 通信までの流れ	20
4.3	ANDER の概略図	22
4.4	マークパケットと比較部におけるパケットの同期	23
4.5	RSS ライクなバッファへのパケットの振り分け	23
5.1	BinBuffer の構造	27
6.1	実験で使用したトポロジ	32
6.2	各パケットサイズにおける, 入力帯域を変化させた時の, 異なる経路の割合での検知率の変化	33
6.3	異なる経路の割合における, 入力パケットレートを変化させた時の, パケットサイズによる検知率の変化	34
6.4	各パケットサイズにおける, 入力帯域を変化させた時の, 異なる経路の割合による Precision および Recall の変化	35
6.5	各パケットサイズにおける, 入力帯域を変化させた時の, 異なるルールの割合での検知率の変化	37
6.6	異なるパケット制御ルールの割合における, 入力パケットレートを変化させた時の, パケットサイズによる検知率の変化	38
6.7	各パケットサイズにおける, 入力パケットレートを変化させた時の, 異なるルールの割合による Precision と Recall の変化	39

表目次

2.1	ネットワークにおける冗長化	4
3.1	本研究における対象機器と機能	13
5.1	実装環境	25
6.1	ネットワーク機器	32
6.2	実験マシン	32

第1章

序論

本章では、本論文の背景と目的を整理し、最後に本論文の構成を示す。

1.1 背景

スマートフォンが普及し、多くの人々がインターネットに接続可能な端末を持つ時代となった。スマートフォンでは、ウェブやメールといった従来のアプリケーションだけでなく、SNSやストリーミングによる動画、音楽といった様々なアプリケーションがネットワーク通信の利用を前提としている。このように、インターネットはじめとする情報通信網の重要性が年々増加し、データの消失率や配送網全体の稼働率といった観点において、ネットワークには高い信頼性や可用性が求められている。

高い信頼性や可用性が求められるネットワークでは、障害の発生時であっても、通信品質の劣化や、サービスの停止といった事態を避けなければならない。そこで、ネットワークの単一障害点をなくすため、冗長構成をとるのが一般的である。冗長構成では、回線やネットワーク機器、及びその部品といった物理的なものから、通信経路といった論理的なものまで、あらゆるものが冗長化の対象となる。

近年では、運用されるネットワークの規模が急速に拡大しているため、管理する機器の台数が膨大となり、運用管理のコストが増加している。こうした大規模ネットワークにおいて、サービスの稼働率を落とさないために、ネットワークの自動化がなされている。ネットワークを自動化することで、障害時に自動的に冗長システムに切り替わることで、サービスを継続的に提供し続けることが可能となる。切り替わった後の機器は、サービスを継続して提供し続けるために、切り替わる前の機器と同じ動作をするように設定されてなければならない。そのため、冗長化された機器を適切に設定し、運用することが重要である。

しかし、実際には同一動作を行うように設定したはずの機器が異なる動作をする場合がある。その原因は、設定のミスやファームウェア、実装の差異など多岐にわたる。ネットワーク機器は、ベンダーによって、設定形式や設定が異なる。そのため、運用者は機器を適切設定するために、各ベンダ毎の設定方法を、熟知していなければならない。また、同一ベンダの場合でも、ファームウェアのバージョンによっては、デフォルトの設定値が異なるために、同一の設定を

用いていても、機器の挙動に違いが発生する場合がある。それに加え、ファームウェアのバージョンが異なると、ソフトウェアの実装の変化により、挙動が変化することもある。こうした多くの要因により、ネットワーク機器は異なる挙動を示す。

こうした差異は、高可用性、そして冗長構成の観点から、事前の検証段階で発見されなければならない。そこで本研究では、同一動作をするように設定したはずの機器間の挙動の差を検出することを目的とし、各機器の挙動によって生じる、パケットの差分情報を収集するシステムを提案する。

1.2 本論文の目的

本研究では、同一動作するように設定されたネットワーク機器の挙動の差を検出することを目的とする。そのために、2台の機器に対して、同時に同一のパケット列を送信し、各機器から送信されてくる出力パケット列により、ネットワーク機器の挙動の差を分類する。本研究では、出力パケット列を5通りに分類して、ネットワーク機器の挙動の差を定義した。この定義にもとづいて、機器の挙動の差を検出するシステムの要求事項をまとめ、システムアーキテクチャ及び効率的にパケットの比較を行うためのバッファの設計・実装を行った。その上で、提案するシステムが、異なるネットワーク機器の挙動の差を検出可能であることを示すために、ルータの誤転送やファイアウォールによるトラフィックの誤制御に関する実験・評価した。

1.3 本論文の構成

本論文の構成は以下の通りである。まず、第2章では、ネットワーク運用において、異なる機器が同一の動作を果たすことの重要性を述べる。第3章では、ネットワーク機器の挙動の差を生む原因について説明したのち、異なる機器の動作比較を行うための要求を整理する。第4章では、本研究で提案するシステムの要件を述べ、その設計について示す。第5章では、本研究で提案するシステムの実装について述べる。第6章では、本研究の想定環境と実験内容を述べ、提案するシステムの評価を示す。最後に、第7章では、本研究のまとめと今後の課題を示す。

第2章

ネットワークの運用

情報通信ネットワークは現代社会において重要なインフラとなった。そのため、現在のインターネットを始めとする情報通信のネットワークは、データの消失率や配送網全体の稼働率といった観点において、高い信頼性や可用性が求められている。このような高い通信品質や可用性を持ったネットワークを構築するには、機器を冗長化し、適切に運用していく必要がある。本章では、まずこうしたネットワークに求められる冗長化について説明する。そして、ネットワーク機器運用における、機器更新の必要性と、その際に重要な異なる機器間での挙動の差異について説明する。

2.1 ネットワークの冗長化

スマートフォンが普及し、多くの人々がインターネットに接続可能な端末を持つ時代となった。スマートフォンでは、従来のウェブやメールといったアプリケーションだけでなく、動画や音楽もインターネット上のサーバからオンデマンドで配信されるストリーミング形式のものも利用されている。こうしたアプリケーションの多くは、インターネットに接続されることを前提としている。このように、ネットワークの重要性は日に日に増している。それにより、インターネットをはじめとする情報通信ネットワークは、重要な社会基盤の1つになった。その結果、ネットワークの障害や停止は大きな損失を生み出すことに繋がる。そのため、現在のネットワークには高い信頼性や可用性が求められている。

信頼性や可用性の高いネットワークとは、通信品質の良さやサービスの稼働率が高いことなどで測ることが出来る。通信中のパケットが落ちると、再送などによってデータの転送に時間を要したり、最悪の場合には疎通性がなくなることがある。通信中のパケットはネットワークの経路の問題で落ちることもあれば、回線の混雑、ネットワーク機器の不調などでも落ちることがある。回線の混雑による場合は、時間が立てば解消することもある。しかし、ネットワーク機器の不具合や慢性的な帯域不足の場合には、ネットワークの帯域幅を増大するなどの根本的な対策が必要である。こうした設備の不具合や故障の際に、ネットワーク全体が停止するようないかなることがないように、単一障害点をなくすように構築することが求められる。

単一障害点をなくし、ネットワークの信頼性や可用性を向上させるためによく利用されるの

4 第2章 ネットワークの運用

が冗長構成である。冗長構成とは、システムの一部を多重化することにより、単一障害点をなくしシステム全体の可用性を上げる構成のことである。冗長化はネットワークにおいて、広範囲で使用されており、設備だけでなく経路の冗長化なども存在する。ネットワークにおける冗長化には、障害時の切り替え時間や、ハードウェアコストによって表 2.1 のように分けられる。

表 2.1: ネットワークにおける冗長化

		切り替え方法	ハードウェアコスト	事前設定
回線	L2	自動	低い	要
	L3	自動	低い	要
部品		人手	中	不要
機器	ホットスタンバイ方式	自動	高い	要
	コールドスタンバイ方式	人手	中	不要

回線の冗長化

回線の冗長化はレイヤ 2 に関するものとレイヤ 3 に関するものに分けられる。

レイヤ 2 (L2)

レイヤ 2 における回線の冗長化には、スパンニングツリープロトコル (STP) やリンクアグリゲーションなどがある。STP はループ構成のネットワークの一部を通常時は無効化しておき、障害発生時にその回線で使用可能なものを有効化して利用するものである。リンクアグリゲーションは複数のインターフェースを束ねて 1 つの論理リンクとして利用する方法であり、耐障害性を上げるとともに帯域幅を増やすことも可能である。

レイヤ 3 (L3)

レイヤ 3 における回線の冗長化には、企業ネットワークなどでよく用いられるマルチホーミングがある。マルチホーミングとは、複数の外部ネットワークへの接続をもつ方法である。これにより可用性の向上だけでなく、回線の負荷分散も行うことが可能である。

部品の冗長化

ネットワーク機器の一部に障害が発生した場合に、その部分のみを取り替えて使用可能なように部品のみを冗長化する方法である。具体的には、電源や光モジュール、インターフェイス、ケーブルなどがある。

機器の冗長化

機器の冗長化は、同じ機能を果たす機器を予備として用意する方法である。ホットスタンバイ方式では、事前に機器に設定を行い、稼働させておき、障害時には自動で切り替わるような方法である。必要な台数分予備の機器が必要となる上に、これは常時稼働のため電気代などのランニングコストが発生する。一方コールドスタンバイ方式では、故障

時に電源を初めていれ、設定を行う。そのため、切り替えに時間がかかるが、コストとしてはホットスタンバイ方式に比べると低い。

近年ネットワークの規模が拡大し、管理する機器の数が膨大となり、機器の管理運用が複雑化している。こうした大規模なネットワークを稼働率を低下させることなく運用するために、自動で冗長回線へと切り替わる手段が広く用いられている。自動で切り替わったネットワーク機器は、以前と同じ機能を果たさなければならない。そのため、機器の冗長化をするにあたり、異なる複数の機器が同一の機能を果たすように設定することが重要である。

2.2 ネットワーク機器の更新による変更

本節では、ネットワーク機器の運用において、異なる複数の機器が同一の動作をしなければならない理由を、2つの事例を通して示す。

2.2.1 ネットワーク設備の更新

長期間ネットワークを運用していると、機器の入れ替え作業などが発生することがある。ネットワーク機器の入れ換えを行う理由には、様々なものがある。2.1節でも述べた機器の不調や故障などの場合には、すぐさま部品の交換や機器の入れ換えを行わなければならない。一方、帯域不足や設備投資による機器の更新の場合には、将来のインフラ計画にあった機器を選定する。このような場合には、機器の入れ換えを行う前に、十分に動作検証が行われなければならない。

しかし、十分な動作検証が行われていても、人的ミスなどにより異なる設定がなされている場合がある。例えば、ファイアウォール機器の入れ換えを行う場合を考える。図 2.1 の様なルールの誤りが存在する場合には、本来、防がなければならない攻撃トラフィックを誤って通過させ、セキュリティインシデントが発生する可能性がある。また、逆に入れ替え以前は通過させていたサービストラフィックを破棄するように誤設定した場合には、サービスにアクセスできないなど障害が発生する。このようなことがないように、機器の入れ替えの前後で、同様に動作するように、事前検証が十分に行われなければならない。

2.2.2 ファームウェアの更新

信頼性の高いネットワークを運用していく上で、セキュリティは重要な事項である。毎年数多くの脆弱性が脆弱性情報データベース Common Vulnerabilities and Exposures(CVE) に登録されている。ソフトウェアの脆弱性を悪用され、サービスの継続が不可能となったり、攻撃に加担することもある。こうしたソフトウェアの脆弱性は、サーバだけでなく、ネットワーク機器も影響を受けることがある。例えば Juniper Networks が提供している Junos[1] のあるファームウェア・バージョンでは、IPv6 の特定の packets を受け取ることで CPU 資源の消費によりサービス運用妨害されるソフトウェアのバグが報告されている [2]。

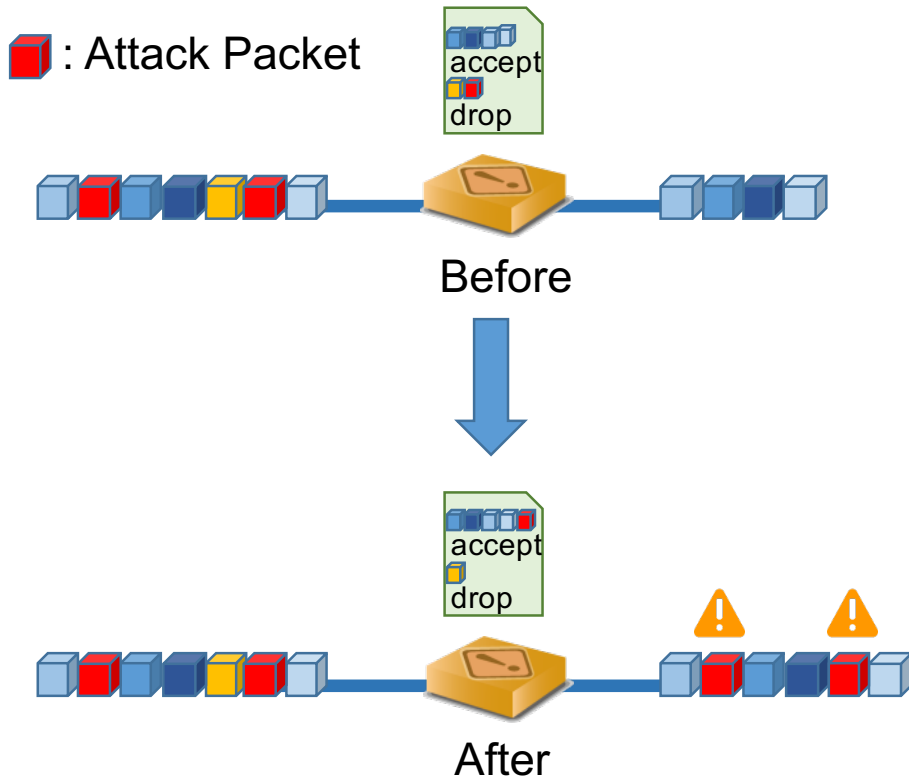


図 2.1: ファイアウォール機器の入れ換え

ネットワーク機器においても、こうしたセキュリティ情報が報告されると、脆弱性に対応したファームウェアがベンダーによって提供される。中には設定を変更するだけで、脆弱性に対応することも可能な場合がある。しかし、セキュリティ・ホールを放置し、運用中の機器が攻撃を受けると大きな被害を被ることになる。そのため、ネットワークを運用していく中で、適宜必要に応じてファームウェアの更新を行わなければならない。

ファームウェアの更新によって、バージョンが変化すると挙動が変化する場合が存在する。これは、同じ設定でもファームウェアのバージョンによってでファオルトの設定値が異なることなどがあるからである。そのため、ファームウェアの更新を実施する場合にも、更新の前後で動作が変化していないかの検証を行わなければならない。

2.3 ネットワーク機器における挙動の差

2.1, 2.2 節で述べたように、ネットワークを運用する上で、機器や設備更新時などに、冗長化の観点から異なる機器が同じ動作をすることは重要である。しかし、運用者が同一の動作をするように設定した複数のネットワーク機器でも、設定ミスや、実装の差、ファームウェアのバグなどにより、実際の機器の挙動が異なる場合が存在する。そこで本節では、このような機器間で発生しうる挙動の差について例を用いて説明する。

2.3.1 ルータによる誤転送

ルーティングとは、各ルータが経路表に従って IP パケットを目的の宛先まで配送することである。ルータは、パケットの宛先 IP アドレスを、保持する経路表内で検索し、最長一致したエントリのネクストホップ宛にパケットを転送する。また、経路表にマッチするエントリが存在しない場合には、パケットは破棄される。

経路表のエントリには静的に追加されるものと、BGP や OSPF などのルーティングプロトコルによって動的に追加されるものが存在する。静的なエントリの場合には、宛先ネットワークもしくはアドレスに対してネクストホップアドレスを指定して設定する。動的なエントリの場合には、各プロトコルに応じて必要な設定を行う。その設定に基づいて、ネットワーク機器間で経路表の交換が行われ、自動で経路表が追加・削除される。どちら方法で追加されても、転送におけるエントリとしての役割は同じである。

同一動作を設定するように設定した機器間で、ルーティングの挙動が異なる場合には、経路表が異なることが原因と考えられる。そのため、ルーティングにおける挙動差は、パケットが異なる宛先へ転送される、もしくは破棄されることで挙動差を確認することが出来る。

2.3.2 ファイヤーウォールによる通信の誤制御

ファイヤーウォールとは、その機器を通過する通信の制御を行う機能である。本論文では、プロトコルタイプと送信元アドレス、宛先アドレス、送信元ポート番号、宛先ポート番号（5 タプル）などの情報を元にして制御を行うものをファイヤーウォールと呼ぶ。ファイヤーウォールには、あらかじめ設定されたルールに従って通信パケットの静的なフィルタを行うものと、機器を通過する向きなどの動的な情報に応じて動作を変更する動的なものが存在する。こうした機能は、意図しない通信を禁止することで攻撃者からネットワークを守るために有効である。

しかし、実際のファイヤーウォールでは、通信を制御するための大量のルールが存在し、各ルールには異なる優先度がそれぞれあるため、ルールの変更は非常に複雑で難しい。そのため、運用者によるルールの設定ミスが発生してしまうことがある。結果として、こうしたルールの設定ミスは、意図しない一部のパケットが破棄されるなどの機器の挙動差として観測される。

2.3.3 パケットのリオーダー

ネットワーク機器には、様々な機能が存在し、複数のオプションが指定可能であることがある。よく使われる機能は、効率的に処理するためにハードウェアによって実装される。しかし、IP 通信におけるソース・ルーティング・オプションなどのあまり使用されない機能は CPU で処理されることが多い。また、アクセス制御リスト (ACL) やファイヤーウォールなどの、複数のルールが設定が可能な機能では、ある一定数まではハードウェアによる処理が行われ、それ以上の数となるとソフトウェア実装で処理されるものも存在する。このような場合には、一定ルール数までは高速で処理が可能であるが、それ以上になると性能が急激に落ちる。このよう

に、ネットワーク機器の機能の種類によって、各機能を実現している方法が異なる。

こうした実装方法の差により、パケットの到着順序の入れ替わり（リオーダー）が発生することがある。つまり、完全に同じ順番でパケットを入力したとしても、出力されてくるパケットの順序が異なることがある。このパケットのリオーダーは、ネットワークを利用するアプリケーションの性能低下の原因となる。リオーダーが発生するものとしては、ARP や IPsec, 非 TCP/UDP のパケットなどの CPU を用いて処理するパケットが多い。こうしたパケットは通常とは異なるデータパスで処理される。そのため、機器ごとに異なる順番となりうる。

このように、実装の差異などに起因して、運用者が同じように設定したつもりでも、実際の挙動が異なる場合がある。この機器間の挙動の差異は、高可用性、そして冗長構成などの観点から、事前の検証段階で発見され、修正されなければならない。そこで本研究では、ネットワークを構築する重要な要素である、ネットワーク機器の動作比較を行うことで、各機器の挙動の差を検出することを目的とする。本研究で対象とするネットワーク機器は、L2 スイッチやルータ、ファイアウォールを、ネットワークの機能としてはパケット制御を行うフォワーディングやルーティング、ACL、ファイアウォールを対象として行う。今後本論文では、ネットワーク機器、機能という単語は、これらを指すものとする。

第3章

ネットワーク機器におけるトラブルシューティング

2章で述べたように、冗長化の観点から、異なる機器が同一動作をすることが重要である。しかし実際には、ベンダーによる実装、ファームウェア、そして設定など、様々な要因によって機器が異なる動作をする場合がある。本章では、まずネットワーク機器の挙動の差を引き起こす要因について述べる。その後、2台の機器の挙動の差を処理されるパケットに従って分類する。そして、最後に既存技術について説明し、それら適用範囲を整理する。

3.1 ネットワーク機器の挙動

2章で述べたように、ネットワークにおいて異なる機器が同一の動作をすることは重要である。しかし、実際のネットワークでは、様々な要因によりルーティングによる誤転送やファイヤウォールによる通信の誤制御などが発生する。本節では、こうした、ネットワーク機器の挙動が異なる要因について説明する。

3.1.1 ベンダー

ネットワーク機器を製造するベンダーは複数存在する。各ベンダーは、異なるハードウェア部品を用いて製品を設計する。そのハードウェアを制御するために独自のファームウェアを搭載するのが一般的である。こうしたファームウェアには、プロトコルのソフトウェアが実装されており、その挙動は異なることがある。そのため、異なるベンダー機器では、相互運用性を欠く場合がある。

また、同一ベンダーによる実装であっても、ファームウェアのバージョンが異なれば、ソフトウェアの実装が異なる場合がある。新規の脆弱性が明らかになると、その脆弱性に対応した新しいバージョンのファームウェアを提供する。こうした新しいファームウェアでは、新しく発見されたソフトウェアのバグが修正される。実装の変更は、機器の動作に大きな影響を与える。このように、ファームウェアのバージョンが異なると、同一製品であったとしても異なる動作

を示すことがある。ファームウェアが原因でおこるソフトウェアの挙動の差は、設定から判断することができず運用者がその原因に気づくことが難しく、障害時には原因の特定に時間を要する。

3.1.2 データパス

2.3.3 節で述べたように、ネットワーク機器の機能は、種類によって実装方法が異なる。各種実装により、パケットは異なるデータパスを通り処理される。これらのデータパスは主にファスト・パスとスロー・パスと呼ばれる2つのデータパスに分けられる [3][4]。ファスト・パスでは、出力ポートが既知のパケットに対してのフォーワーディングを行うなどの比較的な単純な操作をハードウェアで行う。一方、スローパスでは OSPF や BGP といったルーティングプロトコルや ARP などの処理を行う。このようにスローパスでは複雑な処理を CPU を用いて行うことが多い(図 3.1)。

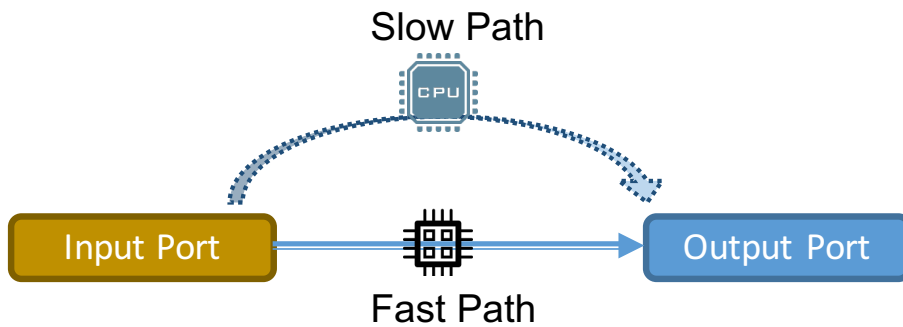


図 3.1: データパス

こうしたデータパスの差は各機器の実装に依存するため、パケットのリオーダが発生する場合がある。また、負荷状態や設定によっては、一部の機能がハードウェアにオフロードされず、ソフトウェアで処理されるような機器も存在する。このように、パケットが処理されるデータパスは、機器間の挙動差を引き起こす重要な要因である。

3.1.3 設定

ネットワーク機器には、その種類や機能に応じて、数多くの設定項目が存在する。こうした、設定はベンダーや機器ファームウェア毎に異なる。使用する機器のファームウェアが異なれば、設定ファイルを流用することは不可能である。また、設定に使用する用語やオプションも異なることもある。そのため、運用者は各ファームウェア毎に設定の形式を熟知し、それらを使い分ける必要がある。

こうした設定には、様々なオプション設定が存在し、非常に複雑である。ファイヤーウォールや ACL の設定などでは、ルールの適用順があり、適切な優先度を設定しなければならない。また、こうした設定は書き方が一通りではないため、新たにルール追加を行う際には、全てのルー

ルを理解し、適切に追加しなければならない。これを誤ると、意図しない通信を禁止したり、逆に許可してしまうなどのミスが発生する。そうした意図しない通信は、セキュリティ上の問題を引き起こす可能性がある。このように正しく機器の設定を行うことはネットワークを運用し、トラブルを引き起こさないために重要である。

また、ファームウェアのバージョンによっては、デフォルトの設定値が異なる場合があるため、同じ設定を用いたとしても異なる挙動を示すことがある。こうしたネットワーク機器の設定は、多くのことを考慮する必要があるため、運用者による人的な設定ミスは避けることは難しい。このように、ネットワーク機器の設定は複雑なため、異なる機器を同一動作するように設定することは、運用者の経験とスキルによるところが大きい。

3.2 既存技術

ネットワークの信頼性や可用性を向上させるため、数多くの研究が行われてきた。本節では、これらネットワークの信頼性や可用性を上げるための既存技術について整理する。

3.2.1 解析によるネットワークの検証

稼働中のネットワークにおいて、エンドツーエンドの接続性を保つことや障害を引き起こさないことは、極めて重要である。しかし、ネットワーク機器の設定は、複雑であるため、運用者による設定ミスが発生することが多い [5]。そのため、機器の設定ファイルを解析することによって、事前にミスを防いだり、障害を検知する手法が多く研究されている。

Margrave[6] や Vantage[7] ツールは運用者に設定変更によって発生する変更範囲の差分などを示す。これにより、運用者の設定変更ミスを無くすことを目的としている。また、Karthickらによる手法 [8] では、パブリック・クラウドなどの大規模ネットワークにおける、疎通性の自動検証方法を提案している。疎通性のポリシーをビット列で表現することで、充足可能性問題 (SAT) ソルバーを用いて、設定の差のセマンティクスを列挙するアルゴリズムを提案している。こうした手法は、解きたい課題に対して、機器の設定をモデル化することによって、既存の SAT ソルバーで設定の検証を可能としている。

こうした設定ファイルの静的解析に基づいた手法は、ネットワーク全体の障害検知や設定ミスの事前発見に大きく役立つ。一方で、設定ファイルの情報から解析するということは、ソフトウェアが正しく動作することを前提としている。そのため、ネットワーク機器のソフトウェアに潜むバグなどの検知などには役立てることは出来ない。

一方で、Haohui ら [9] は設定ファイルではなく、データ・プレーンの情報を静的に解析することでネットワークの検証を行った。ルータの Forwarding Information Base (FIB) を運用者の制約条件と共にモデル化して SAT ソルバーを用いて解くことで、設定に違反している部分を検知する。Peyman ら [10] は幾何学的なモデルを利用することで、ネットワークの静的解析を行った。プロトコルに非依存な形でモデル化することによって、疎通性検証や転送ループなどの障害検知を可能にした。Ahmed ら [11] らは、SDN を用いて、ネットワークにおいて一

定な条件の検証を動的に行った。こうした一定条件の検出は SDN において、動作保証をする上で重要である。このように、解析によって解決可能なネットワークの課題が存在する。

3.2.2 テストパケットを用いたデータ・プレーンの検証

機器の疎通性を確認するツールとして、ping コマンドが一般的に利用されている。ping は、宛先の機器に対して ICMP ECHO メッセージを送信し、機器が ICMP REPLY メッセージを返すところで、その機器へのネットワーク疎通性や、その機器が動作しているかどうかを確認する。また、traceroute コマンドによって経路を調査することが可能である。これにより、パケットが処理される中継機器を特定することができる。このようにテストパケット投げることは、ネットワークの検証において一般的な手法である。実際にパケットを処理させることで、機器の動作確認を行えるためソフトウェアのバグなどについても検知することが出来る。

Hongyi らによる ATPG[12] はルータなどから設定を読み、機器から独立したモデルを構築する。そのモデルに従って、全てのテストパターンを検証可能なパケットを生成する。この手法は、動的にネットワークの異常検知を行えるだけでなく、性能に関する問題が発生している箇所を突き止めることができる。また、Peter らの Monocle[13] は、SDN ネットワークにおけるデータ・プレーンの動的検証手法を提案した。Monocle では、ATPG とは異なり、各データ・プレーンにあるフローテーブル情報から、全てのルール検証を行うための最少のパケットを SAT ソルバーを用いて生成する。SDN のデータプレーンにおける、ルールの優先度の検証を可能とした。

このようにテストパケットを用いたネットワークの検証手法は、多く提案されている。実際にパケットを送信し、機器の振る舞いを観測するため、設定ファイルからでは検知できない種類の課題も解決することができる。

3.2.3 入力テストトラフィックの生成技術

ネットワーク機器の動作を検証する上で、全てのトラフィックパターンに対して、期待通りの動作を示しているかを確認しなければならない。そのため、実際にトラフィックを流して、ネットワーク機器が正しく動作しているかを検証することは非常に重要である。しかしながら、実際のトラフィックは複雑でランダム性を持つ為、その再現率が低い。そのため、不具合が見つかった時にその入力を再現することが困難である。そこで、ネットワーク機器の動作検証において、入力テストトラフィックを生成する技術が存在する。こうしたテストトラフィックを用いることにより、同一の入力トラフィックに対して、動作検証を行うことは再現性の観点から重要である。

パケットキャプチャ技術は、流れているパケットを保存することができる。パケットを記録することにより、ネットワーク機器がどのような振る舞いをしているかをオフラインで解析することが出来る。こうしたパケットキャプチャ技術の代表的なものとして、tcpdump[14] や Wireshark[15] があげられる。これらのパケットキャプチャ技術はスループットの高い環境で

は、パケットロスが大きく、性能面での限界が存在する。しかし、近年の研究成果である高速なパケット I/O ライブラリ [16] [17] [18] はこうしたソフトウェアによるパケット処理の性能を飛躍的に高めた。こうしたパケットライブラリを用いた手法 [19] [20] は、パケットキャプチャの性能を大きく進歩させた。

また同様に、パケットジェネレーション技術は、キャプチャ技術と対なるテストトラフィック生成することより、にネットワーク機器の動作検証をするために、有用である。MoonGen[21] は先述のパケットライブラリを用いたソフトウェアによるパケットジェネレータである。lua スクリプトで発生させるトラフィックを記述できるため、容易に広帯域のテストパケットを送信することが出来る。そのため、機器への入力トラフィックの幅を広げた。このように、ネットワーク機器の動作検証の入力トラフィックを向上させる多くの技術が存在する。しかし、こうしたテストトラフィックを用いる手法では、ネットワーク機器の動的な状態を再現することは難しい。

3.3 ネットワーク機器の動作比較

本研究では、2台のネットワーク機器の動作比較を行うことで、各機器の挙動の差を検出することを目的として、システムの構築を行う。対象のネットワーク機器と機能を表 3.1 に示す。まず、ネットワーク機器同士の動作比較する上で、機器の挙動の差についての定義を行う。その後、ネットワーク機器の動作比較を行うシステムに対する要求を整理する。

表 3.1: 本研究における対象機器と機能

ネットワーク機器	機能
L2 スイッチ	宛先 MAC アドレスに基づく Ethernet フレームの転送
ルータ	宛先 IP アドレスに基づく IP パケットの転送や破棄
ファイアウォール	IP アドレス, トランスポートヘッダ情報の基づく特定フローの破棄

3.3.1 出力パケットに基づく機器の挙動分類

ネットワーク機器の挙動は 3.1 節で述べたような様々な要因により異なる。ネットワーク機器の挙動は、各機器の機能の実現手段である実装によって、異なる挙動を示す場合がある。本研究では、各機器に対して同一のパケット列を送信した時に、各機器から送信されてくる出力パケット列により、ネットワーク機器の挙動を分類する。ネットワーク機器の挙動は次の 3 つによって特徴づけられる。

- パケットの処理時間

ネットワーク機器によって、パケットを処理するための実装が異なるため、パケットの処理時間が異なることがある。パケットの処理時間は、ネットワーク内でのパケットの配送遅延に繋がる。そのため、ネットワークの運用者は、機器のパケット処理時間に対し

て、性能要件として制限を設けていることがある。

- パケットに対する処理

設定が施されたネットワーク機器は、設定内容に従って、パケットに対して転送処理などの処理をおこなう。この処理が異なる場合には、同一の動作をしているとは言えない。これらのアクションが異なる理由としては、3.1 節で述べたように、運用者による設定ミスや実装上のバグなどが考えられる。どちらの場合でも、パケットに対して異なるアクションを行うことは運用者によって意図された挙動とは言えない。

- 出力パケットの順番

ネットワーク機器によって、機能の実装が異なる可能性が存在する。実装が異なると、異なるデータパスを通り処理されることがある。その結果、出力されるパケットの順番が変化することがある。こうしたパケットのリオーダーは、通信品質の低下などを招くことがある。そのため、異なる機器でも、同一の入力パケット列に対して出力パケット列の順番が同じでなければならない。また、リオーダーは見方を変えれば、特定のパケットの処理が他に対する遅れとみなすこともできる。

ネットワーク機器が行う、パケットに対する制御処理は転送するか破棄するかのどちらかである。異なるデータパスを通り、転送される場合には、処理時間が異なるためパケットのリオーダー起こる場合がある。また、転送に際し、MAC アドレスなどのパケットの一部のフィールドが書き換えられて転送される場合も存在する。このような場合には、経路情報の誤りなどによって、異なる書き換えが起こる場合がある。こうした、パケットに対する、ネットワークの挙動によって、機器から送信されてくるパケット列は、図 3.2 のような 6 通りに分類することができ、

図 3.2 において、明らかに同じ動作をしていると判断できるのは、同じパケットが転送されるか、あるいは破棄される場合である。また、どちらかのパケットが破棄されるか、異なるパケットに書き換えられている場合には、同様に明らかに動作が異なると判断できる。しかしながら、パケットのリオーダーが発生している場合や処理遅延が大きい場合には、それがネットワーク機器の同一動作とみなすかどうかは構築するネットワークや機器への要求によって異なる。

3.3.2 機器の挙動の差を検出するシステムへの要求

本研究では、ネットワーク機器の挙動の差を検出するシステム的设计・構築を行う。3.3.1 節では、ネットワーク機器の挙動を特徴づける 3 つの要素について議論した。これを踏まえて、システムに求められる要求を整理する。

- 同一パケット列の入力

3.2.3 節を受けて、ネットワーク機器の各パケットに対する動作を検証するためには、同一のパケット列を各機器に対して入力できなければならない。実際のトラフィックは再現性が低いため、複数台の機器で同一のトラフィックに対して動作検証することは困難である。また、同一パケットであっても、ネットワーク機器のもつ動的な状態を再現し、挙動を比較するためには、パケットの順序や間隔なども同一でなければならない。

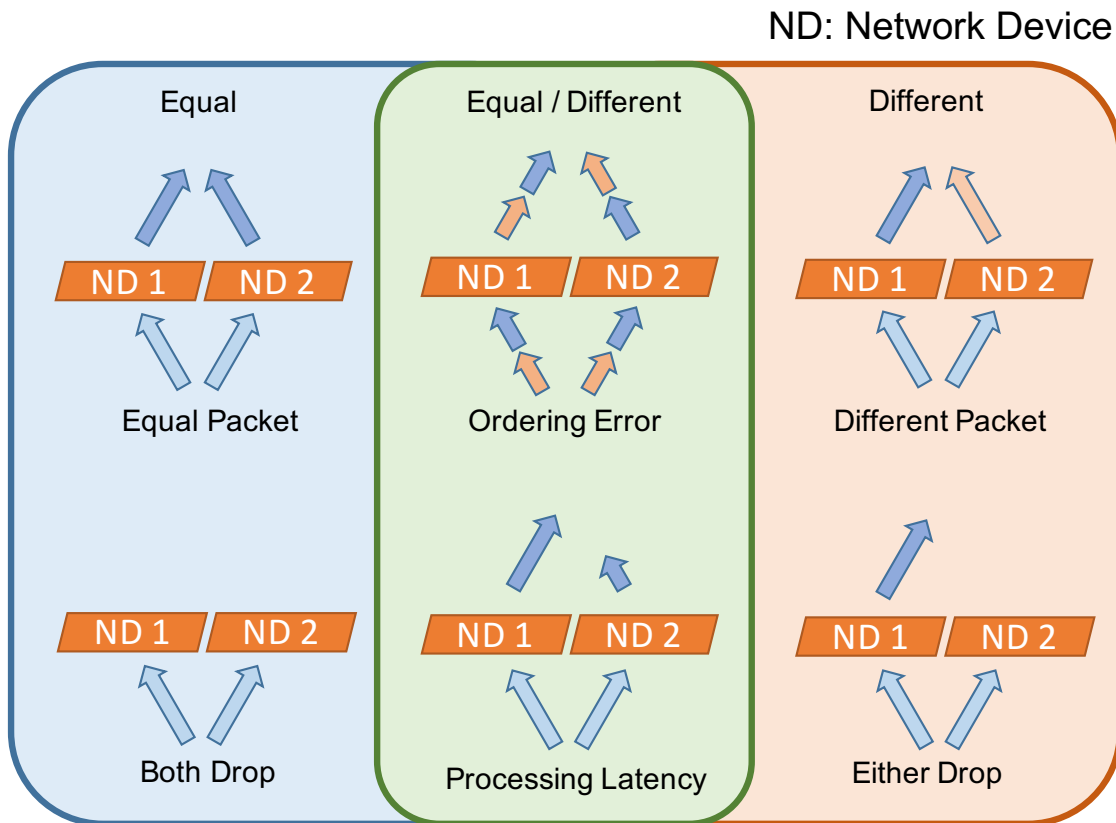


図 3.2: 出力パケット列と機器の挙動分類

- 2 台の機器を同時に動作させ、処理時間の差を計測できること
ネットワーク機器の挙動の差を検証するためには、2 台の機器に対して、同じ入力パケットを同時に与え、動作させることが必要である。2 台の機器のパケットの処理時間の差を検知するためには、同時にパケットを入力して、処理されたパケットを受信できなければならない。また、同一の入力パケットに対して、機器から送信されてきたパケットの同期をとり、処理時間の差を計測できる必要がある。
- 出力パケットの差を検知できること
同一の入力パケット列に対して、各ネットワーク機器からの送信されてくる出力パケットを 3.3.1 節で行った分類で、検知できる必要がある。また、機器の異なる挙動によって発生するパケットの差を検知した場合には、ログとして記録しなければならない。

第 4 章

提案手法

本章ではネットワーク機器の挙動の差を検知するためのシステムを提案する。

4.1 想定環境

本研究では、2 台のネットワーク機器の挙動の差を検出する目的で、機器から送信されてくるパケットを比較するシステムを構築する。このシステムは、同一動作を行うように設定された機器が異なる挙動を示す時に用いるデバッグツールである。そのため、想定される環境として、実運用ネットワークに設置される以前の検証用のネットワークや単体の動作検証時に用いることを想定する。また、対象とする機器のインターフェースの数としては、2 ポートを想定する。これは、ネットワーク機器が機能を果たすのに必要最低限のポート数であるが、ネットワーク機器の機能の動作検証を行ううえでは十分である。次に比較する機器の挙動の差として、異なる挙動の割合が小さいことを想定する。これは検証対象とする機器が、事前に同一動作を行うように設定されているためである。

4.2 要件

本節では、本研究で提案するシステムの機能要件について整理する。本研究では、ネットワーク機器の動作比較を行い、機器の挙動の差を検出することを目的とする。機器の挙動の差である出力パケットの差分を運用者に示し、動作検証の助けとなる情報を提供する。3.3 節での議論を踏まえ、下記に機能要件と性能要件をまとめる。

機能要件

1. 同時かつ同一のパケットの入力

ネットワーク機器の動作比較を同時にするためには、同じパケットを同時に入力する必要がある。そのためには、各ネットワーク機器の対応するポートは、同一のマシンによって管理されなければならない。一台のマシンで各機器の対応するポートを管理することにより、入出力を制御・監視することが出来る。また、同時にパケットを機器に入力することで、各機器による処理時間差を計測することが出来る。

2. 仮想アドレスによる変換

各ネットワークインターフェースは自分の MAC アドレス宛の packets しか受信しない。そのため、ネットワーク機器に packets を送信する際に、イーサネットフレームのアドレスを適切な値に書き換えて送信することが必要である。そのため、本システムでは、各ネットワーク機器の対応するポート組に対して、論理アドレスを割り当てる。検証機器の物理アドレスとこの論理アドレスを本システムでは互いに変換する。

3. ネットワーク機器からの送信されてくる packets の同期

ネットワーク機器の packets の処理時間がそれぞれ異なるため、出力側で packets の同期を取る仕組みが必要がある。また、同期の仕組みがない場合には、周期的に同じ packets が送信されてくるような条件では packets の区別ができない。そのため、比較側では適切な方法で同期をとらなければならない。同期をとるためには、packets をバッファリングする必要がある。このバッファの大きさは対象とする機器の処理時間によって決まるため、設定可能にしなければならない。

4. packets の有効時間

packets のリオーダーは、運用者の要求次第で、ネットワーク機器の同一動作にも異なる動作にもなりうる。そのため、packets を一時的に保存して同じ packets が遅れて到着するかを待たなければならない。その際に、packets の受信時刻からの有効時間を設定し、管理しなければならない。

性能要件

1. 効率的な packets 比較のためのバッファ

ネットワーク機器の運用者が機器に対して求める性能要件によって、バッファする packets の数は変化する。バッファする packets の数は、packets 同士の比較回数に大きく影響を与える。最悪の場合には、packets 同士の比較回数は、 $O(n^2)$ で増加する。そのため、packets のバッファ機構は効率的に比較を行えるように設計しなければならない。

4.3 システム概要

本研究では、4.2 節で示した要件を満たすシステムとして、SPLAND を提案する。SPLAND では、2 台のネットワーク機器に同時に同じ packets を入力するために、リアルタイムで流れる packets を複製し送信する。また、各機器の挙動の差を検出するために、各機器から出力される一連の packets 列を比較する。比較によって、packets 列に違いがあった場合には、機器の挙動の差を生み出す原因究明の手がかりなるよう該当の packets 情報を出力する。

SPLAND は入力側の SPLITTER と比較側の ANDER の 2 つの部分から構成され、動作概要を図 4.1 に示す。SPLAND では検証するネットワーク機器に接続されている 2 つのポート (内部ポート) と検証すべきトラフィックを入出力するための 1 つポート (外部ポート) を管理

する。外部ポートから入るパケットは SPLITTER で複製され、それぞれの内部ポートに転送される。その際、各機器がパケットを破棄しないように、パケットのフィールドを各機器の値に変更する必要がある。

また、ANDER においてパケットの同期を行うために、SPLITTER で定期的にマークパケットを送信する。マークパケットは、事前に検証機器に対向の SPLAND ノードに到達できるように設定を付け加えることで、到達性の保証を行う。ANDER では各ネットワーク機器から送られてきたパケットをバッファに一度蓄えてから、外部ポートに転送する。転送するパケットは事前に設定されたマスター機からのパケットのみである。転送処理を行うとともに、各機器から同一パケットが出力されてきているどうかを検証する。同一のパケットが発見されない場合には、機器の動作差分としてログに出力する。

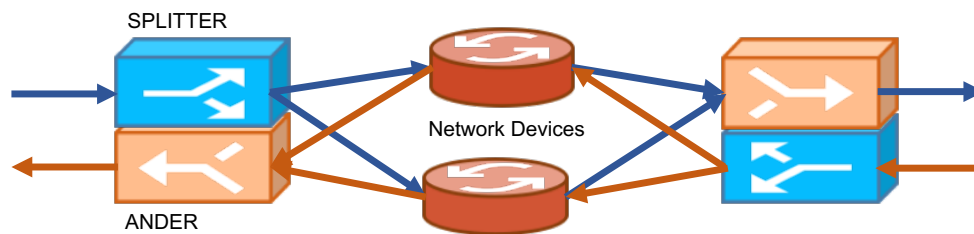


図 4.1: SPLAND の概要

4.4 仮想アドレスを用いた通信

ルータは、イーサネットフレームの宛先 MAC アドレスが自分のインターフェースアドレスでないパケットは破棄する。また、ARP パケットの場合には、パケットのデータ部分のハードウェアアドレスを設定するフィールドが存在する。DHCP パケットの場合には、DHCP リレーアドレスを設定するフィールドが存在する。このようにプロトコル毎に特定のフィールドに、パケットを処理する機器がもつ MAC アドレスが設定されていなければ、きちんと動作しないものがある。そのため、SPLAND では、各機器にパケットを送信する前に各機器のアドレス宛に書きかえる必要がある。そのためプロトコル毎に、特定のフィールドの変換機能の実装を行わなければならない。

このアドレスの変換を行うために、SPLAND では仮想的な MAC アドレスを使う。SPLAND はネットワーク機器に対して透過的に MAC アドレスの変換を行う。そのために、ネットワーク機器と通信するノードに、ネットワーク機器がもつ MAC アドレスをこの論理アドレスとして認識させなければならない。このアドレス変換の流れを、通信のフローと共に説明する (図 4.2)。

1. 通信ノードからの ARP リクエスト

通信ノードから送信された ARP リクエストは SPLITTER で複製され、2 台のルータに送信される。この時、フレームの宛先 MAC アドレスはブロードキャストアドレスで

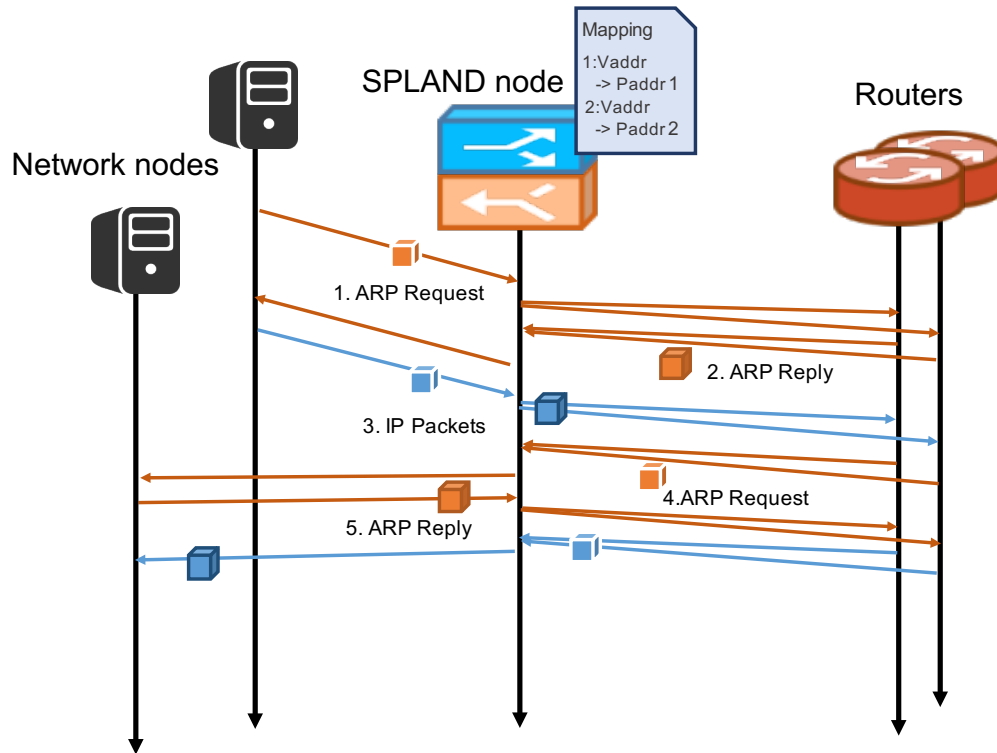


図 4.2: 仮想アドレスを用いた IP 通信までの流れ

あるので、特に変換は行われない。

2. ルータからの ARP リプライ

ルータは自分宛ての ARP リクエストであった場合には、ARP リプライを送信する。この時 ANDER はルータから送られてきた ARP リプライに対してアドレス変換を行う。ARP におけるアドレス変換は、送信元 MAC アドレスと ARP パケットのデータ部分の送信元ハードウェアアドレスを仮想アドレスで書き換える。また、送信されてきたパケットがマスターからのものであれば、そのまま外部ポートに転送する。

3. 通信ノードからの IP パケット

アドレス解決が済んだ通信ノードは、仮想アドレスを宛先 MAC アドレスに設定して、パケットを送信してくる。そこで、SPLITTER は、宛先 MAC アドレスが仮想アドレスになっている全ての IP パケットに対して、パケットを複製する際にルータの MAC アドレスに書き換える。

4. ルータからの ARP リクエスト

ルータが受信した IP パケットの宛先 IP アドレスに対して、ARP テーブルに有効なエントリが存在しない場合、ARP リクエストを送信する。ルータの ARP リクエストパケットには、送信元 MAC アドレスと ARP のデータ部分の送信元ハードウェアアドレスのフィールドに、各機器の MAC アドレスが設定されている。そのため、ANDER でアドレス変換を行い、両方のフィールドを仮想アドレスで書き換える。

5. 通信ノードからの ARP リプライ

通信ノードが送信してくる ARP リプライで、宛先 MAC アドレスが仮想アドレスのものに対してパケットの複製を行うとともに、アドレスの変換を行う。

これらの手順を踏むことにより、2台のルータに同時に同一のパケットを送信しながら、出力パケットの比較を行うことが出来る。また、ANDER ではパケットのアドレス変換を行った後で、パケットの比較を行うことでレイヤ 2 のヘッダに対しても差分を検出できる。

4.5 パケットの比較部分

パケットの比較を行う ANDER では、各機器から送信されてくるパケットを比較し、機器の挙動の差分を見つけることが求められる。2台の機器において処理時間差がある場合には、先に処理が終わる機器のパケットをバッファする必要がある。そのため、ANDER ではパケットの受信部と比較部を別々のプロセスに分割する。パケットを比較する際に、2台の機器の処理時間差に応じて、バッファするパケットの数が線形に増えていく。

一定の速度でパケットを比較し続けるために、比較するパケット数を減らすことが有効である。そのため、比較するパケット数を減らすためには、パケットのグループ分けを行うのが有効である。1段目の比較部では同一のグループ内でのみ一致するパケットが存在するかを調べる。一方、2段目の比較部では、一段目で一致しなかったパケットを全て比較することにする。また、4.2 節で述べた通り、パケットの有効期間を設定することで、2段目においても比較するパケットの数が増えすぎないように調整することを可能としている。

本節では、まずはじめに ANDER の概要を述べる。その後、ネットワーク機器から送信されてくるパケットの同期とパケット比較を効率的に行うためのバッファ機構について述べる。以後、受信部を HASHRX、1段目、2段目の比較部をそれぞれ MERGER、KEEPER、差分情報を保存する部分を Logger とよぶ。また、HASHRX と MERGER の間、MERGER と KEEPER の間それぞれでのパケットの受け渡しに使うバッファを BinBuffer、ExpireBuffer と呼ぶ。

4.5.1 ANDER の概要

ANDER は2台の機器から送信されてくる、パケットを外部ポートに転送するとともに、パケットの差分検出を行う (図 4.3)。HASHRX は一台の検証機器から送信されてくるパケットを受信し続ける。受信の速度が遅いと NIC においてパケットロスが発生させてしまう。そのため、HASHRX では、パケットの比較は行わず、パケットの一部のハッシュ値を計算する。このハッシュ値は比較部において、パケットの比較を効率的に行うためのものである。ここで計算したハッシュ値とパケットをペアにして、比較部に渡すため BinBuffer にパケットを積む。

MERGER では、それぞれ HASHRX から渡されてくるパケットを同一グループ内で比較する。比較を行う上で、逐一パケットを比較するのは効率が良くない。そのため、HASHRX で事前に計算したハッシュ値が一致したパケット同士のみを比較する。もし、一致するパケットが

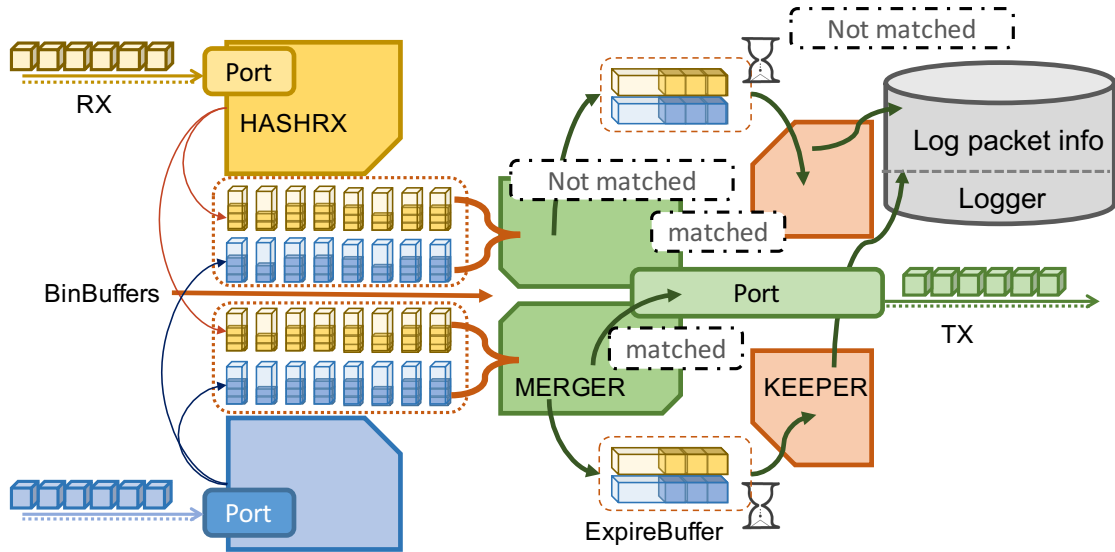


図 4.3: ANDER の概略図

存在しなかったら、次の KEEPER でより多くのパケットと比較する。また、マスター機器から送信されてきたパケットに関しては、通信がとぎれないようにするため、外部ポートに転送処理をする。

次の KEEPER では、パケットの有効期間以内に、同一のパケットが到着するかを検証する。ここに送られて来るパケットは、検証機器の異なる挙動による可能性があるものである。ここでのパケット比較で一致するものが見つかった場合には、マークパケットのロスや、リオーダによって、一段目の比較では誤って渡されてきたパケットである。こうした、パケットは検証機器の挙動として同一の動作として分類される。一方で、見つからなかった場合には、検証機器の挙動の差のてがかりとして Logger にパケットを渡し、ログに記録する。

4.5.2 マークパケット

ANDER において、パケットの比較パフォーマンスを一定に保つのは重要である。検証機器から送信されてくるパケットが、他方の検証機器から送信されてくるパケットに一致するものがあるかの比較は、バッファされているパケットの数を n とすると、 $O(n^2)$ の計算量を必要とする。そのため、比較するパケットの数を減らすのは重要である。また、パケットが一致するかを検証するためには、各機器のパケット処理時間の差を吸収するために、パケットの同期をとる必要がある。これらを解決するために、マークパケットを利用する (図 4.5)。

マークパケットは、SPLITTER から一定間隔で送信することによって、一方のマークパケットに挟まれた範囲のパケットを受信側でグループ分けすることが出来る。これにより、比較するパケットの数を一定に保つことが出来る。そのため、一定の速度比較を行うことが出来る。また、マークパケットに連続した ID をデータとして含めることで、それぞれの機器の同期をとることが可能である。

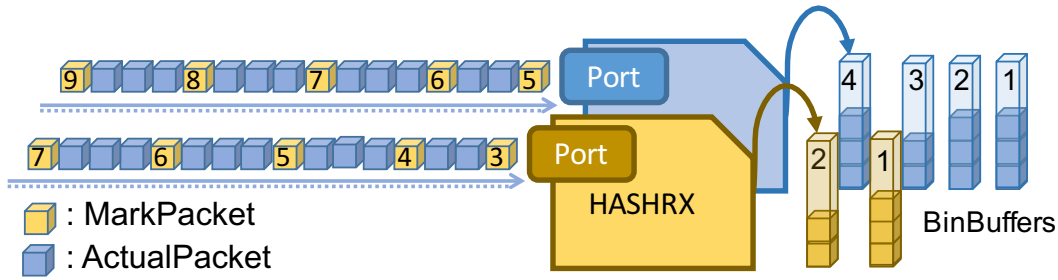


図 4.4: マークパケットと比較部におけるパケットの同期

4.5.3 パケット比較のためのバッファ設計

パケットの比較部では、パケットの比較を一定した速度で行うことが求められる。そのために、比較するパケットの数を減らすことが有効であることは既に説明した。SPLAND では、複数の MERGER で受信されたパケットを並列処理することにより、さらに比較するパケットの数を減らしている (図 4.5)。

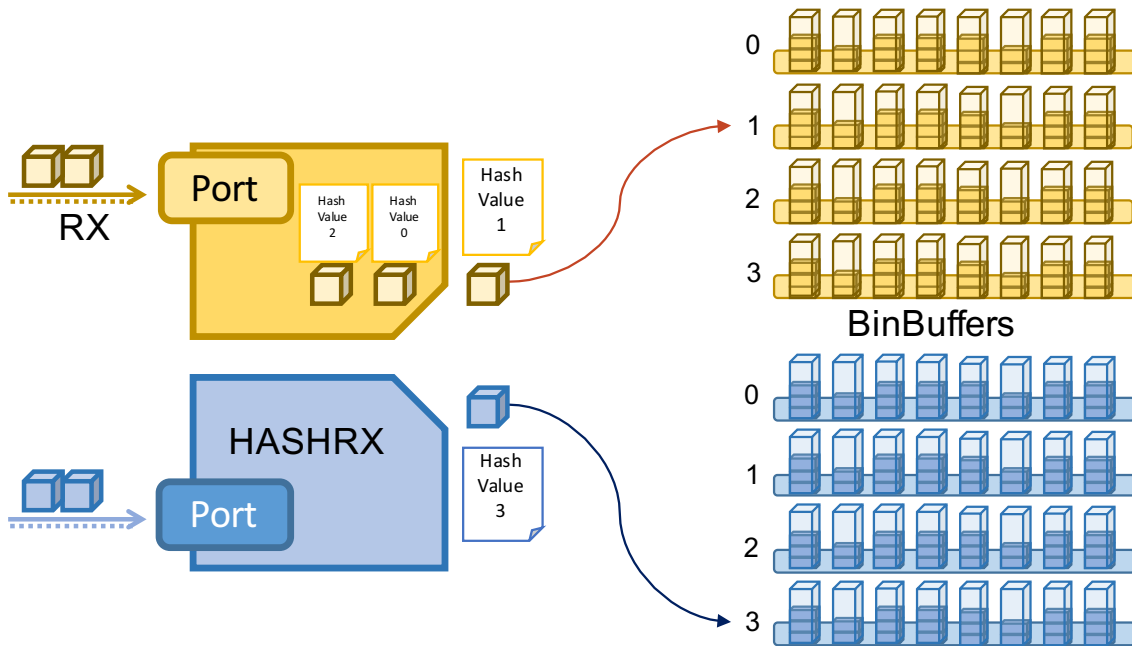


図 4.5: RSS ライクなバッファへのパケットの振り分け

近年の NIC はネットワークの高速化に伴い、マルチキューをサポートしている。そうした、NIC がサポートする機能の 1 つに ReceiveSideScaling (RSS) [22] が存在する。RSS は受信したパケットのフローを識別する、プロトコル番号と送信元 IP アドレス、宛先 IP アドレス、送信元ポート番号、宛先ポート番号 (5 タプル) のハッシュ値に従って、複数の受信キューにパケットを分配する機能である。

HASHRX では、このような RSS の機能を模して、5 タブルのハッシュ値に従って、パケットを積む BinBuffer を決定する。これにより、通信時に同一フローのパケットのリオーダーを生み出さないようにする。BinBuffer は各 MERGER に対して、2 つ用意される。HASHRX 側ではそれぞれ MERGER が持つ、片方の BinBuffer に対してパケットを渡す。これにより、一定時間内に到着するパケットが増加しても、1 台の MERGER で処理するパケットの増加数を抑えることができる。また、処理する MERGER の数の設定を増やすことにより、入力のパケット数が増えても対応することができる。

4.5.4 パケット比較を効率的に行うためのハッシュ値

本システムでは、MERGER および KEEPER で、パケットの比較を何度も行う必要がある。そのため、パケットが同一か異なるかの確認を効率的に行うことが求められる。そこで、パケットからハッシュ値を生成し、パケット全体の比較を行う回数を減らすことにする。ネットワークでよく利用される 5 タブルのハッシュ値は同一フローのパケットには同じ値となるように設定されている。しかし、このハッシュ値はパケットの比較を行う上では、ハッシュ値の衝突する可能性が高いため本システムのパケット比較では用いることは出来ない。そのため、適切なハッシュ値の計算を考えなければならない。

ルータのようなネットワーク機器では、ネクストホップが決まっており、イーサネットのフィールドは同じ値が使われる事が多い。一方、IPv4 ヘッダには、同一フローにおいても、異なる値が使われる識別子番号が存在する [23]。このフィールドを利用することで、パケットのハッシュ値が、その有効期間内で、ほぼ一意な値をとるようにすることができる。

第 5 章

実装

本章では、まずはじめに SPLITTER と ANDER に共通する実装環境を説明する。その後、SPLAND の SPLITTER と ANDER の各部分の実装詳細について説明する。

5.1 実装環境

SPLAND では、高速にパケットを処理することが必要とされる。そのため本実装は、Linux のソケットインターフェースではなく、パケット I/O フレームワークである Data Plane Development Kit (DPDK) [16] を用いた。DPDK はユーザ空間で動作するポーリングモードのイーサネットドライバやメモリ管理機構などを提供している。従来のソケットインターフェースを用いて、パケットを処理するよりも、コンテキストスイッチやメモリ確保・解放にかかるオーバーヘッドを大幅削減できる。

ネットワーク処理を行うプロセスにはリアルタイム性が求められる。そのため、DPDK では、各プロセスはそれぞれ物理コアをほぼ専有するように設計されている。また、ローレベルの操作を必要とするため C 言語でフレームワークは実装されている。本研究では、同様に全て C 言語を用いて実装した。

表 5.1: 実装環境

使用 OS	GNU/Linux Debian 7.9
Kernel バージョン	3.2.0
パケット I/O フレームワーク	Intel Data Plane Development Kit 2.2.0

5.2 SPLITTER

SPLITTER の実装は、パケットを複製部とパケットの転送部分、マークパケット作成部の 3 つからなる。

パケットの転送部

パケットの転送部では、DPDK の API を利用してバルクでパケットの送受信を行う。これにより、キャッシュ効率を高める。これにより、高速にパケット処理を行うことが出来る。また、スプリッタは同時に同じパケットを送信しなければならないので、送信キューは1つのみを使用した。

パケット複製部

パケットの複製部分では、パケット全体をコピーしなければならない。コピー先は動的に確保する実装は遅い、そのため DPDK で提供されているメモリプールを使用した、メモリプールは、NIC の受信用に確保したメモリプールと同じものを用いることで、キャッシュの使用効率を高める。ここの処理でコピーするパケットは、NIC から受信されたパケットのみであり、マークパケットの複製については次に述べる。

マークパケットの作成部

マークパケットは、UDP パケットのデータ部分に 64bit の ID フィールドのみが存在するパケットとして実装した。マークパケット用にメモリプールを事前確保し、全てのエントリに事前にマークパケットの雛形を代入しておく。これにより、送信時に行う作業は、ID 部分の更新とチェックサム計算のみとすることが出来る。マークパケットはコピーせず、毎回メモリプールから取得し、先に述べた手順でパケットの作成を行う。これにより、実行時の処理を最少化することが出来る。

5.3 ANDER

ANDER の実装はパケット受信部分と各比較部分、パケットの受け渡しを行う2つのバッファ、および、Logger から構成される。本節では、まずはじめめにパケットの受け渡しを行う、パケットバッファの実装について説明し、その後各処理を行うプロセスの実装について説明する。

5.3.1 BinBuffer

BinBuffer は HASHRX と MERGER の間でのパケットの受け渡しの用いられるバッファである。BinBuffer は、検証機器間のパケットの処理時間差を吸収しなければならない。また、マークパケットによって、パケットをグループ分けする必要がある。それに加え、MERGER において、効率よくパケットの比較を行うために適したデータ構造でが求められる。そのため、BinBuffer をは図 5.1 のような、各要素がアンロールドリンクドリスト (ULL) の先頭を指すリングバッファとして実装した。

この BinBuffer は pop と push によるインデックスを変化させる操作と、bulk_insert 及び delete による、ULL にの操作の API を持つ。HASHRX は bulk_insert と pop を使用し、MERGER は push と delete を使用する。HASHRX と MERGER が互いにすべての操作をロックフリーで行えるように実装した。

また、ULL の要素の各中身には、ハッシュ値とパケットの構造体を指すポインタのペアが含

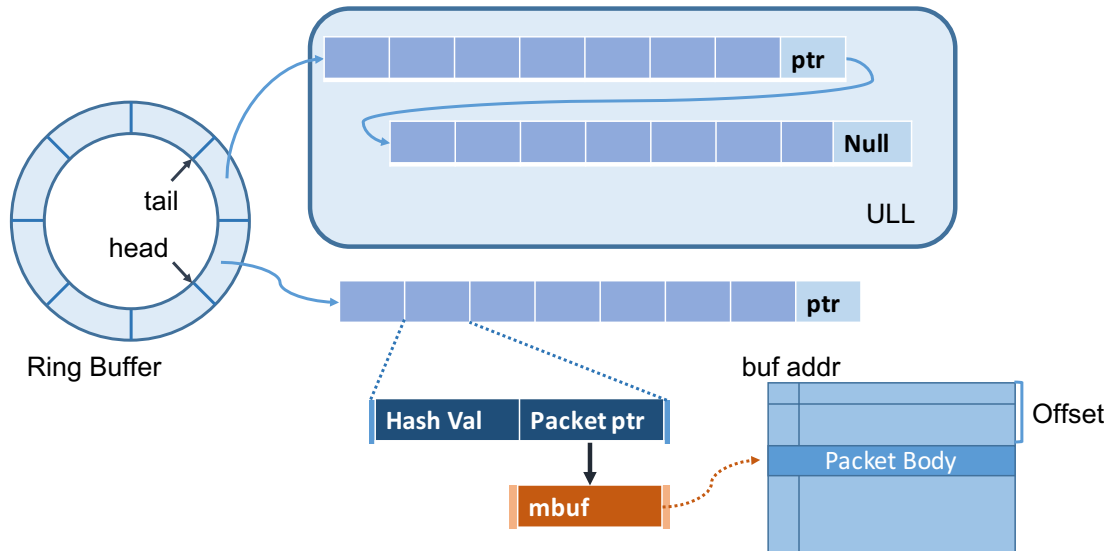


図 5.1: BinBuffer の構造

まれる。ポインタによるデータへのアクセスでは、メモリ領域にランダムアクセスするため、キャッシュの利用効率が悪い。そのため、パケットの構造体の中にハッシュ値を持たせた場合には、パケットの比較を行う毎に、ポインタアクセスが発生するため、キャッシュミスを起こす可能性がある。ULL の要素にハッシュ値を含めることで、パケットを比較する時に、ポインタによるアクセスを削減し、キャッシュの利用効率を高めることができる。

5.3.2 ExpireBuffer

ExpireBuffer は MERGER と KEEPER の間でのパケットの受け渡しに用いられるバッファである。ExpireBuffer では、パケットのネットワーク機器の挙動の差に繋がるパケットを有効期限内保存しなければならない。KEEPER は、本来同一動作をしているパケットであるにも関わらず、パケットのリオーダーやマークパケットのロスにより送られてきたパケットの再比較を行う。そのため、MERGER と同様に、効率的なパケット比較が求められる。また、比較するパケットの数が KEEPER の処理性能を上回って増えることがないようにしなければならない。そのため、ExpireBuffer はハッシュ値とパケットの構造体を指すポインタ、有効期限時刻を要素として持つ FIFO として実装した。

この ExpireBuffer は固定長 N の配列で実装することにより、処理性能を上回ってパケットが追加されることがないことを保証する。この ExpireBuffer には、通常の push と pop の他に、任意の位置のデータを削除する delete を実装した。この delete は、一致するパケットが見つかった時に使用される。

KEEPER がパケットを比較して、一致するパケットが見つかった場合には、特定の位置の要素を削除する必要がある。配列では全ての要素をずらすのは、 $O(N)$ のコストが発生する。そのため、ExpireBuffer では全てのフィールドを 0 埋めすることで空要素であることを表現する。

pop を呼んだ時には、次の要素が空要素である場合には、有効なエントリが存在するところまで、FIFO の先頭を表すインデックスをずらす。

5.3.3 HASHRX

HASHRX は各機器から送信されてきたパケットの受信を行い、そのハッシュ値計算を行う。パケットの受信は SPLITTER と同様である。パケットの受信後には、各機器の物理アドレスと論理アドレスの変換を行うために、パケットのプロトコル判別を行い、各プロトコル固有の処理を行う。本研究では、ARP 及び IPv4 の処理のみを実装した。

各 HASHRX は受信したパケットを複数の MERGER に振り分ける機能が必要である。このパケットの振り分けを行う際の、5 タブルのハッシュ値計算には、NIC のハードウェアオフローディング機能を用いる。これにより、HASHRX での処理の削減をおこなう。

マークパケットを受信した際には BinBuffer を先にすすめることで、MERGER にパケットを渡す。マークパケットの ID を確認して、マークパケットのロストが発生していた場合には、ロストしていた分もバッファを先にすすめる。その時、バッファがフルで進められなくなった場合には、パケットの受信処理に復帰する。

また、本システムのハッシュ値計算には、暗号学的なハッシュ関数のように、厳密さは必要ではない。それよりも、速度を重視した軽量なハッシュ関数の実装が求められた。そのため、軽量で速度が早く、キーバリューストアで用いられる、CityHash[24] をハッシュ関数として用いた。

5.3.4 MERGER

MERGER では、2 つの HASHRX から送られてきたパケットの同期をとる必要がある。パケットの同期をとるために、各バッファが保持しているマークパケットの ID を用いる。片方の機器からのパケットが到着していない、もしくは HASHRX でまだ処理中の状態には、次のパケットバッファが利用可能となるまでビジー状態で待機する。

処理可能なパケットバッファが存在する場合には、同一グループのパケットを順番に比較していく。この時、ハッシュ値が一致したもののみ、パケット全体を比較する。その結果、一致するパケットが見つかった場合にはその 2 つのパケットをメモリプールに返す。一致するパケットが見つからなかった場合には、ExpireBuffer にその時刻とともに積む。もし、バッファがフルの場合には、パケットを解放し、メモリプールに返す。

また、比較しているパケットがマスター機器から送信されてきたものであれば、外部のポートに送信処理を行う必要がある。ここでは、パケットのコピーはオーバーヘッドとなるため、パケットのコピーは行わず、パケットのリファレンスカウントを増加させて、参照のみを得る。その時、DPDK の API である `rte_pktmbuf_clone` 関数を用いて行う。

5.3.5 KEEPER

KEEPER では, ExpireBuffer に積まれたパケットの有効期限について管理を行う. 有効期限 ExpireBuffer は有効期限が先にくるものから, 順番となっている. そのため, 有効期限を迎えたパケットが存在すれば, 順番に pop を呼び, 同じパケットが存在するかを, 順番に確認していく. 確認方法としては, MERGER で行う方法とおなじである.

KEEPER において, 同じパケットが見つからなかった場合には, 機器の挙動の差分として, 記録するために Logger にパケットを転送する. この時, フレームワークで提供されている struct rte_ring というロックフリーのリングバッファを使うことでデータのやり取りを行う.

5.3.6 Logger

Logger はディスクへの Input / Output (I/O) や画面への出力処理などの I/O 全般を担う. ディスクへの I/O はコンテキストスイッチが発生するため, パケット処理を行うプロセスが直接行うのは厳しいためである. そこで, 各プロセスは DPDK で提供されている struct rte_ring というロックフリーのリングバッファを用いて, 他の全てのプロセスとデータの受け渡しを行うように実装した.

Logger には, 通常のログとパケットのダンプ情報を記録するために, 二種類のファイルに対して書き込みを行う. パケットのダンプ情報は, 機器間の挙動の差を発見するためパケットのフルダンプを保存する.

第6章

評価

本章では、提案手法である SPLAND の評価について述べる。

6.1 評価方針

本研究では、同一の動作をするように設定された、2台のネットワーク機器の挙動の差を検出することができるかどうか、および、そのシステムの性能評価の2点を評価軸とする。2台のネットワーク機器が同一の挙動をしているかを検証するために、ルータによる誤転送、Firewallによる通信の誤制御、2つのユースケースについて評価を行う。各ユースケースはそれぞれ、3.3節で分類した、ネットワーク機器が異なる挙動を示す場合となっている。そのため、この2つユースケースにおいて、各機器の挙動の差を検出することができるかどうかをもって、ネットワーク機器の挙動の差を検知できるかの評価とする。

まずはじめに、2つのユースケースに関して、性能評価の実験を行う。その後、各性能評価実験の結果を受けて、機能評価を行う。性能評価を行う2つのユースケースでは、実験を行う際には、次のようにルータに設定する。

- ルータによる誤転送
2台の検証機器のルータの経路情報に、一定の割合で異なる経路を設定する。
- Firewallによる通信の誤制御
2台の検証機器のルータのファイアウォールに、一定の割合で異なるパケットの制御設定をする。

6.2 ルータによる誤転送の性能評価

本節では、SPLANDの、ルータによって誤転送されるパケットの検出に関する性能評価の実験を行う。パケットサイズと入力トラフィック、ルータに設定される異なる経路の割合をそれぞれ変化させて実験を行った。

6.2.1 実験環境

実験に用いた機材は、検証ネットワーク機器を表 6.1 に、マシンを 6.2 に示す。実験は図 6.1 に示す、トポロジで行った。

表 6.1: ネットワーク機器

	Vendor	Model number	Firmware
ルータ 1	Juniper Networks	ex4200-24t	JUNOS 12.3R7.7
ルータ 2	DELL	S6000-ON	Cumulus Linux 2.2.1

表 6.2: 実験マシン

	CPU	Memory	NIC	台数
SPLAND ノード	Intel(R) Xeon E5-2697 2.60GHz dual-socket	256GB	Intel(R) X520-QDA1 x 2	2
パケットジェネレータ	Intel(R) Core i7 3770K 3.5GHz	32GB	Intel(R) X520-QDA1	1

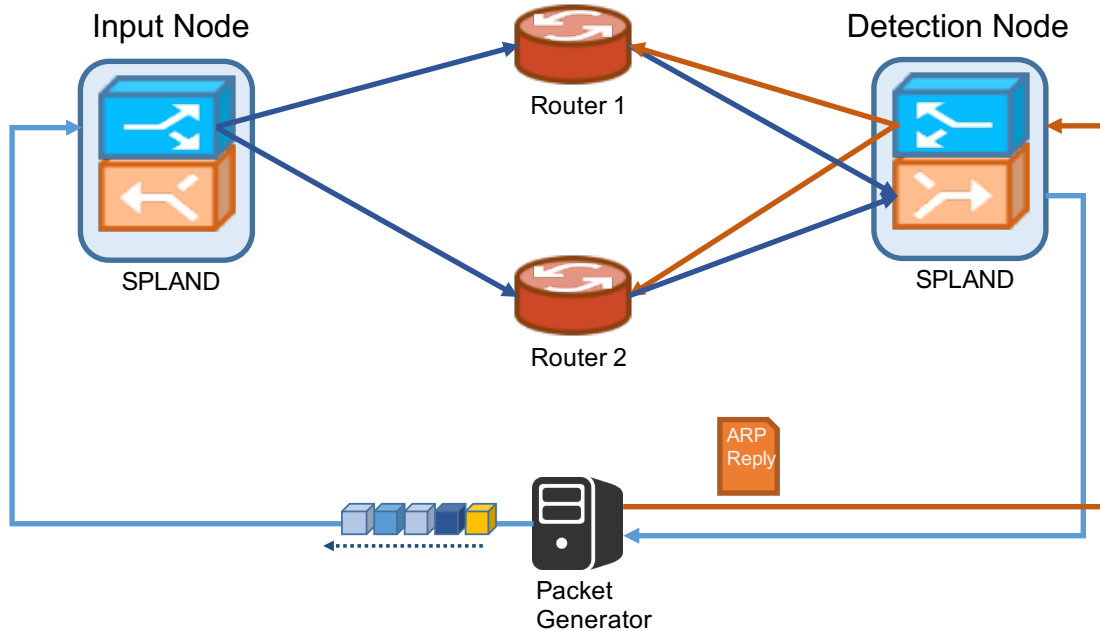


図 6.1: 実験で使用したトポロジ

2 台の SPLAND ノードで 2 台の異なるルータを挟み、一台のパケットジェネレータから、トラフィックを発生させる。パケットジェネレータでは、xorshift アルゴリズム [25] を用いて、ランダム値を生成し、宛先アドレスをランダム化したパケットを一定時間、片方の SPLAND ノードに送信した。64, 128, 256, 1518 バイトのそれぞれのパケットサイズにおいて、ルータ 1

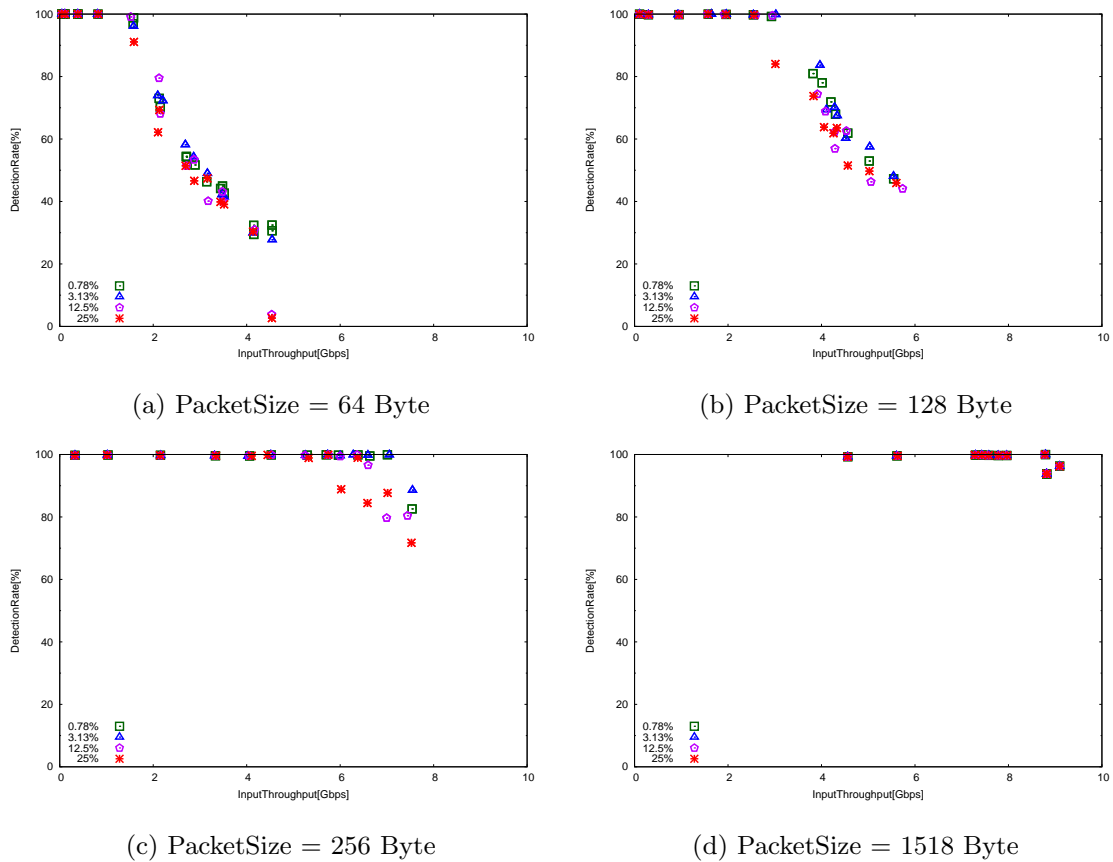


図 6.2: 各パケットサイズにおける, 入力帯域を変化させた時の, 異なる経路の割合での検知率の変化

とルータ 2 で異なる経路の割合と入力パケットレートを変化させて, 計測をおこなった. 各経路は全て, 同じプレフィックスレングスを持つ. そのため, ランダムに発生させたパケットの中で, ネットワーク機器において, 異なる挙動をする割合は, 機器間で異なる経路数の割合と一致する. 入力側のトラフィックは, 入力側の SPLAND ノードの SPLITTER で行い, 出力側のトラフィックの測定は, 出力側の SPLAND ノードで計測した. 入力パケットレートは, マークパケットを含まない値である. この性能評価実験では, 各ルータの処理時間による挙動の差は考えないものとする. また, マークパケットの間隔とリオーダの許容時間を共に $1\mu\text{second}$ ($1\mu\text{sec}$) と設定した. これは実験で用いたルータによる, リオーダがほぼ発生しないと仮定からである. また, MERGER, KEEPER とともに各 4 プロセス稼働させた.

6.2.2 実験結果

パケットサイズを固定し, 異なる経路の数の割合と入力トラフィックを変化させて実験を行った. この時のグラフを図 6.2 に示す. 64, 128, 256 バイトの順でパケットサイズを増加させると, 検知率がほぼ 100% で処理できる入力帯域が増加していることがわかる. また, 64,

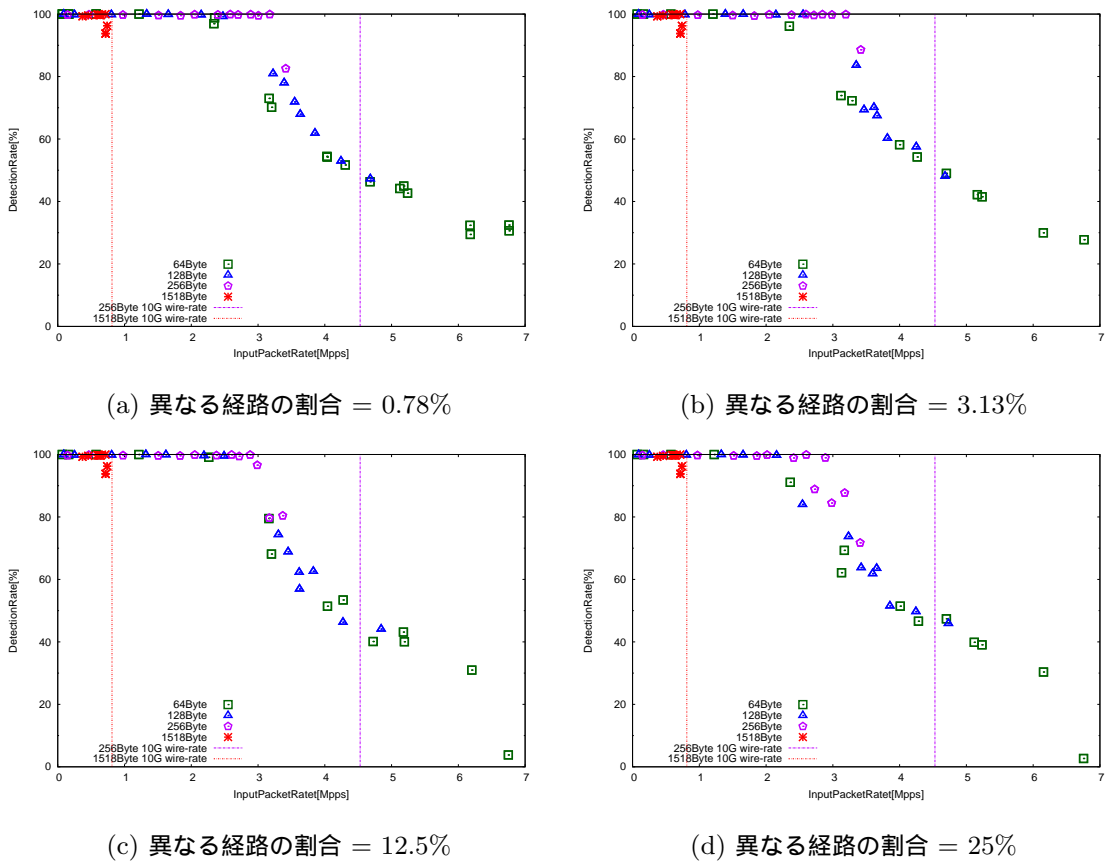


図 6.3: 異なる経路の割合における, 入力パケットレートを変化させた時の, パケットサイズによる検知率の変化

128 バイトのグラフからは異なる経路数の割合に対して, 検知率に影響はなかった. 256 バイトの実験では, 異なる経路の割合が 25% の時には, 6 Gigabit per second (Gbps) あたりから検知率が下がり始めている. 一方で異なる経路の割合が 3.13% 以下の場合には, 7Gbps 付近までほぼ 100% の検知率を保っている. しかし, それ以上のところでは, 全ての場合において, 検知率が減少していく. 一方 1518 バイトパケットの場合には, パケットジェネレータの不具合により, 低い帯域での測定ができていない. 入力が 4Gbps ~ 8Gbps の帯域区間では, 検知率 100% を達成している. また, 他のパケットサイズの結果から推察すると, 8Gbps 以下では検知率 100% を達成できていると考えられる. また, 異なる経路の割合によらず, 同じ結果を示している. このことから, 異なる経路の割合は本システムのボトルネックではないと考えられる. このことを示すために, 異なる経路の割合に対してパケットサイズにおいて, 入力パケットレートと検知率の割合に関するグラフを図 6.3 に示す. この 4 つのグラフ結果には, ほぼ差はなく, 本システムの実装では, 25% 以下での異なる経路の割合は, 2Mpps 以下の入力トラフィックに対して, ボトルネックではないことが判明した.

64, 128, 256 バイトパケットについて比較した時, 小さなパケットサイズの方が, 僅かに検知率が下がり始めるのが早い. また, 1518 バイトでは, 8Gbps 以上の帯域において検出率が低

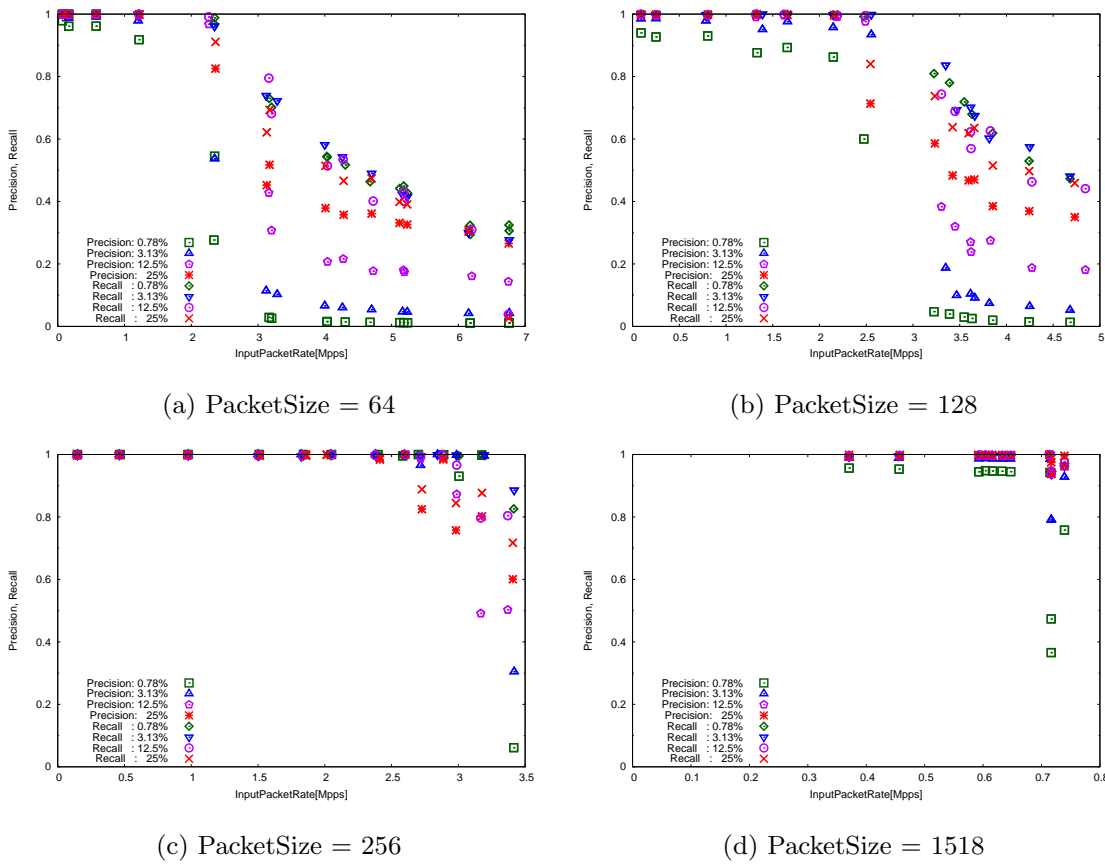


図 6.4: 各パケットサイズにおける、入力帯域を変化させた時の、異なる経路の割合による Precision および Recall の変化

下している。そもそも、約 0.81 Million packet per second (Mpps) が 10Gigabit Ethernet Card (10G NIC) のワイヤーレートである。 $1\mu sec$ の間隔でマークパケットを送信していること考慮すると、最大で受信可能パケット数は、約 0.76Mpps ほどである。このことから、1518 バイトパケットにおいて、0.76Mpps 以下で検知率が下がっている原因は帯域不足ではない。この検出率の低下について考察するために、誤検出したパケットに注目する。

本システムにおいて、トゥルー・ポジティブ (TP) は入力したパケットの内、ルータ 1, 2 で異なる経路にヒットするパケットの中で、検出できたパケットを指す。一方、フォールス・ネガティブ (FN) は、そうしたパケットの中で検出できなかったパケットのことである。また、フォールス・ポジティブ (FP) は、入力したパケットの内、ルータ 1, 2 で同じ経路にヒットするパケットの中で、誤検出されたパケットのことを指す。トゥルー・ネガティブ (TN) は、その他の同一経路にヒットするパケットで誤検出されなかったパケットである。このように、各値を求めて、適合率 (precision) および再現率 (recall) は次の式で求まる。

$$\text{適合率} = \frac{TP}{TP + FP}, \text{再現率} = \frac{TP}{TP + FN}$$

各パケットサイズにおける、Precision および Recall と入力パケットレートのグラフを、図 6.4

に示す。入力パケットレートを上げていった場合には、異なる経路の割合が低いほど、適合率が早く減少する。また、再現率に関しては、あまり差がないが、異なる経路の割合が低いほど、僅かに遅い。

6.3 ファイアウォールによる誤制御の性能評価

本節では、SPLAND の、ファイアウォールによって誤制御されるパケットの検出に関する性能評価の実験を行う。パケットサイズと入力トラフィック、ファイアウォールに設定される異なるのルール割合をそれぞれ変化させて実験を行った。

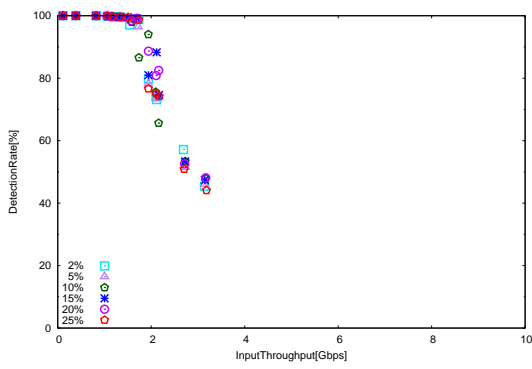
6.3.1 実験環境

実験に用いた機器及びトポロジは、6.2 節で述べた通りである。本実験では、ファイアウォールに、宛先アドレスによってトラフィックを制御するルールを設定した。各制御ルールは、同じプレフィックスレンジスのネットワーク宛ごとに、通過および破棄のどちらかの設定を行った。そのため、入力パケットをランダムに入れた場合には、ドロップされるパケットの割合は、異なるルールが設定された割合によって決まる。ルータ 1 とルータ 2 で異なる制御ルールの割合と入力パケットレートを変化させて、計測をおこなった。また、マークパケットの間隔とリオーダーの許容時間をは、ルータの実験と同じく $1\mu\text{sec}$ とした。

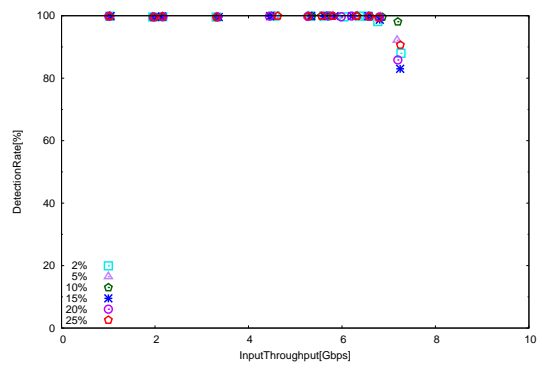
6.3.2 実験結果

パケットサイズを固定し、異なるルールの数の割合と入力トラフィックを変化させて実験を行った。この時のグラフを図 6.5 に示す。入力トラフィックのパケットサイズが、64 バイトにおいては、1Gbps、256 バイトにおいては、6Gbps、1518 バイトでは、8Gbps の入力トラフィックに対して 100% の検知率を達成することが可能である。また、異なるルールの割合は、各機器において、異なる挙動を示すパケットの割合と等しい。これから、異なる挙動を示すパケットによって、検知率 100% で処理可能な入力トラフィックの帯域は変化しないことがわかる。つまり、本システムの実装において、検知するパケットの数はボトルネックになることはない。

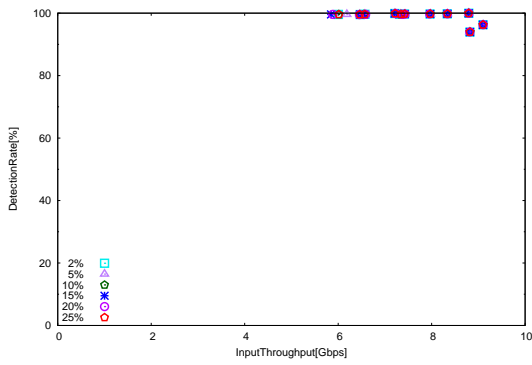
次に、異なるルールの割合で、各パケットサイズ毎に入力パケットレートを変化させて、検知率を計測した(図 6.6)。1500 バイトパケットでは、ワイヤレートの値で迎える前に検知率が僅かに低下する。64、256 バイトパケットでは、どの異なるルールの割合においても、2Mpps まで 100% で検知できていることがわかる。また、64 バイトパケットの方が先に検知率が減少し始めている。このことから、本システムにおいて、パケット比較におけるバイト数がボトルネックの原因となっていないことがわかる。



(a) PacketSize = 64 Byte

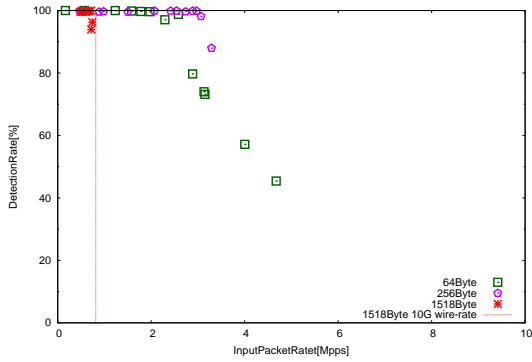


(b) PacketSize = 256 Byte

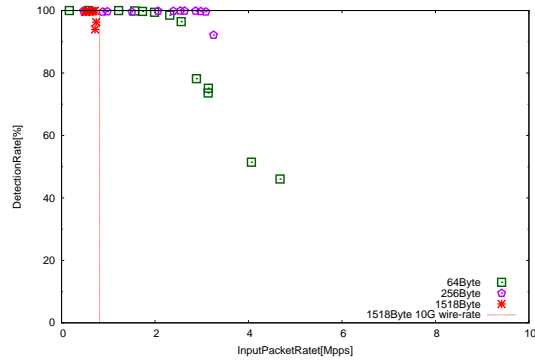


(c) PacketSize = 1518 Byte

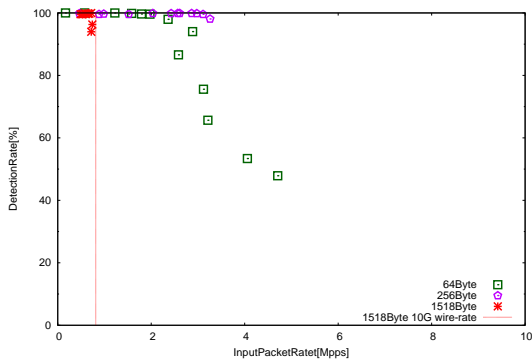
図 6.5: 各パケットサイズにおける, 入力帯域を変化させた時の, 異なるルールの割合での検知率の変化



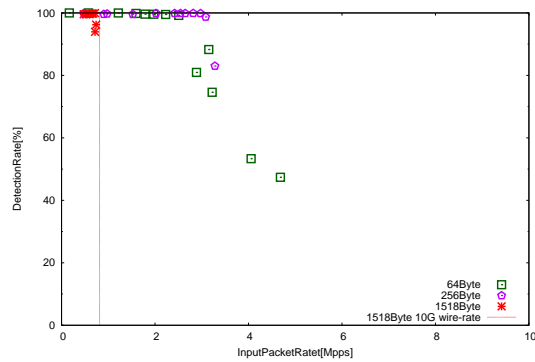
(a) 異なるルールの割合 = 2%



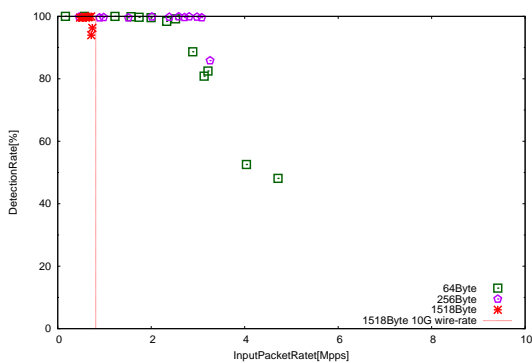
(b) 異なるルールの割合 = 5%



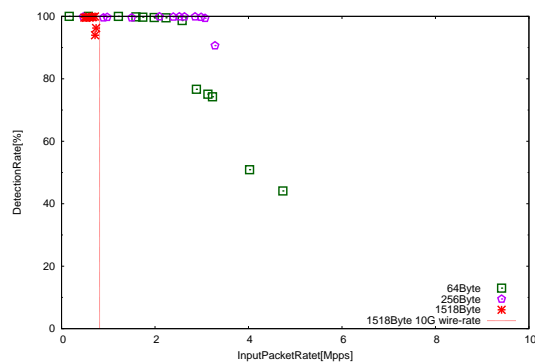
(c) 異なるルールの割合 = 10%



(d) 異なるルールの割合 = 15%



(e) 異なるルールの割合 = 20%



(f) 異なるルールの割合 = 25%

図 6.6: 異なるパケット制御ルールの割合における, 入力パケットレートを変化させた時の, パケットサイズによる検知率の変化

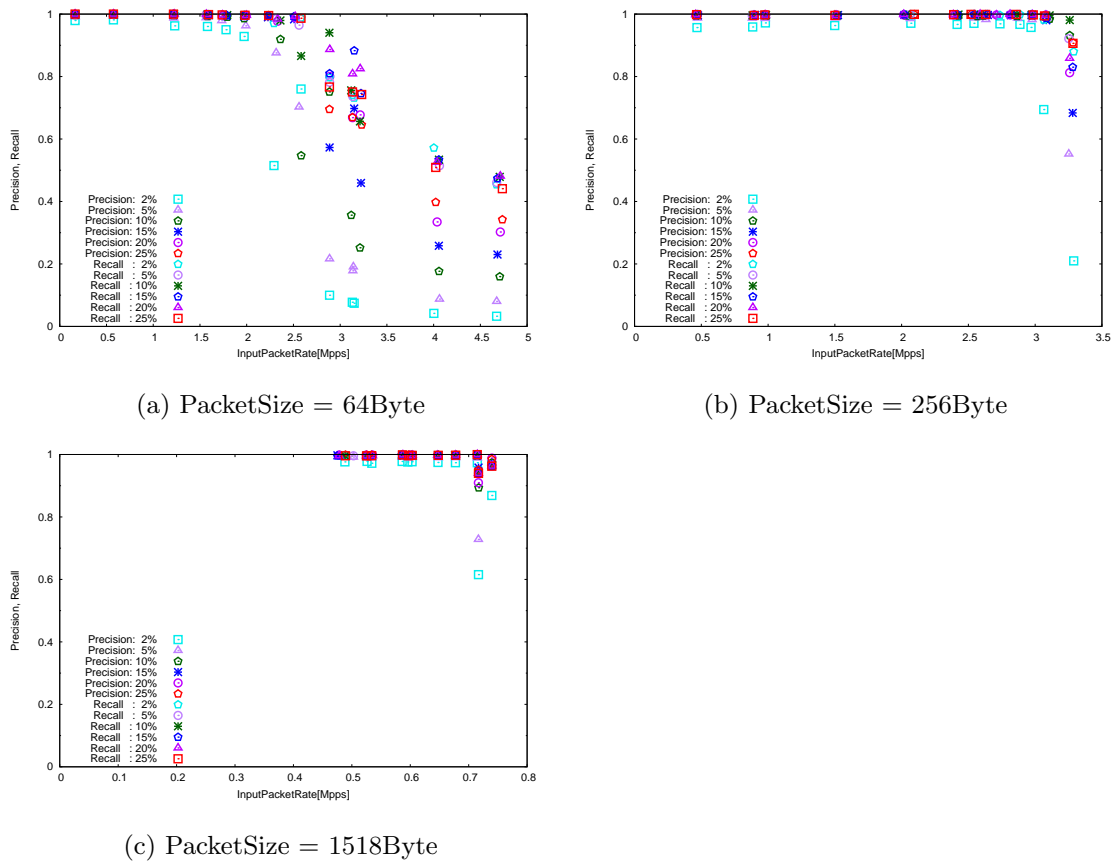


図 6.7: 各パケットサイズにおける, 入力パケットレートを変化させた時の, 異なるルールの割合による Precision と Recall の変化

次に, 各パケットサイズにおける, 入力パケットレートを変化させた時の Precision, Recall の変化を図 6.7 に示す. 64 バイトパケットでは 2Mpps まで, 256 バイトパケットでは約 3Mpps, 1518 バイトパケットでは 0.7Mpps まで高い適合率で再現率あることがわかる. しかし, その後再現率に比べて適合率の減少が早いことがわかる. また, 異なるルールの割合低いほど, 適合率がの減少が早く, 再現率に関しては特に差はみられない.

6.4 機能評価

6.26.3 節において, 行った実験・性能評価から, SPLAND が 3.3.2 で挙げた基本的な機能要件を満たすことを確認した. また, それぞれユースケースに対して, パケットサイズが 256Byte 以下の時には, 約 2Mpps まで, 1518Byte の時には約 8Gbps までの入力に対して, ほぼ 100% で検知出来ることを示した. その際, 適合率は高く, 誤検出の割合が低く実用的なシステムであるといえる.

ルータとファイアウォールのユースケースは, それぞれ, 図 3.2 で挙げた機器が異なる挙動をする時に見られる, パケットの分類の 2 つに該当する. これら 2 つの検出するパケット列の

種類の差によって、性能的な違いが生まれなかった。この理由は、SPLAND のシステム設計上、これら 2 つを区別せずに、出力パケットの差を検知しているからである。本研究では、残りの 2 つにあたるパケットのリオーダーおよび、処理時間差に関する実験を行わなかった。処理遅延によって発生するパケットの遅延に関しては、SPLAND システム設計上、2 台の機器の同期をとるために導入したマークパケットにより解決したと言える。それに加えて、パケットのリオーダーに関しては、システムの設計上、ルータとファイアウォールのユースケースで検知したパケットと同じ扱いをしている。そのため、各ユースケースにおいて、同程度での検出は可能と考えられる。よって、SPLAND は 3.2 で分類した全ての場合の機器の挙動の差を検出できると考えられる。

6.5 考察

SPLAND は、256Byte 以下のパケットにおいては、約 2Mpps, 1518Byte の時には約 8Gbps までの入力に対して、ほぼ 100% の検知率で動作することを示した。しかし、現在 25G, 40G をはじめとするより高速な NIC も登場している。そこで、SPLAND もより広帯域での高い検知率およびその精度を達成する必要がある。本実験の結果わかった、検知率 100% を維持できる入力が、2Mpps である理由を考察する。

今回の実験では、マークパケットの間隔およびリオーダーの許容時間を $1\mu\text{sec}$ としていた。マークパケットの間隔は各グループに含まれるパケットの数、つまり、一段目で比較するパケットの数を決定する。 $1\mu\text{sec}$ でマークパケットを送信した場合、最大 14 個のパケットが同一グループに割り当てられる。ランダムパケットを入力したため、マークパケットのロスがないと仮定すると、1つの MERGER では、3, 4 個のパケットの比較を行えばよい。また、KEEPER でも、 $1\mu\text{sec}$ 毎にパケットが破棄されていくので、バッファが溢れることはない。

次に、適合率と再現率の定義式から考える。??, 6.7 節のどちらにおいても、異なるルールの割合が小さいほど検知率が悪くなっている。このことから、FN に比べて、FP の増加が高い、つまり誤検出が多いことがわかる。本システムにおいて、誤検出が発生する条件は、パケットのリオーダーもしくはマークパケットのロスによる同期ミス、受信部分でのパケットのドロップが考えられる。本実験の検証ネットワーク機器での、パケットのリオーダーは発生しないものと考えられる。また、通信路上でのパケットのロスの可能性も極めて低いと考える。よって、受信部分でのパケットドロップが原因であると考えられる。

各検証機器に繋がる NIC の片側の受信部分で、パケットがドロップされると、そのパケットをもう一方が正しく受信した場合には、システム設計上誤検出してしまふ。また、マークパケットがドロップすると、一段目の比較部である MERGER でパケットの同期がとれなくなるため、多くのパケットが ExpireBuffer に積まれる。片側の ExpireBuffer があふれた場合には、多くのパケットが二段目の比較部分に渡らず、比較されない。その時、もう片側で正しく ExpireBuffer に積まれたパケットを、誤検出してしまふ。これにより、適合率の急激な低下が発生すると考えられる。よって、本システムのボトルネックは、受信部分にあると考えられる。

受信部分では、パケットを受信するために、パケットのためのメモリプールが存在する。こ

のメモリプールから、確保されたパケットのメモリは全てのコアで処理される。そのため、その実装および利用方法により受信処理がロックされている可能性がある。そこで、今後の課題は 100% の検知率を維持できる帯域幅を伸ばすことである。そのために、メモリプールの効率化や RSS によるマルチキューを利用したパケット受信部の再設計が有用だと考えられる。

第7章

結論

本章では、本論文のまとめと今後の課題を示す。

7.1 まとめ

本研究では、冗長されたネットワークにおいて重要となる、同一動作をするネットワーク機器の動作検証を行うために、機器間の挙動の差を検出するシステムを提案した。本研究では、対象とするネットワーク機器と機能を分析し、出力パケットに基づく機器間の挙動の差を定義を行った。それに基づき、挙動の差を検出するシステムの要求事項をまとめた。この要求事項に基づいて、システムの要件定義を行い、2台のネットワーク機器から送信されてくるパケットをバッファしながら、機器の挙動の差から発生する、異なるパケットを検出するためのシステムアーキテクチャ及びデータ構造の設計を行った。

その後、実際にシステムの実装を行い、2つのシナリオに対して実験を行い、その性能について評価を行った。この評価により、現在の SPLAND の実装で、64 バイトのショートパケットにおいても、1 Gbps の速度で十分に動作検証を行えることを示した。また、インターネットの一般的な MTU サイズである 1500 バイトパケットでは、約 8 Gbps の速度で検証できることを示した。このことから、現在の SPLAND の実装は実用的であるといえる。

7.2 今後の課題

本研究では、2ポートのネットワーク機器を対象にして、SPLAND の設計を行った。しかし、現実には、より多くのポートを持つ機器がほとんどである。こうした機器に対応するため、複数ポートでの動作検証方法が必要である。現在の構成では、複数ののマークパケットを重複受信する。そのため、同期をとるためのマスターノードを決定するなど工夫が必要だと考えられる。

また、本研究では、挙動の差のパケットを検出したが、それがどの挙動の差に当たるのかについてはなにも得られなかった。そのため、挙動の差によって検出したパケットの解析により、実際の差分がどのようなものであったかを判明させていくことも必要である。

そして最後に、現在、25G や 40G をはじめとした、高速な NIC が登場している。そこで、SPLAND もよりより広帯域での高い検知率を達成する必要がある。そのために、比較処理をより最適にする方法や、よりよいアーキテクチャの設計・開発を行う必要がある。

参考文献

- [1] Juniper Networks. Junos network operating system - juniper networks. <http://www.juniper.net/us/en/products-services/nos/junos/>. (Visited on 02/09/2016).
- [2] Juniper networks - 2015-07 security bulletin: Junos: Ipv6 sendd denial of service due to crafted packet (cve-2015-5360) - knowledge base. http://kb.juniper.net/InfoCenter/index?page=content&id=JSA10688&cat=SIRT_1&actp=LIST. (Visited on 01/14/2016).
- [3] Network processor overview — embedded systems experts. <http://www.barrgroup.com/Embedded-Systems/How-To/Network-Processors>. (Visited on 01/25/2016).
- [4] Differences between packets in slow path, fast pat... - live community. <https://live.paloaltonetworks.com/t5/Learning-Articles/Differences-between-packets-in-slow-path-fast-path-and-offloaded/ta-p/58845>. (Visited on 01/30/2016).
- [5] David Oppenheimer, Archana Ganapathi, and David A Patterson. Why do internet services fail, and what can be done about it? In *USENIX Symposium on Internet Technologies and Systems*, volume 67. Seattle, WA, 2003.
- [6] Timothy Nelson, Christopher Barratt, Daniel J Dougherty, Kathi Fisler, and Shriram Krishnamurthi. The margrave tool for firewall analysis. In *LISA*, 2010.
- [7] Sandeep Bhatt, Cat Okita, and Prasad Rao. Fast, cheap, and in control: a step towards pain free security! In *Proceedings of the 22nd conference on Large installation system administration conference*, pages 75–90. USENIX Association, 2008.
- [8] Karthick Jayaraman, Nikolaj Bjørner, Geoff Outhred, and Charlie Kaufman. Automated analysis and debugging of network connectivity policies. Technical report, Technical Report MSR-TR-2014-102, Microsoft Research, 2014.
- [9] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P Godfrey, and Samuel Talmadge King. Debugging the data plane with anteatr. *ACM SIGCOMM Computer Communication Review*, 41(4):290–301, 2011.
- [10] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *NSDI*, pages 113–126, 2012.
- [11] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P Godfrey. Veriflow: verify-

- ing network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review*, 42(4):467–472, 2012.
- [12] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. Automatic test packet generation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 241–252. ACM, 2012.
- [13] Peter Peresini, Maciej Kuzniar, and Dejan Kostic. Monocle: Dynamic, fine-grained data plane monitoring. Technical report, 2015.
- [14] Tcpcap/libpcap public repository. <http://www.tcpdump.org/>. (Visited on 02/09/2016).
- [15] Wireshark go deep. <https://www.wireshark.org/>. (Visited on 02/09/2016).
- [16] DPDK Intel. Data plane development kit. URL <http://dpdk.org>.
- [17] Luigi Rizzo. Netmap: a novel framework for fast packet i/o. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112, 2012.
- [18] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. Packetshader: a gpu-accelerated software router. *ACM SIGCOMM Computer Communication Review*, 41(4):195–206, 2011.
- [19] Luca Deri, Alfredo Cardigliano, and Francesco Fusco. 10 gbit line rate packet-to-disk using n2disk. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 441–446. IEEE, 2013.
- [20] Jihyung Lee, Sungryoul Lee, Junghee Lee, Yung Yi, and KyoungSoo Park. Flosis: a highly scalable network flow capture system for fast retrieval and storage efficiency. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, pages 445–457. USENIX Association, 2015.
- [21] Paul Emmerich, Florian Wohlfart, Daniel Raumer, and Georg Carle. Moongen: A scriptable high-speed packet generator. *arXiv preprint arXiv:1410.3322*, 2013.
- [22] 316125-001us. <http://www.intel.com/content/dam/doc/white-paper/i-o-acceleration-technology-paper.pdf>. (Visited on 02/03/2016).
- [23] Rfc 791 - internet protocol. <https://tools.ietf.org/html/rfc791>. (Visited on 02/03/2016).
- [24] google/cityhash - shell. <https://github.com/google/cityhash>. (Visited on 02/03/2016).
- [25] George Marsaglia et al. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 2003.

謝辞

本論文を執筆するにあたり、多くの方にご指導、ご協力いただきました。ここに心より感謝の意を表します。

学部時代より3年という短い間でしたが、多くのことをご指導いただきました東京大学情報理工学系研究科教授 江崎浩博士に感謝します。また、優しく、時には厳しく、研究のことからネットワークの運用、おいしいラーメン屋までいろいろなことを教えてくださった情報理工学系研究科特任助教 浅井大史博士に感謝します。

研究室の配属から、ネットワークに対する深い経験・知識から、研究の方針・方法や論文の書き方など様々なことを絶えずご指導いただきました、東京大学江崎研究室 中村遼氏に感謝します。

研究に関する様々な議論・質問にお付き合いいただきました、東京大学情報理工学系研究科講師 落合秀也博士、情報理工学系研究科特任助教 塚田学博士に感謝します。

研究室の先輩として、コンピュータへの向き合い方やいろいろと知らないことを教えてくださった池上洋行氏に感謝します。研究の相談に乗ってくださった小林諭氏に感謝します。また、研究室の同期として、研究室の仕事や作業、運営を行い、規律ある生活の見本を示してくれた中神啓貴氏に感謝します。共に研究励んだ、沼田進氏、李聖年氏、田中晋太郎氏、Tran Quoc Hoan 氏、陶冶氏に感謝します。また、ワークショップにおいて、ハードスケジュールに付き合ってくれた、内海究氏、田口奨也氏、井原大将氏、坂本裕紀氏に感謝します。ミーティング大臣として研究室を支えてくれた平田歩氏に感謝します。江崎研究室において、共に過ごした樋口海里氏、小林敦氏、楊璞安氏、菰原裕氏、江間実氏、石嶋紘大氏、肥嶋達也 氏、北里知也氏、戸間明日香氏に感謝致します。

そして、諸事務を通じて、研究室生活を支えてくださった江崎研究室秘書の高橋富美秘書、岩井愛映子秘書に深く感謝致します。

日々の息抜きとして楽しんだ Splatoon を開発して下さった、任天堂の任天堂情報開発本部制作部第2グループの皆様に感謝します。研究室に自らの私費で、Wii U および Splatoon のソフトを提供して下さった kawa 氏に感謝します。故障した Wii U のコントローラの代わりに新規コントローラを購入し快適なゲーム生活を提供して下さった sat 氏に感謝します。人が成長がする様子を日々示してくれた choruru 氏に感謝します、これからも成長を期待しています。

最後に、本論文の執筆にあたってお世話になりました家族、友人、お世話になりました全ての皆様に深く感謝致します。

