卒業論文　2018年度（平成30年度）

netmapを用いた高スループットなログフォワーダの提案と実装

慶應義塾大学 環境情報学部

増田 和晃

徳田・村井・楠本・中村・高汐・バンミーター・植原・三次・
中澤・武田 合同研究プロジェクト

2018年01月

卒業論文　2018 年度（平成 30 年度）

netmap を用いた高スループットな ログフォワーダの提案と実装

論文要旨

サーバ管理者は多様化していく Web サービスを安全に運用していくために、サーバ管理者は syslog や NetFlow といったシステムやネットワークのログ情報を管理することがある。ネットワークを飛び交うトラフィック量の増大に伴い、管理しなければならないシステムやネットワークのログ情報も肥大化しているため、それらを効率的に扱うために、ログの出力と収集の機器の中間で、集約や転送を行うログフォワーダという仕組みが存在している。しかし、従来のログフォワーダでは、今後増大していくトラフィック量に対応する上での性能的な問題が想定される。

その為本研究では、既存のパケット転送処理手法を改善した、高速パケット I/O フレームワークである netmap を利用することでスループットを向上させ、今後更に増えていく事が想定されるトラフィックにも対応できるログフォワーダを提案する。netmap を用いることで、OS がパケット毎に行うメモリコピーなどのコストを抑えパケット単位の処理時間を削減し、複数のサーバからの大量のトラフィックへの対応を実現する。

この結果、高スループットなトラフィックが発生することを想定された環境として構築したログ出力サーバからの、大量なトラフィックの処理に成功し、また従来利用されている fluentd などのフォワーディングアプリケーションが処理しきれない、10GbE において 1,500 バイト長パケットで 50,000pps を越える入力に対しても、100%のパケット転送処理を実現できた。本研究は、今後更に発展していくネットワークの傍らで増え続ける、膨大なログデータの収集方法に対し新たな解決策を示した。

キーワード

netmap、 ネットワークトラフィック、 NetFlow、ログフォワーダ

慶應義塾大学 環境情報学部

増田 和晃

Abstract Of Bachelor's Thesis Academic Year 2018

Proposal and Implementation of High-Throughput Log Forwarder Using Netmap

Summary

Server administrators have the opportunity to manage log information of systems and networks such as syslog and NetFlow in order to safely operate diversifying Web services. Since the log information of systems and networks that must be managed is also becoming enlarged as the amount of traffic that flies over the network increases, in order to handle them efficiently, in the middle of log output and collection machines, There is a mechanism called a log forwarder that performs forwarding. However, in conventional log forwarders, performance problems in dealing with the increasing traffic volume in the future are assumed.

Therefore, in this research, throughput is improved by using netmap, a high-speed packet I/O framework which improved existing packet processing method, and it is also possible to handle traffic expected to increase further in the future We propose a log forwarder. Using netmap reduces the cost of memory copying performed by the OS for each packet, reduces processing time per packet, and realizes response to a large amount of traffic from multiple servers.

As a result, we succeeded in handling a lot of traffic from log output server built as the assumed environment, and can not handle forwarding applications such as fluentd, which was used conventionally, with 1,500 byte long packets at 10 GbE even for inputs exceeding 50,000 pps, packet processing of 100% could be realized.

This research has significance in showing a new solution to the collecting method for enormous log data that continues to increase beside the network that will further develop in the future.

Keywords

netmap, Network Traffic, NetFlow, Log Forwarder

Bachelor of Arts in Environment and Information Studies
Keio University
Kazuaki Masuda

目 次

図 目 次

表 目 次

第1章　序論

本章では、まず本研究の背景となる現代のサーバ、及びトラフィック管理についての現状を述べ、解決すべきパケット転送処理に関する潜在的な問題点について明らかにし、その上で本研究の目的について説明する。また、本論文の構成についても述べる。

1.1　本研究の背景

情報通信技術が社会の根底を支えるインフラストラクチャとして欠かせないものとなった近年では、毎年多くの企業や団体がインターネットを利用した新たなアプリケーションやサービスを提供し、またそれを利用するユーザも年々増加している。書籍や映画、論文といったメディアデータがインターネットを通じて提供され、オンラインゲームを楽しんだり、ビデオ通話やチャットといったコミュニケーションを行ったりといった日常的なところをはじめとして、公的な手続きや、更には株や仮想通貨を用いた商取引までもがインターネットを用いて行われるようになった。また、人々の間にはスマートフォンが広まり、取得できる情報量が格段に向上した。最近では Raspberry Pi をはじめとするネットワークに接続して情報のやり取りが可能な機器の小型化、低価格化も進み、個人による購入も容易となった。

このようなネットワーク産業の発展に伴い、通信を管理するサーバの技術も大きく進歩し、より複雑化したシステムを安定して運用していくため、ネットワークの管理者は様々な対策をとる必要がある。

複雑なネットワークを監視する上で用いられる技術のひとつに、NetFlow と呼ばれるものがある。NetFlow とは、ネットワーク上で流れるトラフィックのフロー情報を計測できる技術である。フロー情報とは、ルータやスイッチなどで計測される、ネットワーク上の共通の属性を持ったパケットグループのことを指す。例えば、送信元や送信先の IP アドレスやポート、プロトコルなどの情報が共通なパケットを集約して「フロー」と呼ばれる情報単位で管理し、それによってユーザやアプリケーション単位でのトラフィックの監視、分析が可能となる。フロー情報を分析することによって、操作上またはセキュリティ上の問題を明らかにし、ネットワークセキュリティをより強固にすることができる。また、大規模なネットワークを構築するために用いられる、Cisco のキャリアグレード NAT などでは、ユーザの利用状況を確認するために用いられるセッションログを NetFlow を

利用して保存している。

NetFlow は一般に、セキュリティ系の監視や定常モニタリングといった、個別のアプリケーションに振り分けられ、複数の解析に用いられる。複数存在する宛先毎に各機器からのフロー情報を直接送信するためには、全ての機器同士を接続する必要があるが、図 1.1 に示すとおり、送信や受信機器を一台を追加するだけでも接続作業や、config の変更の量が大きく増え、更にはネットワークが複雑化していく為、保守作業が難しくなるといった問題が存在する。

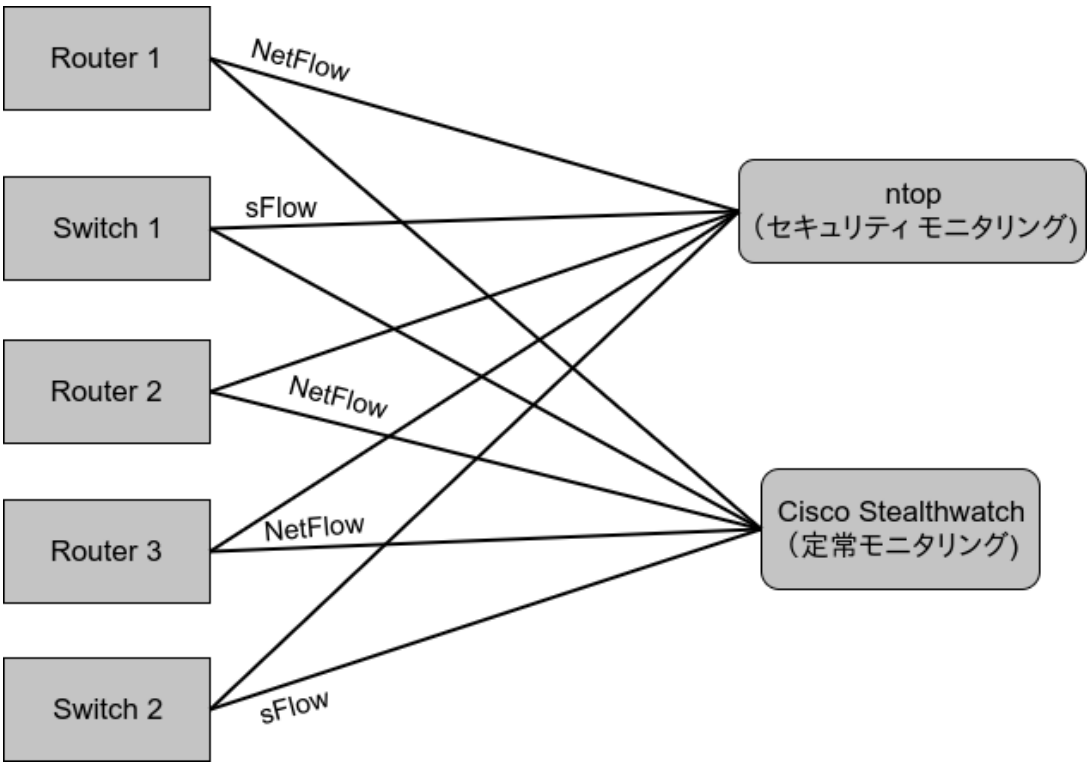


図 1.1:　ログ管理におけるネットワーク機器の接続の複雑化 [1][2]

この複雑化した構成を解決するために、複数の送信機器から得られるフロー情報を集約するインターフェースを用意し、一つのデータフローとしてそれぞれの宛先に送信する、ログフォワーダの仕組みが考案されている。基本的な考え方としては、送信機器の送り先を一つに絞り、その先をログフォワーダにして、パケットのポートなどによってサービスを識別し、送信先を決める形である。

1.2　本研究が着目する課題

前節で述べたログフォワーダは、構成上一時的に全てのログ送信機器からのトラフィックが集中するため、ネットワークの発展やハードウェアの進化によって増加するログの量を捌き切れなくなるという性能的な問題が想定される。

例えばNAT等の環境では、出力されるNetFlowには利用ユーザのログインやログアウトを含めたセッションログなど様々なものが含まれる為、膨大な量のデータが生まれてしまう。2017年に行われたInterop Tokyoのために構築された大規模ネットワーク、「ShowNet」では、一週間という短い期間でありながら、実に161GBものログデータが取得された[3]。ログデータの増大による高トラフィックに対し、通信が集中する従来のログフォワーダでは、全てのパケットの処理が困難になっていくことが想定される。以上のことから、ログフォワーダに対する大量のトラフィックをどう扱うが課題として挙げられる。

1.2.1 パケット転送処理における遅延

技術の進歩により、イーサネットの通信速度自体は年々高速化しており、その過程で10GbE、40GbEといった極めて高速な通信をサポートするNICが個人の用いるPCサーバにおいても用いられるようになっている。しかし、パケットを処理するOS側のAPIが、こうした高速のパケットI/Oを想定した作りになっていないため、性能を十分に発揮することができずにいる。具体的には、パケットの送受信時、ペイロードをアプリケーションとカーネルメモリ間でメモリコピーを行っていることや、パケットバッファのメモリ領域確保、解放の処理に時間がかかってしまうことが挙げられる。

1.3 本研究の目的

1.2.1項にて、通信速度が上がらない原因の一つに、OSによるパケットの処理方法に問題があることを示した。この問題が、1.2節に示した課題に影響していると仮定し、本研究では、netmapと呼ばれる高効率パケットI/Oフレームワークを利用して、パケット毎にOSが行うメモリコピーなどのコストを抑えた高スループットなログフォワーダ「gForwarder」を開発することで、パケット単位の処理時間を減らし、ログフォワーダが複数の送信機器からの大量のトラフィックに耐えうることを目的とする。ログフォワーダを用いたログ管理における全体の構成図を、図1.2に示す。

1.4 本論文の構成

本論文は全6章で構成される。第2章では、本研究の対象となるフロー情報の規格について詳しくまとめ、また関連研究についても記す。第3章で本研究で提案する手法について述べ、関連技術についても紹介する。第4章で実装について述べる。第5章にて提案手法の評価を行い、関連研究と比較する。第6章で本研究によって得られた結論を述べる。

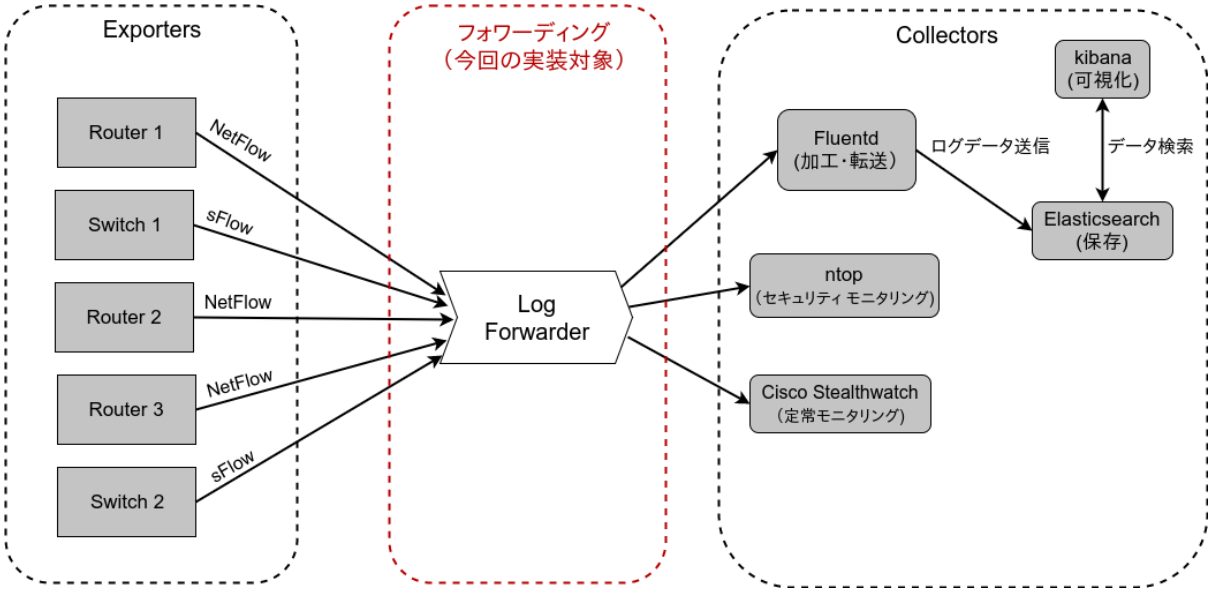


図 1.2: ログフォワーダを含む全体の構成

第2章　既存技術・関連研究

本章では、本研究の対象となるフロー情報の規格である NetFlow や、パケット I/O フレームワーク netmap について整理を行う。また他のフロー情報規格やネットワーク I/O 性能向上手法、既存のログフォワーダについても記す。

2.1　NetFlow

NetFlow とは、ネットワーク上で流れるトラフィックのフロー情報を計測できる技術である。Cisco によって 1996 年に開発された。フロー情報とは、ネットワーク上を流れる、共通の属性を持ったパケットグループのことを指す。例えば、送信元や送信先の IP アドレスやポート、プロトコルなどの情報が共通なパケットを集約して「フロー」と呼ばれる情報単位で管理し、それによってユーザやアプリケーション単位でのトラフィックの監視、分析が可能となる。一般に、このフロー情報を生成、送信するルータなどのネットワーク機器を総称して「Flow Exporter(Exporter)」と呼び、それらを収集、保存、計測するサーバやストレージなどを「Flow Collector(Collector)」と呼ぶ。

NetFlow 自体は Cisco によって開発されたが、他の企業でも同じ機能を持つものが開発されており、その一つに Huawei 社の NetStream[4] が存在する。

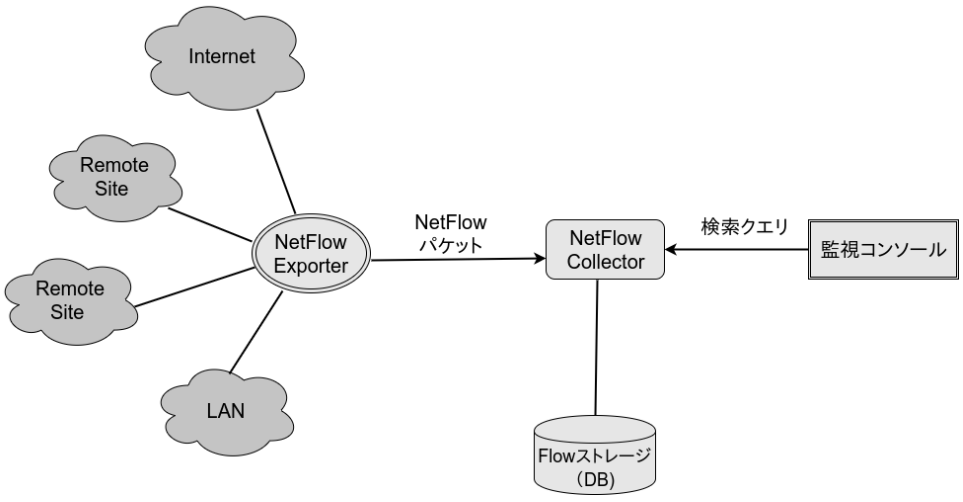


図 2.1: NetFlow (version 5) アーキテクチャ [5]

パケットのモニタリングは NetFlow に対応した Cisco のインターフェースによって行われ、同一フロー情報が発見されると、インターフェース内に NetFlow 用のキャッシュが作成され、保存される。NetFlow には複数のバージョンがあり、2018 年 01 月現在では 1 から 9 までが存在する。中でもスタンダードなバージョンである version 5 では、以下の 7 つの情報をキーとして固定利用しフロー識別の判断を行う [6]。

- 宛先 IP アドレス
- 送信元 IP アドレス
- 宛先ポート番号
- 送信元ポート番号
- L3 プロトコル
- ToS
- 入力インターフェース

7 つの属性において、同一のものがなければ新規フローとして、NetFlow キャッシュ内に新たにフローエントリが作成される。マッチするものがあれば、既存のフローデータを更新する。

取得されたフロー情報には、フィールドとして以下の内容が含まれる。

- 送信元＆宛先 IP アドレス
- 送信元＆宛先ポート番号
- 入力＆出力インターフェース
- ToS バイト (DSCP)
- フローのバイト数＆パケット数
- 送信元 AS 番号、あて先 AS 番号

NetFlow キャッシュが期限切れとなると、フロー情報はキャッシュテーブルから削除され、NDE(NetFlow Data Export) パケットとしてカプセル化され、NDE に対応するサーバやストレージに UDP パケットによって送信される。

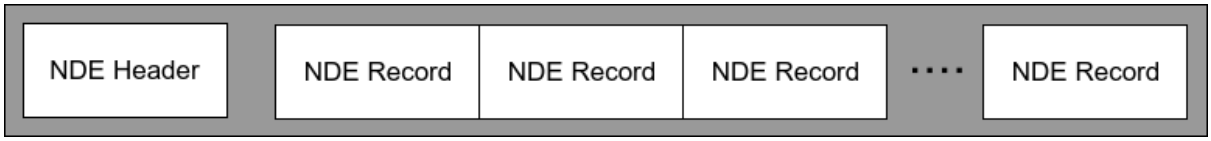


図 2.2: NDE パケット (version 5) の構造

2.1.1 NDE パケット (v5) のヘッダー

NetFlow version 5 パケットのヘッダの内部構造を図 2.3、表 2.1 に示す [7]。

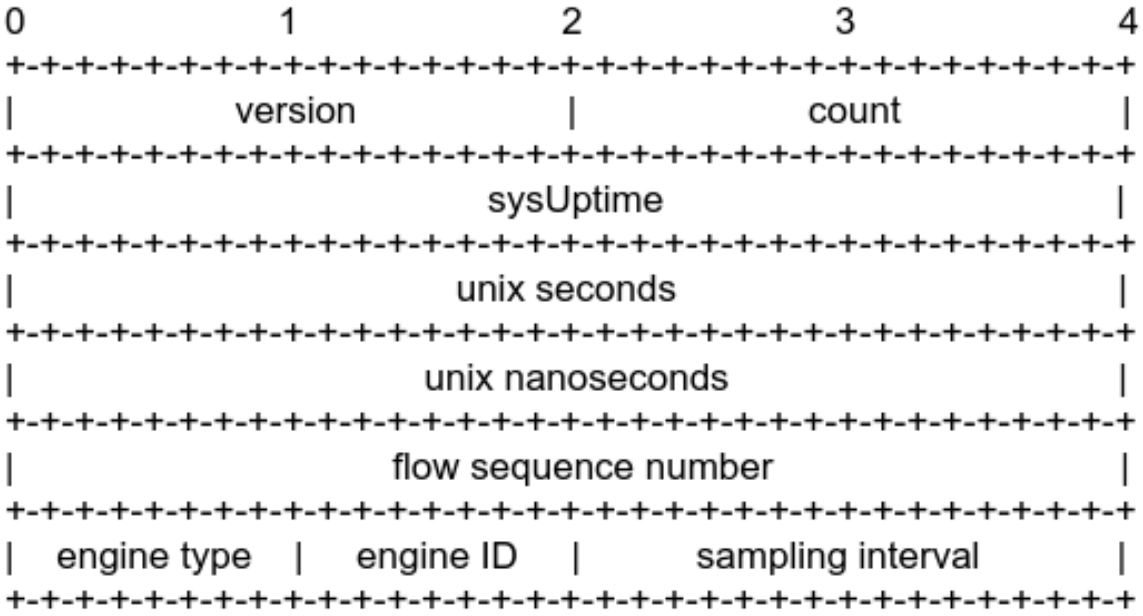


図 2.3: NetFlow v5 パケットヘッダ

表 2.1: NetFlow v5 パケットヘッダ構造

項目名	説明
version	NetFlow パケットのバージョン情報 (=5)
count	パケットに含まれるフロー情報数 (最大 30)
sysUptime	パケットの生成時刻（デバイス起動後からのミリ秒）
unix seconds	パケット生成時刻 （1970 年を 0000 として何秒後か）
unix nanoseconds	パケット生成時刻 （1970 年を 0000 として何ミリ秒後か）
flow sequence number	フロー単位統計データの生成ごとに増加するシーケンス番号
engine type	フロー中継エンジンの種類
engine ID	フロー中継エンジンの ID 番号
sampling interval	2bits:サンプリングモード番号 14bts:サンプリング間隔

2.1.2 NDE パケット (v5) のレコード

NetFlow version 5 パケットのレコードの内部構造を図 2.4、表 2.2 に示す [7]。

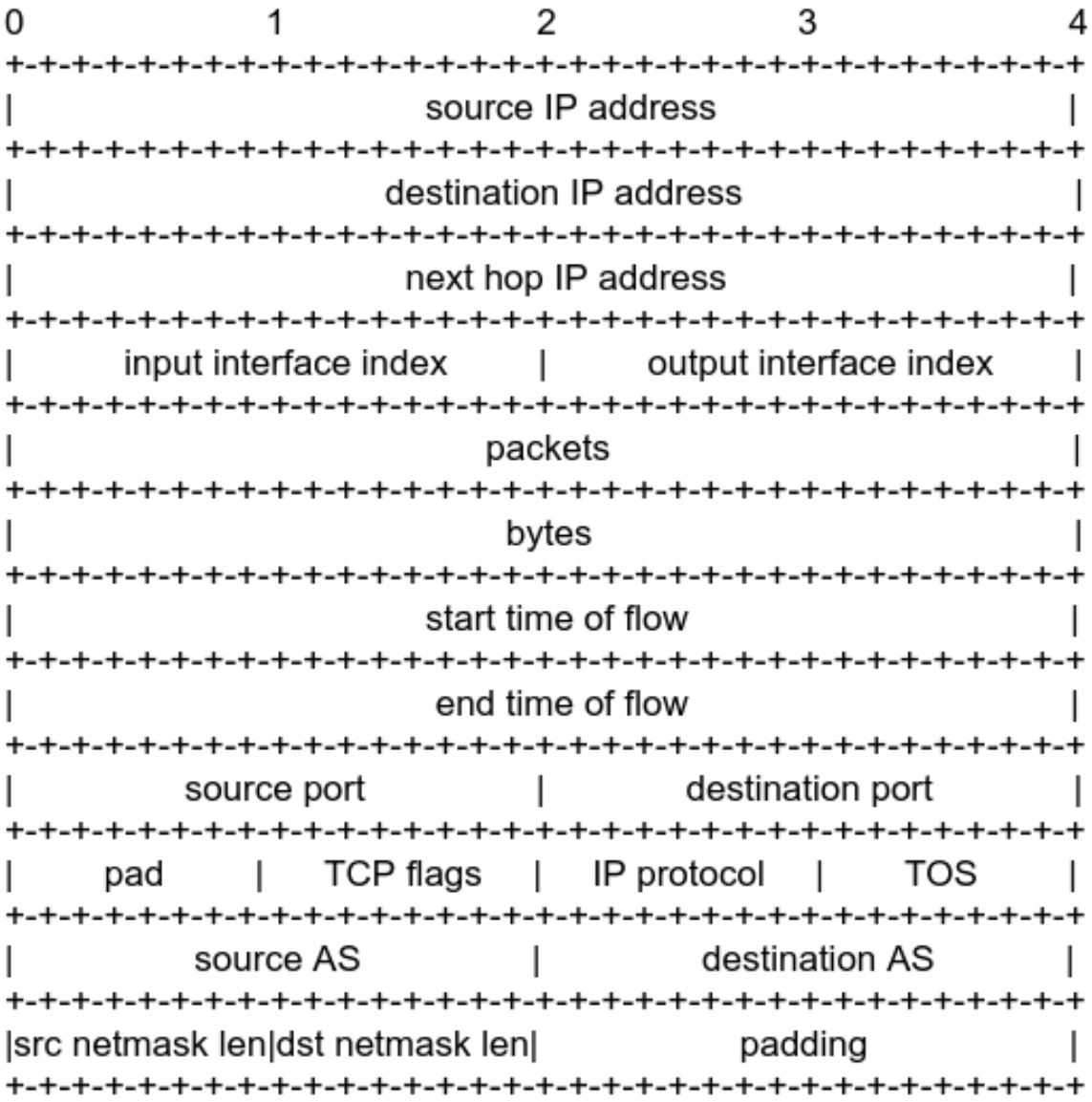


図 2.4: NetFlow v5 パケットレコード

また最新のバージョンである version 9 では、サポートされるプロトコルの数が増え、更に Flexible NetFlow と呼ばれる新しい仕組みが使用されている。この仕組みは、version 5 においては固定のフォーマットを用いていたのに対し、判断基準となるキー及び取得されるフィールドを大幅に拡張した上で、その中から自身で任意に選択・設定ができるようになっている。Exporter は指定されたキーとフィールドに従ってフロー情報を生成する。設定されたフォーマットはテンプレートレコードとして Collector に定期的に送信さ

表 2.2: NetFlow v5 パケットレコード構造

項目名	説明
src IP addr	送信元 IPv4 アドレス
dst IP addr	宛先 IPv4 アドレス
next hop ip addr	次の転送先ルータの IPv4 アドレス
input interface index	受信インターフェースの SNMP interface idx
output interface index	送信インターフェースの SNMP interface idx
packets	フローのパケットの総数
bytes	フローのパケットの総バイト数
first	フロー開始パケット受信時の sysUptime(秒)
last	フロー最終パケット受信時の sysUptime(秒)
src port	TCP/UDP 送信元ポート番号
dst port	TCP/UDP 宛先ポート番号
flags	TCP フラグの累積
IP protocol	IP プロトコルタイプ (6=TCP, 17=UDP)
TOS	IP の TOS
src AS	送信元もしくは送信元側隣接ピアの AS 番号
dst AS	宛先もしくは宛先側隣接ピアの AS 番号
src netmask length	送信元 IPv4 アドレスのプレフィックスマスクビット数
dst netmask length	宛先 IPv4 アドレスのプレフィックスマスクビット数

れ、後続して指定されたフォーマットを元にしたデータレコードが送信される。Collector は、このテンプレートレコードを元に後続するレコードをパースし、情報を取得する。

2.1.3 NDE パケット (v9) の構造

NetFlow version9 のパケット構造を図 2.5 に示す。

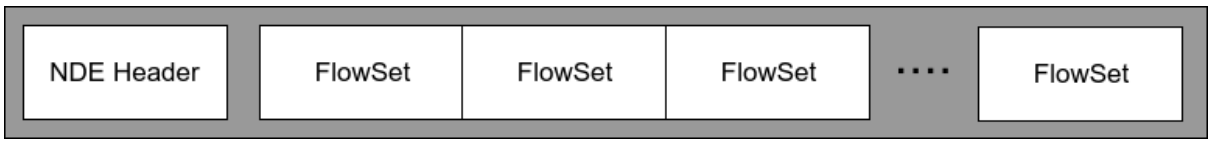


図 2.5: NDE パケット (version 9) の構造 [8]

NetFlow version 9 は、パケットの概要を示すヘッダと、それに続く一つ以上の FlowSet で構成される。FlowSet には、収集するフロー情報のテンプレートレコードとなる Template FlowSet、フロー情報以外の、アプリケーションデータやインターフェース情報などの特殊データのテンプレートレコードとなる Options Template FlowSet、そして収集データ自体であるデータレコードとなる Data FlowSet が存在する。NetFlow version 9 では Template を定義することによって機器の構成変更や停止を行うことなく取得するパラメータを調

整することができる。

2.2 sFlow

sFlow とは、InMon 社によって導入された、NetFlow と同じフロー計測技術である [9]。特徴として、全てのフロー情報を計測する NetFlow と異なり、数フローに 1 フローの割合で計測を行うサンプリングベースであるため、計測漏れが発生する分、機器への負荷は小さくなる。NetFlow と同じく対応したルータやスイッチで計測を行い、フロー情報が一定数溜めてから出力される。また IPX、Appletalk、XNS といった IP 以外の L3 プロトコルや、L2 レイヤーにも対応している。

パケットの基本的な情報を示すヘッダ情報、複数個のフローサンプル、複数個のカウンタサンプルから成る。

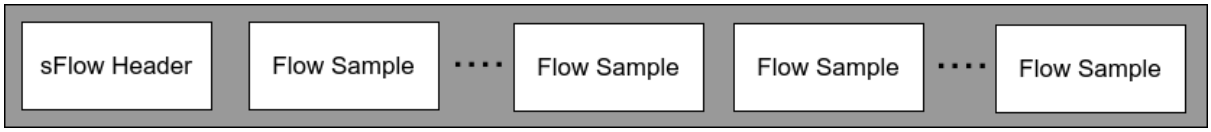


図 2.6: sFlow パケットの構造

2.2.1 フローサンプル

フローサンプルとは、受信されたパケットからサンプリングされたパケットの情報を、Collector へ送信するためのフォーマットである。NetFlow と同じく、パケット自体が持つ情報に加え、送受信インタフェースなど、パケットには含まれない情報も収集するため、詳細なネットワーク監視が可能となる。

フォーマットの内部構造は、一つのフローサンプルの概要を示すフローサンプルヘッダ、IP ヘッダその他のプロトコルヘッダの情報を示す基本データ形式、スイッチ情報やルータ情報などのデータを示す拡張データ形式の三つから成る。

2.2.2 カウンタサンプル

カウンタサンプルは、到着したパケット数やエラー数などの、インターフェース統計情報を送信する。インターフェースの種別により、コレクタに送信するフォーマットが異なる。フォーマットの内部構造は、カウンタサンプルの概要を示すカウンタサンプルヘッダ、インターフェースの種別ごとに分類されるカウンタサンプル種別、カウンタサンプル種別によりそれぞれ指定された統計情報を示すカウンタサンプル情報から成る。

2.3 IPFIX

IPFIX は、Cisco 規格である NetFlow の v9 をベースに IETF によって標準化が行われている取り組みである [10]。フィールド指定フォーマットを導入することで、ベンダーによって取得可能な情報の範囲を拡張でき、セッション層以上の情報も取得可能である。加えて、テンプレートレコードを明示的に消去する Template Withdraw Message の導入のほか、IPsec や TLS を用いたセキュリティ対策など、様々な機能が存在する。

2.4 netmap

netmap とは、Pisa 大学の Luigi Risso 教授によって設計・開発された、高効率のパケット I/O フレームワークである [11]。FreeBSD には標準搭載され、また Linux や Windows といった OS にも対応している。高速化するネットワークに対応できるようにするための試みの一つで、事前に用意された固定長の線形バッファが事前に確保され、パケット毎にメモリを確保、破棄するコストを削減している。

ソケットを用いた通常の OS の処理との違いを以下に示す [12]。

2.4.1 OS によるパケット転送処理

NIC には、RX(受信) と TX(送信) それぞれのためのリングバッファが存在する。NIC 内のレジスタにはこれらのリングにおける利用開始位置 (`head`) と終了位置 (`tail`) が記録されている。リング内の利用可能なスロットには、個数分確保された空の `sk_buff` で埋められている。`sk_buff` はソケット・バッファと呼ばれ、linux 内でパケットデータを格納するためのバッファであり、`linux/include/linux/skbuff.h` にて宣言されている [13]。表 2.3 に示す通りの、データの位置を管理する 4 つのフィールドが存在し、この位置に対応した別の領域にパケットデータが格納され、ネットワークレイヤで扱われる。

表 2.3: `sk_buff` のデータ位置を管理するフィールド

`head`	データ格納用バッファの先頭を示す
`data`	バッファに格納されているデータの先頭
`tail`	バッファに格納されているデータの終端
`end`	データ格納用バッファの終端

(1) 受信時

OS 処理によるパケット受信について、図 2.7 と共に整理する。

接続線上で受信されたパケット (①) は、必要に応じて分割された後、DMA によってカーネルメモリにコピーされ、RX リングの head の位置のスロットの sk_buff に登録される (②)。完了後、head が更新され (③)、NIC は新たなパケットが来たことを CPU へ通知する。これにより NIC ドライバは割り込みを承認し、続いてソフトウェア割り込みを行う。sk_buff はホストスタックへ転送される (④)。それが完了すると、新たな sk_buff を確保し、それに合わせて tail を更新する。スタックに届けられたメッセージは、最終的に CPU にてヘッダ処理などを行った後、システムコールによってユーザーランドにコピーされ、受信に利用した sk_buff は破棄される。

仮に複数のパケットが一度に届いた場合、キューに存在する有効なスロットの分だけパケットで埋めてから head を更新し、それらを一つずつ処理していく。全ての処理が完了してから、tail が更新される。

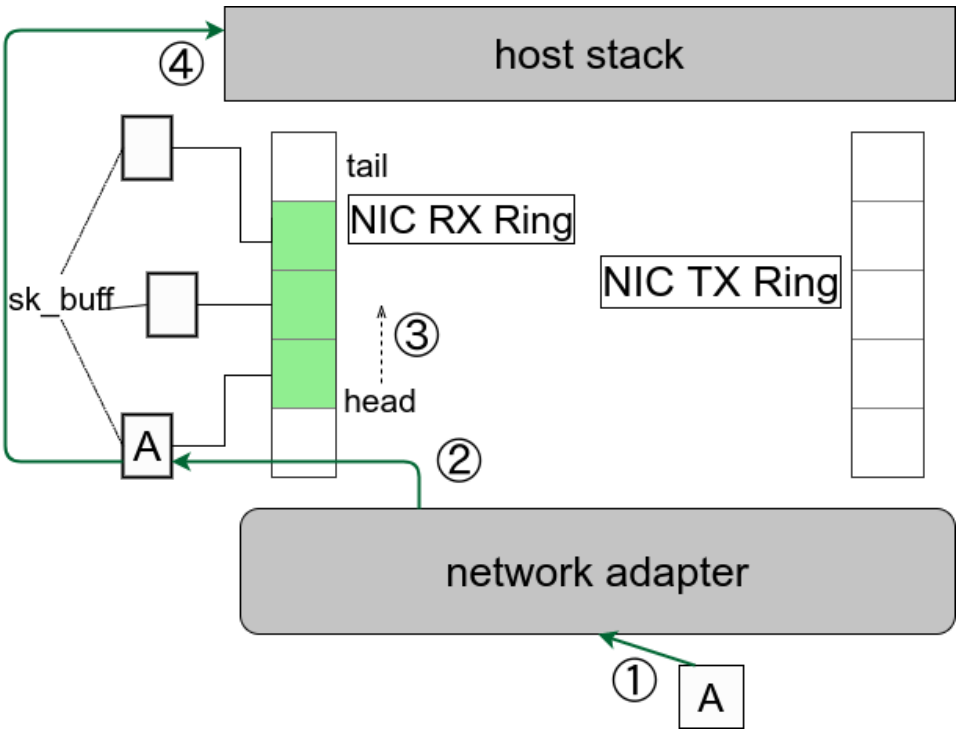


図 2.7: NIC 受信時の動き

(2) 送信時

続いて、送信時の動きについて、図 2.8 を用いて整理する。

初期状態では、TX リングは全て空の状態になっている。アプリケーションからソケットを経由して送信を指示されたデータは、CPU によって、プロトコル処理をされホストスタックへコピーされる (①)。そして OS はパケットデータがコピーされた領域を sk_buff 構造体として確保し、TX リングにリンクさせる (②)。追加された分、tail の位置を更

新される。NIC はパケットのデータを DMA によって読み込み、ネットワークへ送信される (③)。送信後、NIC は head の位置を更新し (④)、完了を通知した後、確保されていた sk_buff を解放する。

複数のパケットを処理する際は、受信時と同じように、1 パケット毎に送信の処理を行っていき、他のリング上のパケットは、送信完了の通知があるまで待機される。ただし、sk_buff の解放は一度に行われる。

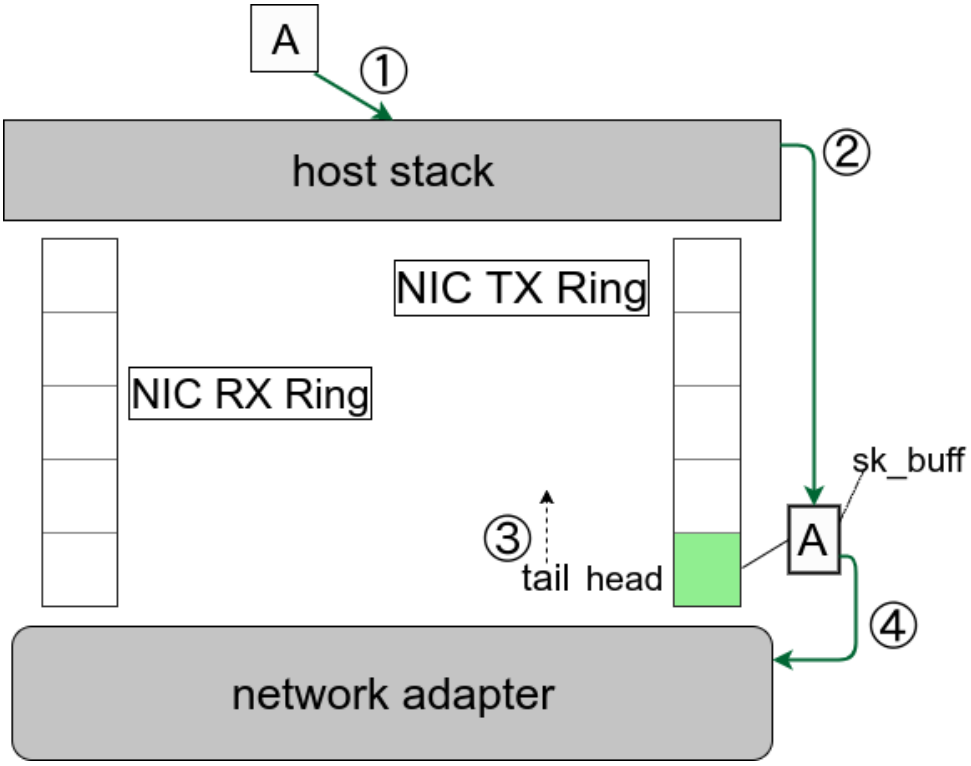


図 2.8: NIC 送信時の動き

このように、複数のパケットを処理する場合でも、一つのパケット毎に、データのメモリコピーのシステムコールを実行、パケットデータと sk_buff の領域の確保と解放といった処理を、割り込みを受けて行う必要がある。1GbE や 10GbE などの高速な通信に対応した現代のコンピュータでは、秒間でも大量のパケットを受け取ることができる。84byte のデータを 1GbE と 10GbE の NIC が、秒間に最大でどれだけのパケットを受信可能なのかを計算すると、1GbE の NIC で約 1.49Mpps、10GbE の NIC では約 14.88Mpps となる。

このような高いパケットレートでは、割り込み処理が CPU リソースを消費し、性能の低下や、通信の遅延がアプリケーションのボトルネックとなる可能性がある。

2.4.2 netmap によるパケット転送処理

続いて、netmap を用いたパケット転送処理について説明する。

まずは、netmap モードによって構成されるアーキテクチャについて、図 2.9 と共に説明する。デバイスを netmap モードでオープンすると、ホストスタックと NIC リングが切断される。そして共有メモリ上に、NIC リングの複製である netmap リングと、そのリング全てに対応した netmap バッファが確保される。そしてそれらの netmapr リングは、確保した netmap バッファにリンクされ、続いて NIC リングも同じバッファにリンクされる。

結果として、netmap バッファを介して、NIC リングと netmap リングが同一に保たれることになる。NIC は自身のリングと、netmap バッファのみに対してアクセスすることができる。そして netmap のプログラムは、netmap リングと対応する netmap バッファにのみアクセスでき、NIC リングを直接操作することはできない。

NIC 自体は、OS による処理と同じように、NIC リングの `head` と `tail` 間のスロットを使用可能とする。リングに対応しているのは netmap バッファだが、`sk_buff` と同じように扱えるようになっている。netmap リングは NIC のリングと同じようにそれぞれ `head` と `tail` のポインタを持ち、netmap のプログラムはこの区間内のリングとバッファに対してのみアクセスする。

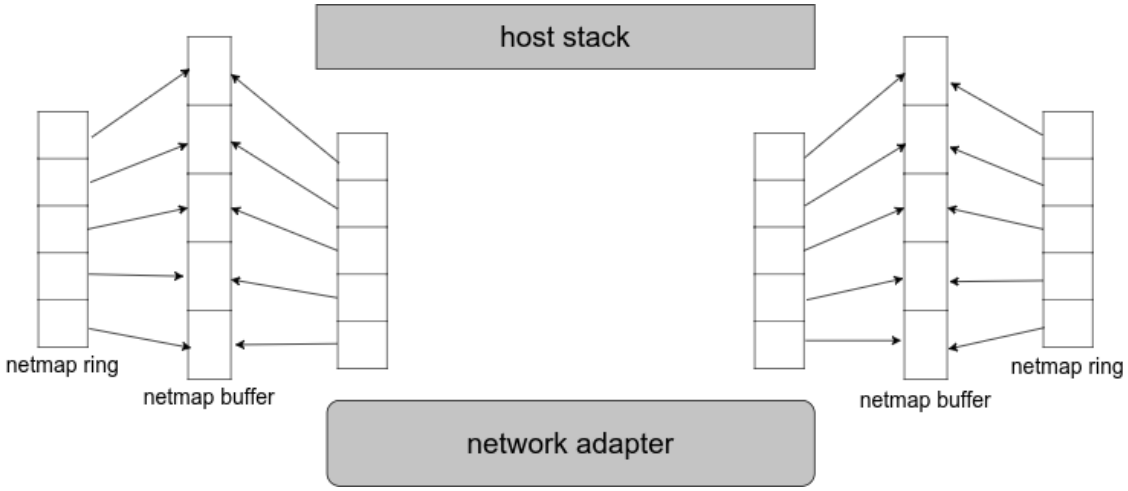


図 2.9: netmap モードのアーキテクチャ

(1) 受信時

実際の受信時の動きについて、図 2.10 と共に説明する。

NIC はパケットを受信すると (①)、RX リングから、対応した netmap バッファに対してパケットをコピーし (②)、netmap に対して通知を行う。これに応じて netmap は netmap リングの同期の要求を行い、それによって NIC リングの `head` 部分と (③)、netmap リングの `tail` の位置 (④) が更新される。

こうして head、及び tail 間にアクセス可能なスロットが生まれる。netmap のプログラムは netmap リングを通してバッファに記録されたパケットを読み取り (⑤)、その上で head を更新する (⑥)。そして再度リングの同期要求が行われ、NIC リングの tail の位置が更新される。

複数のパケットが届いた場合、その分 tail を更新される。そのパケットを netmap のプログラムが読んでいる間、NIC 側で新たなパケット受信の処理を行うことが可能である。この状態で netmap プログラムが再度リングの同期を行うと、netmap リングの tail と NIC リングの tail がそれぞれ更新される。

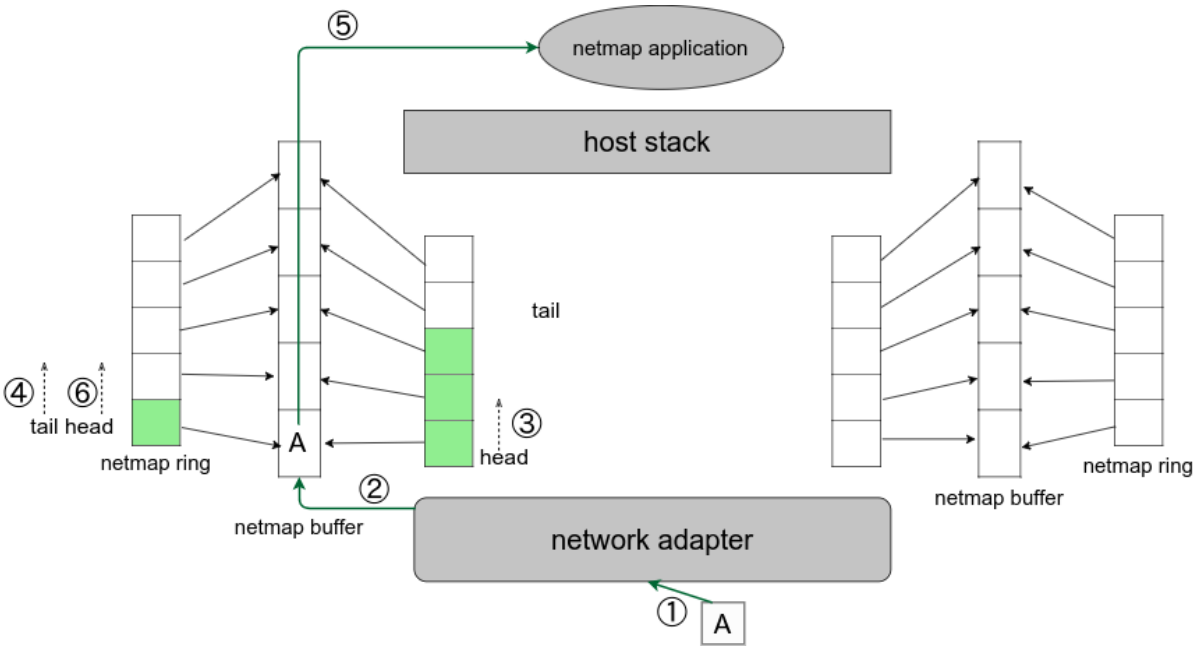


図 2.10: netmap モードにおけるパケット受信の流れ

(2) 送信時

続いて送信時の動きについて、図 2.11 と共に説明する。

netmap の TX リングは、head と tail の間の利用可能なスロットに対しその対応している netmap バッファに対してアクセスされる。NIC のリングには、最初は利用可能バッファは一つもない状態となる。

netmap のプログラムは、netmap リングを通して、バッファにパケットを追加していき (①)、それに応じて head の位置も更新される (②)。プログラムが同期の指示をだすと、カーネルは NIC リングの tail を更新して NIC に伝え (③)、パケットの送信を開始させる (④)。また、netmap リングの tail も更新される (⑤)。

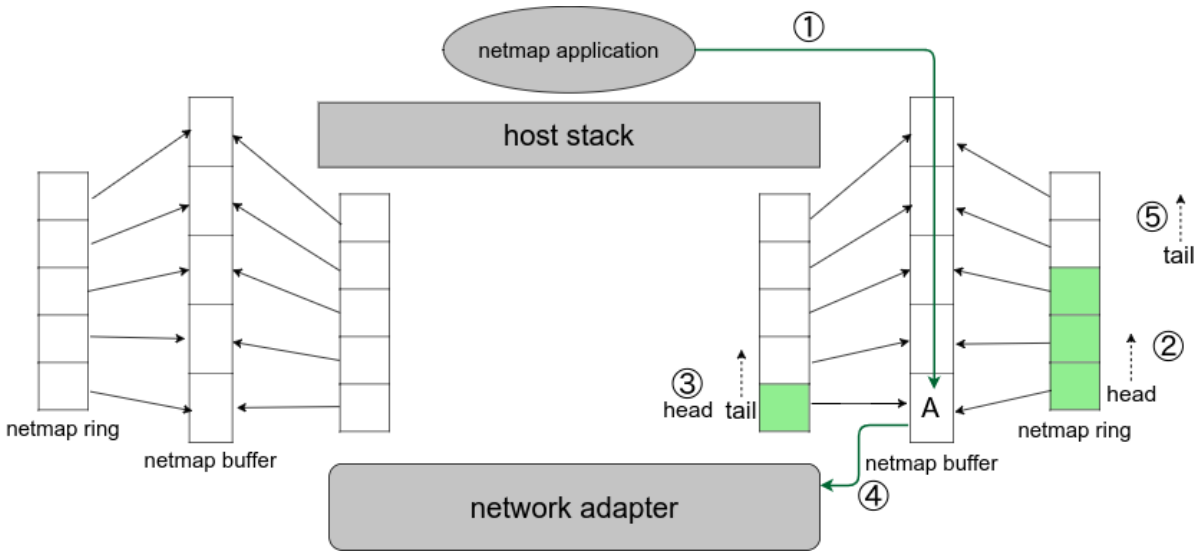


図 2.11: netmap モードにおけるパケット送信の流れ

以上のように、事前に対応する全てのバッファを共有メモリ上に確保することで、汎用 OS の Socket API を用いたパケット転送処理で問題となっていた、1 パケット毎のカーネル・ユーザ間のメモリコピーや、バッファの個別確保及び解放が必要なくなる。更に、パケット毎に行うプロトコルスタック処理などのカーネル内部の処理を迂回することになるので、ネットワーク I/O の性能を向上させることが可能となる。

2.5 Intel DPDK

Intel DPDK は、Linux OS 上で動作するドライバを含めたソフトウェアライブラリである。

netmap がネットワークスタックをユーザメモリ側で共有し、受信処理はカーネルで書き換えられた NIC ドライバを使用していたのに対し、DPDK はカーネルで NIC のドライバや、2.4.1 節で示したようなリングバッファの処理を、UIO と呼ばれるユーザスペースにドライバを作成する機能などを利用して特定の CPU コアを割り当ててエミュレートし、ポーリングにより受信データを監視している。パケットデータに直接アクセスできるためカーネル標準のプロトコルスタックを利用できる netmap と異なり、パケットデータを独自の構造体で管理し、専用の API でパケットの情報の取得や管理を行う。

2.6 fluentd

fluentd[14] とは、Treasure Data が開発するログの収集・集約・転送などを行う管理ツールである。オープンソースで公開されており、Linux など、各種 UNIX OS で動作する。

fluentd は、ファイル読み込みや外部からのログの受信などのイベントを INPUT として、

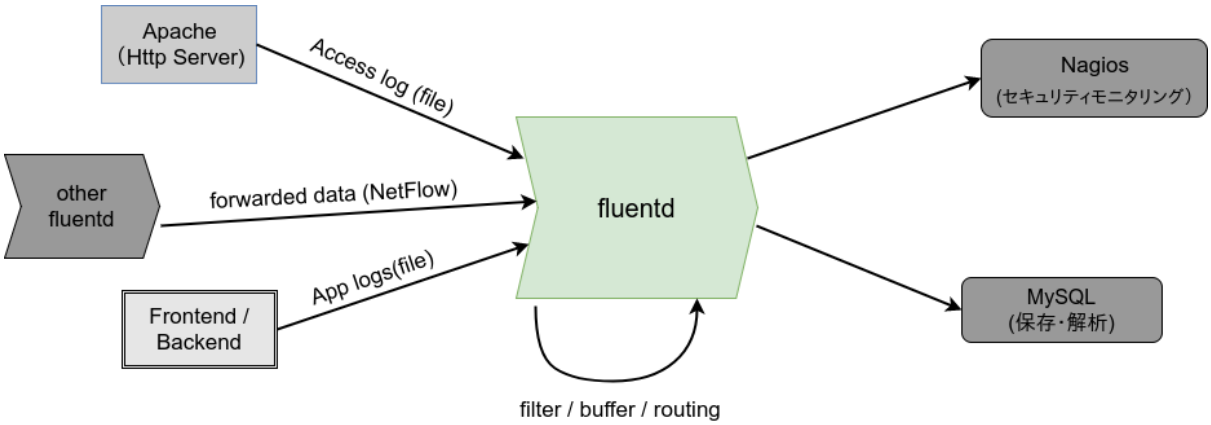


図 2.12: fluentd アーキテクチャ

Nagios[15] などのセキュリティモニタリングプログラムや、DB への保存、別ホストの fluentd への転送など、様々な出力先への出力イベントを OUTPUT として、それぞれプラグインで管理している。

そして、それらを組み合わせることによって、入力の条件にあわせてフィルタリング、バッファリングなどの処理や、保存、フォワーディングなどの出力をカスタマイズする事ができる。

2.7 UDP Director

UDP Director[16] とは、既存のログフォワーダの一種である。複数のインターフェースから送信されたログ情報を集約し、収集サーバへフォワーディングする手法として、Lancope 社が開発した。NetFlow や sFlow を初めとする、OS などのログメッセージの IP ネットワークで転送するための標準規格である「syslog[17]」や、ネットワーク機器の監視、制御を行う「SNMP[18]」などの複数の規格に対応している。syslog を、その収集・解析ツールの一つである「Kiwi Syslog Server[19]」へ送信したり、SNMP を監視を行うエージェントプログラムの一つである「Net-SNMP[20]」へ、状況に応じて転送される。

そして、接続された複数の Exporter 送信機器から送られた情報を集約して、単一のデータストリームにして Cisco のフロー情報収集 Collector である「Cisco Stealthwatch Flow Collector」へ転送することで、ネットワークの構造やフロー情報の収集を簡素化し、インフラ構築をより単純なものにすることができる。先に述べた「ShowNet」でも、膨大なログを収集するためにこのシステムが利用されていた [3]。

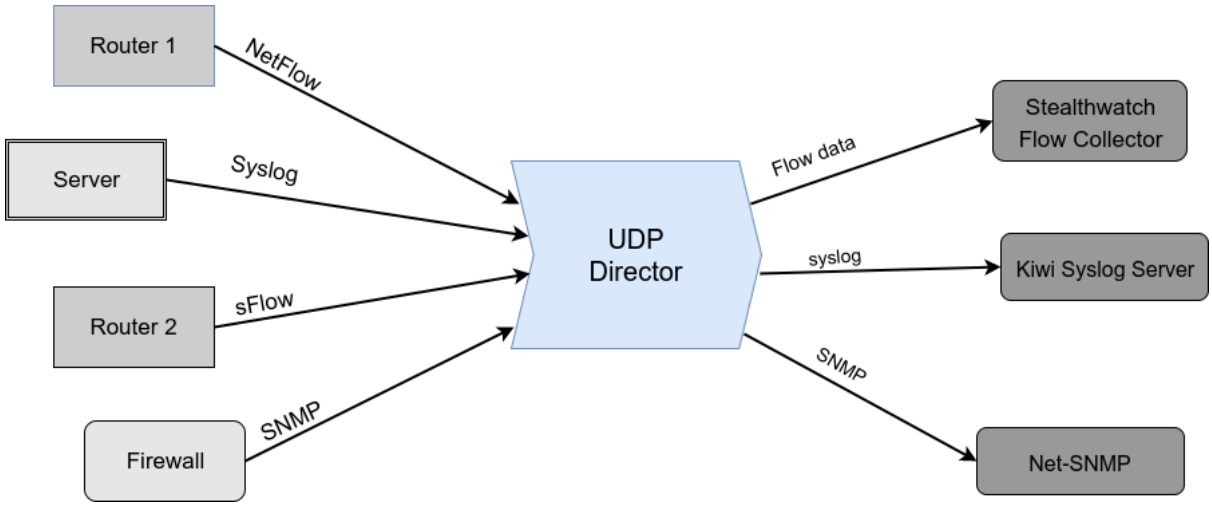


図 2.13: UDP Director

第3章　提案手法

本章では netmap の性能を予備実験で示した上で、本研究の提示する課題についてのアプローチを示す。

3.1　予備実験

実際に netmap を用いたパケットの転送で、どれくらいの速度が出るのかを計測し、従来の Socket API の速度と比較する。Socket API を利用した計測には、ネットワークベンチマークツールである `iperf` を利用する。

3.1.1　実行環境

送信、及び受信に用いるマシンの概要は表 3.1 の通りである。

表 3.1: 予備実験に用いるマシンの概要

	OS	NIC
送信	FreeBSD 11.0-RELEASE-p9	Intel 82599ES 10-Gigabit
受信	Fedora release 25	Intel 82599ES 10-Gigabit

3.1.2　実験結果

指定した回数、同じパケットを送信し続けるプログラムを作成して実行する。

表 3.2: netmap を用いたプログラムの計測結果

送信パケット数	パケットサイズ [bytes]	経過時間 [s]	スループット [pps]
1,000,000,000	84	124.20	8,051,805.64

続いて、ベンチマークツール `iperf` を利用して計測した結果を表 3.3 に記す。

表 3.3: iperf を用いた計測結果

送信パケット数	パケットサイズ [bytes]	経過時間 [s]	スループット [pps]
1,000,000,000	84	1050.3	998,643.81

スループットを比較すると、84bytes のパケットを送信する上での、Socket API を利用した通信は 998.64Kpps であるのに対し、netmap を利用した通信は約 8.05Mpps と、8 倍ほどの値が出た。以上のように、netmap を用いたプログラムは、Socket API と比較して非常に高いスループットを期待することができる。

3.2 本研究のアプローチ

3.1 節で示したように、netmap を用いることで、通常の Socket API と比較して高いスループットが得られることがわかった。本研究では、送受信に netmap を利用し、従来の手法で大きなボトルネックの一つとなっていた、汎用 OS におけるパケット受信及び送信処理におけるオーバーヘッドを改善し、パケットごとの処理時間を削減することで、高スループットのパケット転送処理を実現するログフォワーダ「gForwarder」を実装する [21]。

第4章　実装

本章では、ログフォワーダの構成について整理し、具体的な実装について説明する。

4.1　機能要件

ログフォワーダ「gForwarder」を設計する上で必要な事項について整理する。

- 稼働する OS は、BSD 系を想定する。
- UDP/IP プロトコルにのみ対応する。
- 送信元 IP、及び送信先ポートによって、宛先の Collector(アプリケーションサービス) を選定する。
- 送信先 IP など、必要に応じてプロトコルヘッダの書き換えを行う。

以上の条件で NetFlow パケットの受信及び転送を行うログフォワーダを実装する。また、実際にデータを転送する手順は以下のようになる。

1. 複数台の Exporter からの接続、及びパケット受信を行う。
2. 宛先の Collector を選定する。
3. 送信先 IP など、必要に応じてプロトコルヘッダの書き換えを行う。
4. 2 で記した条件に対応した一台以上の Collector へ向けて、受け取ったパケットをフォワードする。

大まかな流れは以上の通りである。複数の Exporter から送られてくる大量のパケットに対し、netmap によりパケット単位の処理時間を大幅に削減した状態での処理速度の向上を目的とする。

4.2　設計

4.2.1　構成

システム構成図を図 4.1 に示す。

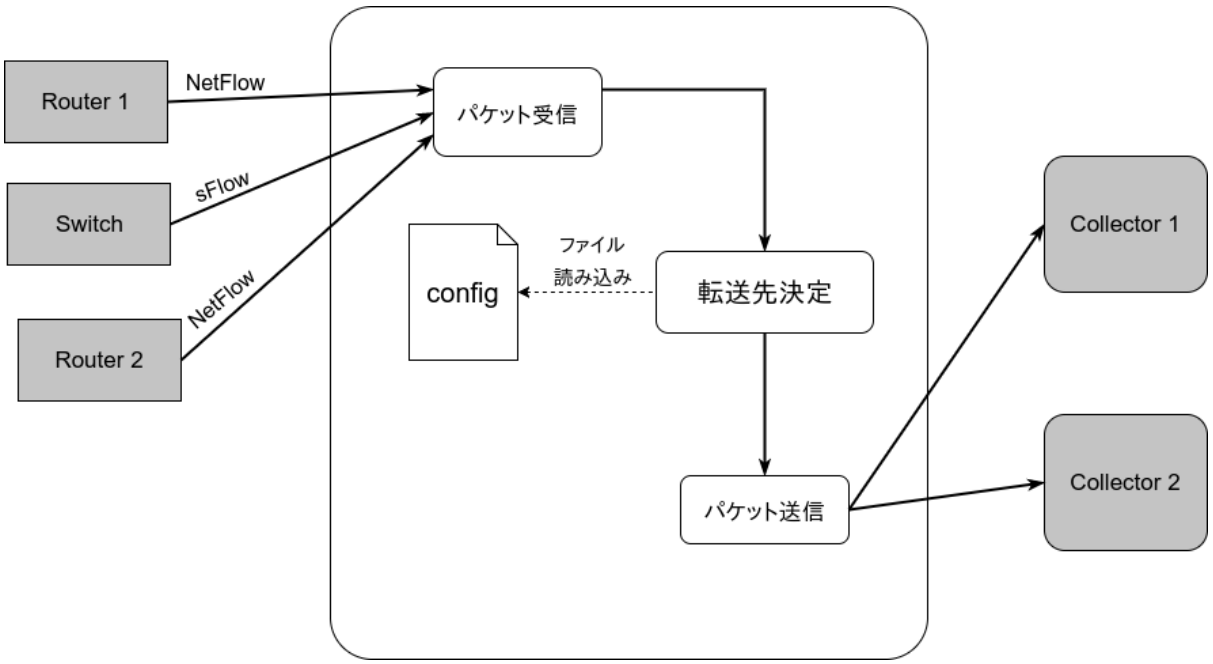


図 4.1: システム構成図

フォワーディングの宛先を決めるルールは、用意された設定ファイルを読み込むことで決定される。パケットの受信と送信には netmap を利用する。

4.3 実装

続いて 4.1 節で示したとおりの手順に従った動きを踏まえ、実行環境と実装の詳細について、具体的に整理する。

4.3.1 環境

まず、gForwarder が動作する環境について、簡単に表に示す。OS は、2011 年より正式に netmap がマージされた FreeBSD を用いて行う。

表 4.1: gForwarder の実行環境

OS	FreeBSD 11.0-RELEASE-p9 (GENERIC)
NIC	Intel 82599ES 10-Gigabit SFP/SFP+
言語	C
ライブラリ	netmap v11.3

実際の処理手順について、設定読込フェーズ、待機フェーズ、フォワードフェーズと、大きく 3 つの段階に分けて考える。

(1) 設定読込フェーズ

4.1 節で示したとおり、gForwarder は、パケットに付与された情報によって、転送する先の Collector を決定する。

そのため、gForwarder を起動する際に、事前に用意した設定ファイルを読み込ませ、その内容にしたがってフォワーディングのルールを設定する。設定に必要な項目は以下の表にまとめる。

表 4.2: 設定項目の種類

項目名	詳細
送信元アドレス	受信パケットがどこから送られたか
送信先ポート	宛先としてどのポートが指定されているか
送信先アドレス	送信先 Collector のアドレス。複数記述可能
優先度	ルールの重複時、優先される度合い

フォーマットは、一行に一つのルールを記述し、項目はスペースで区切る。

送信元アドレスと送信先ポートにおいてはワイルドカードを指定することができ、また仮に、優先度が同等のルール内で衝突が起きた場合は、後に記述されたルールが優先される。その他に、全てのルールに対応しない場合のデフォルトルールも存在する。

設定ファイルのサンプル

```
10.2.2.3 8086 10.2.2.3 A
10.2.2.5 9996 10.2.2.3 A
10.2.2.5 8086 10.2.2.4 A
10.2.2.3 9996 10.2.2.7 B
```

上記のルールを保存するための構造体として、一つのルール情報を保持する構造体 `rule_dic` と、その `rule_dic` をまとめた構造体 `rule_box` を宣言する。それぞれの構造を表 4.3、4.4 に示す。

設定読込フェーズでは、指定された設定ファイルを読み込み、各行のルールを `rule_dic` の形にした上で `rule_box` の `rule_dics` に保存する。ルールの総数は `rule_num` に保存する。

表 4.3: 構造体 rule_dic

struct in_addr	srcaddr	送信元アドレス
int	dstport	宛先ポート
struct in_addr	dstaddr	対応する宛先アドレス
char	priority	ルールの優先度

表 4.4: 構造体 rule_box

struct rule_dic[ ]	rule_dics	rule_dic の配列
int	rule_num	ルールの総数

(2) 待機フェーズ

設定の読み込みと反映の完了後、パケットをフォワードするための待機段階に入る。今後はパケットが届くたびにフォワードフェーズを行い、受け取ったパケットを転送する。

パケットが届くまでの待機にはシステムコール poll を用いる。poll は指定されたファイルディスクリプタが I/O イベントを実行可能になるまで待つ命令である。受信可能になった時に動作を行い、それまで他の処理をブロックすることがないため、複数のパケットを同時に扱える。

(3) フォワードフェーズ

パケットが届くたびにそれを転送するための処理が行われる。この処理は、ルール探索とパケット送信の 2 段階から成る。

ルール探索段階では、受信したパケットの IP ヘッダ及び UDP ヘッダから、送信元アドレス及び送信先ポートを読み取り、対応するルールから一致した宛先のアドレスを割り出す動作である。(1) 小節で取得したルール群から、送信元 IP アドレスと送信先 UDP ポートが一致するルールを探索する。どのルールにも一致しない場合、そのパケットは破棄される。

一致するルールが存在した場合、そのパケットの宛先 IP アドレスを、一致した rule_dic の dstaddr に書き換える。宛先アドレスを書き換えた後、そのパケットを TX リングに移動させる。2.4 項にて示したように、netmap のリングは全てが共有メモリ上に確保されているため、メモリコピーの必要はなく、zero-copy による移動が可能である。パケット受信と同じく、送信にも poll を用いるため、送信を待っている間に他の受信パケットの処理を行える。

第5章　評価

本章では、実装した gForwarder を用いた評価の形について、その想定される構成に触れつつ述べ、実験結果についてまとめる。その後実際に得られた値を既存手法と比較し、成果物の優位性について検証する。

5.1　評価手法

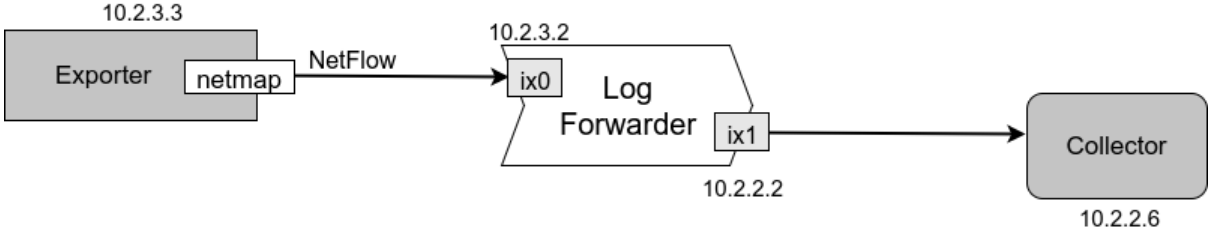
本検証では、gForwarder が、1.2 節で示した本研究の問題である大量のトラフィックが発生している状況で、通常の Socket API を用いたログフォワーダと比較してより高いパケットレートを維持でき且つ、全てのパケットを捌き切れるのを証明することを目的としている。

そのため、まず大量のトラフィックが発生することを想定とした環境を構築する。まず、パケットを送信する Exporter となる 10GbE の NIC が搭載されたマシンを用意し、netmap を用いた高スループットな NetFlow パケットを送信し続ける NetFlow パケットジェネレータを作成する。

事前に入手した NetFlow パケットのキャプチャファイルからパケットを取り出し、それを複製して送信していく。送信する部分は 3.1 節で用いたプログラムを利用する。

続いて Collector となるマシンを用意する。スループットやパケット受信数を計測するために netmap をベースにしたパケット I/O ベンチマークツールである `pkt-gen` を使用する。最後に、ログフォワーダを 1.3 節の図 1.2 で示した構成にしたがって、先程構築した Exporter と Collector にそれぞれ図 5.1 のように接続する。

図 5.1: 評価に用いるシステム構成図



Exporter から送信されたパケットをログフォワーダが受け取って選定し、Collector へ

向けてパケットを転送する。転送のルールは表 5.1 の通りとする。

表 5.1: 評価に用いるログフォワーダのフォワーディングルール

src IP	dst Port	dst IP
10.2.3.3	8081	10.2.2.6
10.2.3.3	9996	10.2.2.6

この状態で netmap を利用した Exporter からの高スループットなトラフィックを受けて、ログフォワーダがパケットを破棄せず受信処理できた割合と、スループットを計測する。

5.1.1 評価項目

(1) スループット

フォワーディング時のスループットを記録する。Collector のマシンに単位時間あたりどれくらいの数のパケットが送られたかをカウントしてパケットレートを算出する。計測には `pkt-gen` を利用し、図 5.1 で示した構成のシステムを用い、Exporter から netmap を用いて一定時間 NetFlow パケットを送信し続け、Collector で受信時のスループットを計測する。計測の基準となる、理論的最大値のパケットレートを計算する。パケットレートの計算はデータリンク層を含めた Ethernet フレームサイズで計算するため、パケットサイズに加算を行う。今回利用するパケットのサイズは 174bytes であった為、フレームの内訳は表 5.2 の通りとなり、フレームサイズは 212bytes となる。フレームサイズから、10GbE における最

表 5.2: Ethernet frame 内訳 [22]

Interframe gap	12 bytes
MAC preamble	8 bytes
MAC header	14 bytes
payload size	174 bytes
Ethernet CRC	4 bytes

大パケットレートを算出すると、$(10*10^9)bps \div (212bytes*8) = 5,896,226pps \fallingdotseq 5.90Mpps$ となる。

(2) 受信率

送られてくるパケットの内、破棄されずに処理された割合を計測する。パケットの送信には `pkt-gen` を利用して複数のパケットレートの中で測定を行い、受信の際のパケットレー

トと比較し、その割合を求める。ペイロードのサイズは MTU の最大値である 1,500bytes に設定し、1,000pps から、想定される最大のパケットレート間における計測を行う。表 5.2 から、Etherframe のサイズは 1,538 bytes となる。

10Gbps の通信速度における最大パケットレートは、このサイズを用いて $(10 * 10^9) bps \div (1,538 bytes * 8) = 812,744 pps$ で求められる。

比較対象として、fluentd を用いて受信したパケット数を用いる。fluentd には、tagomoris 氏の fluentd 用プラグインである fluent-plugin-datacounter[23] をインストールし、表 5.1 で記述されたものと同じルールに対応した、一定時間毎のパケット数をカウントすることでパケットレートを測定する。

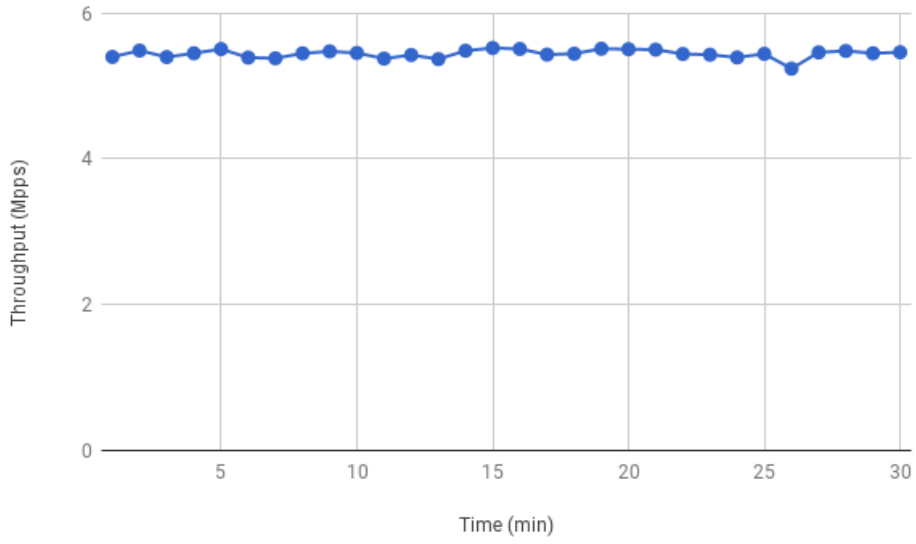
5.2 実行結果

5.2.1 スループット

まず、スループットの算出結果を図 5.2 に示す。

横軸が、送信後何分経過したかを示し、縦軸にスループットを示している。

図 5.2: gForwarder のスループット



グラフから、gForwarder によるフォワーディングが一定して非常に高いスループットを得られ、ある程度の時間経過の中で大きい変動のない安定した挙動をしていることがわかる。30 分間継続して計測を行い、平均値を出した結果、5.44Mpps という結果になった。最大パケットレートである 5.90Mpps と比較すると、92.20%とほぼ最大パケットレートに近いスループットを得ることができた。

5.2.2　受信率

送信側のパケットレートと gForwarder 及び fluentd での受信側のパケットレートの比較を、表 5.3、表 5.4 に示す。この 2 つの表の結果をまとめたグラフを、図 5.3 に示す。

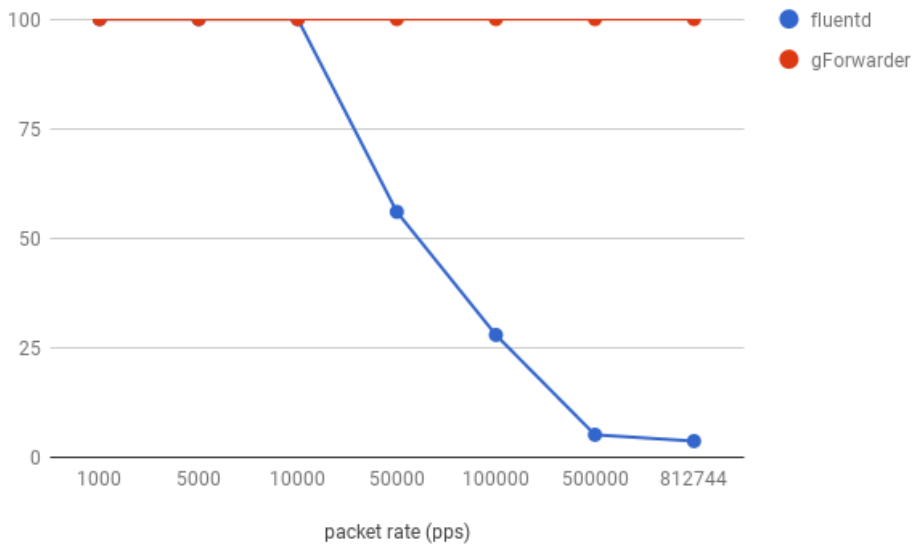
表 5.3: fluentd のパケットレートと受信率

送信パケットレート (pps)	受信パケットレート (pps)	受信率 (%)
1,000	1,000	100.00
5,000	5,000	100.00
10,000	9,999	99.99
50,000	28,005	56.01
100,000	27,928	27.93
500,000	25,741	5.15
812,744	30,118	3.71

表 5.4: gForwarder のパケットレートと受信率

送信パケットレート (pps)	受信パケットレート (pps)	受信率 (%)
1,000	1,000	100.00
5,000	5,000	100.00
10,000	10,000	100.00
50,000	50,000	100.00
100,000	100,000	100.00
500,000	500,000	100.00
812,744	812,744	100.00

図 5.3: ログフォワーダのパケット受信率



fluentd は、パケットレートが 50,000pps を越えてから受信率が下がり始め、最大パケットレートとなる 812,744pps では 3.71%とほぼ全てのパケットが破棄されたのに対し、gForwarder は一貫して 100%を維持し、gForwarder が大量のトラフィックを捌くことができることが証明された。

5.3 その他の既存手法との比較

??項で紹介した、実際に製品として利用されている UDP Director について、公開されているスループット [16] を用いて、表 5.5 にて比較を行う。

表 5.5: UDP Director と gForwarder のスループット比較

	gForwarder	UDP Director 2010
INPUT	5,336,287 pps	37,500 pps
OUTPUT	5,401,010 pps	75,000 pps

UDP Director の値と比較して、INPUT では 142.30 倍、OUTPUT では 72.01 倍のスループットが得られた。以上の結果から、今回の実装は既存研究と比較しても大きな優位性を出せることがわかった。

5.4 考察

netmap を用いた gForwarder は、非常に高いパケットレートを維持することができ、通常の Socket API を用いたログフォワーダでは捌ききれないトラフィックに対しても、ほぼ 100%の処理率を示すことができた。したがって、通常の Socket API を用いたログフォワーダと比較して、優位性を認めることができた。

従来のメモリコピーなどのコストが存在する Socket API に比べ、netmap を用いたパケット転送処理がより効果的であった為と考えられる。本研究の成果を、高いパケットレートが予想されるネットワーク内で用いることで、ログの管理が効率化されることが期待される。

しかし、今回の実験結果は、送受信先が限られ、ルール数が非常に少ない状態で、宛先のアドレスも一つのみに固定であったため、より多くのルールが存在していたり、ホップ数の多い複雑なネットワークに接続された状態で実験を行った際の懸念は残る。また、gForwarder はログの受信パケットレートだけでなく、送信パケットレートも高いため、受信側となる Collector も、パケットレートの多い機種を選ぶ必要がある。

第6章　結論

6.1　本研究のまとめ

複雑化したネットワーク上でログを効率的に管理するための手段として、ログデータの集約、分配を行うログフォワーダという仕組みが存在している。しかし、ログデータの一種である、ネットワークのフロー情報などを記録する NetFlow は、CGN などの大規模ネットワークにおいてはセッションログとしても扱われるため、膨大な量のデータが発生し、ログフォワーダがトラフィックを捌き切れなくなるという性能的な問題が想定される。

本研究では、netmap と呼ばれる高効率なパケット I/O フレームワークを利用し、OS におけるパケット転送処理を効率化することで、膨大なトラフィックをさばくことができるという仮説の元、高スループットに対応したログフォワーダ「gForwarder」を実装し評価した。

評価手法として、10GbE の NIC を用いて構築した簡易ネットワーク内で、netmap を用いて擬似的に高いトラフィックを発生させ、その中でのフォワーディングを行った際のスループットを計測した。また一般に利用されているフォワーディングアプリケーションである fluentd と比較して、どれだけのパケット数を破棄せずに処理できるかを計測した。

結果として、フォワーディング時に理論的最大パケットレートの約 96%という非常に高いスループットを得ることができ、また、fluentd の受信率が最終的に 3.7%になる 812,744pps でも継続して受信率 100%を得ることができた。また、実際に製品として利用されている Cisco の UDP Director の仕様書にて公開されているスループットとの比較も行い、INPUT で 142 倍、OUTPUT では 72 倍のスループットを出すという結果を得ることができた。

6.2　本研究の結論

本研究では、ネットワーク内のログデータを効率的に運用するための仕組みであるログフォワーダに対し、発生することが想定される従来の Socket API を用いた仕組みでは対処することが困難となる膨大なトラフィックに対し、netmap を用いてパケット転送処理を効率化する手法を提案し、その有効性を検証した。その結果、netmap を用いること

によって、高いスループットに対応し、パケットをほぼ破棄することなく処理することができることがわかった。

例外の少ないルーチンワークが主となるログフォワーダのような仕組みでは、従来のメモリコピーなどのコストが存在する Socket API を利用するより、netmap を用いる事の有用性を示すことができた。しかし、INPUT だけでなく、OUTPUT も高スループットで出力するため、最終的にパケットを受け取る Collector となるマシンにも高スループットの INPUT が求められてしまうという問題も想定される為、運用していく上で注意が必要である。加えて、評価を行う上で必要な最小限の機能のみを実装したため、ARP のアドレス解決や、名前解決などの機能が存在しなかった。そのため今回の成果物は実用性自体は高くなく、また実際にこれらの機能を実装に加える場合、処理の遅延によるスループットの低下なども考える必要がある。

6.3 今後の展望

6.3.1 機能性の充実

今回の評価では、宛先となる Ethernet アドレスがすでに確定していたり、宛先となるマシンが絞られていたことも在り、現時点で gForwarder はより広大なネットワーク内で用いられる技術である DNS 解決や ARP 解決、ICMP や、RARP などの異なるプロトコルなどに対応していない。実際の運用で用いられることを想定するために、対応させなければいけない技術が多く存在している。また、項目数を増やすなどして、ルーティングに用いる設定ファイルの自由度を上げたり、設定の変更をリアルタイムで反映できるようなユーザビリティ性の向上、スループットやパケット受信数などのリアルタイムの可視化などといった機能性の充実が、課題として挙げられる。

6.3.2 ネットワークの発展

1.1 節で述べたように、ネットワークの発展やハードウェアの進化は今後も続いていく事が想定されるため、更に高いパケットレートが発生する可能性もある。そのため、高スループットに対応し続けるためには、今後も継続的に最新の技術に対して確認を行っていく必要がある。

謝辞

本論文の執筆にあたり、ご指導いただきました慶應義塾大学 環境情報学部教授 村井純博士、同学部教授 中村修 博士、同学部准教授 Rodney D. Van Meter III 博士、同学部准教授 三次仁博士、同学部准教授 楠本博之博士、同学部准教授 植原啓介博士、同学部准教 授 中澤仁博士、政策・メディア研究科特任准教授 鈴木茂哉博士、SFC 研究所 上席所員 (訪問) 斉藤賢爾博士に感謝致します。

研究について日頃からご指導頂きました政策・メディア研究科博士課程 松谷健史氏、政策・メディア研究科特任助教 空閑洋平氏に感謝致します。両氏には、研究室に所属したばかりの頃から本研究に至るまで、ご多忙の中、研究者として多くの面で未熟であった自分を見捨てることなく、技術面やストーリー面で絶えず多くのご指導をいただきました。両氏のご指導無くして本研究を卒業論文としてまとめることはできませんでした。重ねて感謝申し上げます。

ログデータの取扱いや、ログフォワーダの仕組み、ShowNet の紹介など、本研究のテーマを決める上で重要な助言を頂いた東京大学助教の中村遼氏に感謝致します。また、NetFlow を始めとするトラフィックに関連する技術についての知識を提供して頂いた政策・メディア研究科修士課程 鈴木恒平氏に感謝致します。

研究室を通じた生活の中で多くの示唆を与えてくれた河口綾摩氏、東海林晃氏、豊田安信氏、鎧坂文菜氏、尾崎周也氏、重田桂子氏、押見太雄氏、桑原誠尚氏、Arch 研究グループの皆様、及び村井・徳田・楠本・中村・高汐・重近・バンミーター・植原・三次・中澤合同研究プロジェクトの皆様に感謝致します。最後に、私の研究を支えてくれた、両親、妹、祖母をはじめとする親族、多くの友人・知人に感謝し、謝辞と致します。

参考文献

[1] ntop. https://github.com/ntop. last visited on 2018-01-09.

[2] Cisco stealthwatch enterprise. https://www.cisco.com/c/ja_jp/products/security/stealthwatch/index.html. last visited on 2018-01-10.

[3] Interop Tokyo ShowNet NOC Team. Shownet2017 report. https://www.slideshare.net/InteropTokyo-ShowNet/shownet2017-report-83974680. last visited on 2017-12-19.

[4] Netstream tecnology white paper. http://enterprise.huawei.com/ilink/enenterprise/download/HW_201022. last visited on 2018-01-10.

[5] Netflow - wikipedia. https://en.wikipedia.org/wiki/NetFlow. last visited on 2017-12-9.

[6] Cisco ios netflow data seet. https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/product_data_sheet0900aecd80173f71.html. last visited on 2018-01-03.

[7] Flowscan architecture. https://www.caida.org/tools/utilities/flowscan/arch.xml. last visited on 2018-01-18.

[8] Netflow version 9 フォーマット. https://www.ibm.com/support/knowledgecenter/ja/SSCVHB_1.2.1/collector/cnpi_netflow_v9.html. last visited on 2018-01-20.

[9] sflow. http://www.sflow.org/. last visited on 2018-01-19.

[10] IPFIX overview. https://www.ibm.com/support/knowledgecenter/en/SSCVHB_1.2.1/collector/cnpi_collector_IPFIX_overview.html. last visited on 2018-01-07.

[11] Luigi Rizzo. netmap: A novel framework for fast packet i/o. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pp. 101–112, Boston, MA, 2012. USENIX Association.

[12] Giuseppe Lettieri. An introduction to netmap - events. https://conferences.sigcomm.org/sigcomm/2017/files/tutorial-netmap/01-intro.pdf. last visited on 2017-11-30.

[13] IBM - linux ネットワーク・スタックの徹底調査. https://www.ibm.com/developerworks/jp/linux/library/l-linux-networking-stack/index.html. last visited on 2018-01-10.

[14] fluentd. https://www.fluentd.org/. last visited on 2017-12-30.

[15] Nagios. https://www.nagios.org/. last visited on 2018-01-17.

[16] UDP director. https://www.lancope.com/sites/default/files/udp-director-data-sheet-eMark.pdf. last visited on 2017-12-19.

[17] The syslog protocol - rfc 5424. https://tools.ietf.org/html/rfc5424. last visited on 2018-01-10.

[18] Simple network management protocol - rfc 1157. https://datatracker.ietf.org/doc/rfc1157/. last visited on 2018-01-18.

[19] Kiwi syslog server. https://www.jtc-i.co.jp/product/kiwisyslogserver/kiwisyslogserver.html. last visited on 2018-01-10.

[20] Net-snmp. http://www.net-snmp.org/. last visited on 2018-01-19.

[21] Phi-phi/gforwarder - github. https://github.com/Phi-phi/gForwarder. last visited on 2018-01-18.

[22] The calculations: 10gbits/s wirespeed - netoptimizer. http://netoptimizer.blogspot.jp/2014/05/the-calculations-10gbits-wirespeed.html. last visited on 2018-01-18.

[23] tagomoris/fluent-plugin-datacounter - github. https://github.com/tagomoris/fluent-plugin-datacounter. last visited on 2018-01-15.