博士論文 2019年度

コンテナ型NFVアーキテクチャ

慶應義塾大学大学院政策・メディア研究科
堀場 勝広

概要

本研究は，コンテナ型 Network Function Virtualization(NFV) アーキテクチャを提案し，仮想マシン (VM) を利用した NFV の従来の実装では困難だった，可搬性とパケット処理性能の両立を実現した．

NFV とは，本来専用のハードウェアで実装されてきたネットワーク装置を，汎用的なハードウェアとソフトウェアによって仮想化されたネットワーク機能 (VNF) に分離する技術体系である．この技術体系は多様化し複雑化するネットワークのトラフィックに対応するために，ネットワークの拡張性と柔軟性に関する課題を解決する技術として期待されている．一方で，インターネットにおけるアプリケーションの多様化に伴い，通信品質への要求が多様化した結果，その要求に最適化されたネットワークを動的に提供することを目指した「ネットワークスライシング」という概念が提唱されている．しかし，その実現には NFV，Software Defined Networking(SDN)，クラウドコンピューティングなど複数の技術を組み合わせて利用されることが想定されているが，具体的な実装方法は明らかにされていない．

本研究では，まず，ネットワークスライシングの実現に向けて，アプリケーションの要求に応じて動的にその構成要素を供給する Platform as a Service(PaaS) の概念をネットワークに適用したネットワークサービスの PaaS 化を提唱した．そして，その要件として VNF の最適な配置を保証する「可搬性」が重要な要素であることを明らかにした．次に，ネットワークサービスの PaaS 化の実現に向けた産学連携コンソーシアムを設立し，多くのベンダで独自に実現されている多様な NFV 関連製品を共同的に検証し，VNF の可搬性とパケット処理性能を網羅的に評価した．さらに，その経験と知見に基づき，実運用としての大規模な実ネットワークの構築・運用の実証実験をおこない，その相互運用性と性能を評価した．その結果，VNF の動作環境に VM を前提とした従来の NFV では，VNF の可搬性とパケット処理性能の両立に課題があることを明らかにした．

これらの評価と分析に基づいて，本研究では次に，可搬性とパケット処理性能を両立するコンテナ型 NFV アーキテクチャを提案した．提案したアーキテクチャでは，VNF をコンテナ，すなわち OS におけるプロセスの集合として扱い，VNF のネットワーク I/O インターフェースを POSIX API で抽象化することで，高いレ

ベルの相互運用性を実現し，VNF の可搬性を確保することができた．その結果，ボトルネックとなっていた汎用 OS が持つカーネルのネットワークスタックをバイパスすることで，VNF のパケット処理性能を向上することが可能となった．更に，提案したコンテナ型 NFV アーキテクチャの有用性を実証するため，VNF のネットワーク I/O に Netmap を利用したプロセスとして実装した結果，要求される可搬性を担保した従来方式と比較して，VNF の種類に応じて約 2 から 5.5 倍のパケット処理性能を達成し，可搬性とパケット処理性能の両立を実証した．

本論文の研究は，ネットワークスライシングの概念の実現に向けた課題を明らかにし，その課題の解決のための具体的な方法を提示した．自動運転や遠隔医療をはじめとした高度で多様な通信品質を要求するさまざまなアプリケーションが共通のプラットフォーム上で実現できる道を切り拓いた．

キーワード：**NFV**，ネットワーク仮想化，通信事業者，インターネット

指導教員

主査
村井 純 (慶應義塾大学)
副査
中村 修 (慶應義塾大学)
楠本 博之 (慶應義塾大学)
関谷 勇司 (東京大学)

Abstract

This dissertation proposes the Container-based "Network Function Virtualization (NFV)" Architecture. It realizes high-performance in packet processing while keeping the portability of network functions that is difficult to achieve by the conventional Virtual Machine (VM)-based NFV implementation.

NFV is a technology to virtualize the network functions so that they (Virtualized Network Functions, or VNF) can run on the general-purpose computer hardware instead of the traditional function-dedicate equipment. NFV is a pivotal technology to provide the network extensibility and flexibility to support more diverse and complicated network traffic expected in the future. On the other hand, the network quality required by the applications on the Internet becomes more diverse due to the expansion of the usage of them. To dynamically provide the optimized network functions for each application, a new concept called "Network Slicing" was proposed. The technology components to implement this concept includes NFV, Software Defined Networking (SDN), and cloud computing. However, the methodology of implementation using those components has not been yet clearly designed.

First, as an essential concept to implement the "Network Slicing," this research proposes a new idea of "Network as a Service" that dynamically provides the optimized network functions to the application, while a similar concept "Platform as a Service (PaaS)" dynamically provides the necessary resources to the application. This thesis discusses the importance of "portability" that can guarantee the optimized VNF allocation.

Secondly, to make the "Network as a Service" into reality, I contributed to establish an industry-academic consortium to assess the feasibility of existing NFV products from the various vendors. The consortium played a vital role to conduct a comprehensive evaluation and analysis of each product as well as the wide-range of the experiment in the real network environment jointly among various vendors. The result identified the major problem in the traditional NFV that runs VNF on VM, the difficulty of pursuing the high-performance in the packet processing while

keeping the portability of VNF.

As a solution to this problem, this thesis proposes the "Container-based NFV Architecture." This architecture enables both high-performance in packet processing and portability at the same time. In the proposed architecture, a VNF is a container, a cluster of processes in the Operating System. Also, the network I/O interface is abstracted by POSIX API to realize high-level interoperability and portability of VNF. This architecture enabled to increase the packet processing performance of VNF by bypassing the bottleneck staying in the network stack of the general OSs. Lastly, the proposed "Container-Base NFV Architecture" is validated by a sample implementation of a VNF process using Netmap for VNF's network I/O. The result shows 2 to 5.5 times higher performance of the packet processing, comparing with the traditional implementation given a similar portability requirement.

This research identified the problems in realizing the concept of "Network Slicing" and proposed a new methodology to solve them. It will contribute to establishing a future network platform that can support diverse and advanced quality requirements. Then the platform will make a variety of applications such as auto-driving and remote-medical-service reality.

Keywords: NFV, Network Virtualization, telecom carriers, the Internet

Advisory Group

supervisor
Jun Murai (Keio University)
co-advisor
Osamu Nakamura (Keio University)
Hiroyuki Kusumoto (Keio University)
Yuji Sekiya (The University of Tokyo)

目 次

図 目 次

表 目 次

第1章　序論

1.1　Network Function Virtualization(NFV)

近年，通信事業者におけるネットワークの設備と運用にかかる費用が収入の成長を上回る勢いで増大している．図 1.1 に通信事業者における費用と収入の変遷を示す[1]．スマートフォンの普及に伴い，トラフィックが音声通話からデータ通信が中心となった結果，通信事業者が提供するサービスが，従量課金の音声通信から，定額課金のデータ通信にシフトし，ストリーミングサービスを中心とした Over The Top(OTT) の台頭によるトラフィックの大容量化，モバイルの普及による端末数の増加，ゼロレーティングやペアレンタルコントロールなどによる運用の複雑化によって，ネットワークの設備や運用にかかる費用が飛躍的に増大した．その結果，近い将来において事業の継続性を失う恐れがあり，ネットワークの設備と運用にかかる費用の削減は緊急の課題である．

このような課題の技術的な原因の一つは，ネットワークの構成と運用における柔軟性の欠如である．多くのネットワーク装置は，専用のハードウェアで実装されており，Network Function(NF) と呼ばれる何らかのパケット処理をするネットワーク機能は，ネットワーク装置のハードウェアと密結合である．そのため，昨今のアプリケーションの多様化に伴ってトラフィックの傾向変化が激化したことによって，ネットワーク装置のライフサイクルは短くなってきている．また，実際の運用では，Service Function Chaining(SFC)[2, 3] と呼ばれる複数のネットワーク機能を連結することによってネットワークサービスを構成する．ネットワーク機能の連結は，物理的なネットワーク装置の接続によって実現されている場合が多く，ネットワーク機能の組み合わせや順序を変更するには，物理構成の変更を伴う複雑なオペレーションが必要になる [4]．その結果，ネットワークの設備と運用にかかる費用は肥大化している．

このような状況を解決する技術として，NFV(Network Function Virtualization)[5, 6] が提案されている．図 1.2 に ETSI(European Telecommunications Standards Institute) が打ち出した NFV のコンセプトの概要を示す．NFV では，これまでネッ

[1] 引用元の図を著者が和訳.

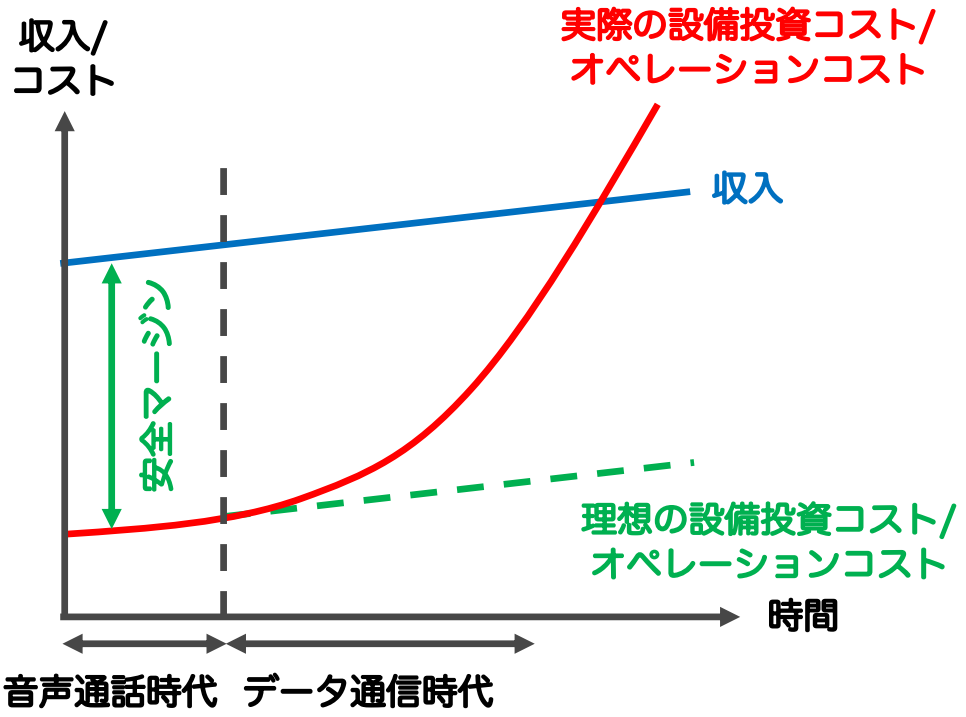


図 1.1: ネットワークの設備と運用にかかる費用と収入の関係性 [1]

トワーク装置は専用ハードウェアと密結合として実現されていたネットワーク機能を，汎用的なハードウェアで動作する仮想化されたソフトウェアとして取り扱う．これらのソフトウェアは，Virtual Network Function(VNF) と呼ばれる．ネットワーク機能をハードウェアから分離することにより，ネットワーク装置におけるハードウェアを再利用可能にし，ライフサイクルを長く保つことができる．また，ネットワークの構成をソフトウェアによって柔軟に変更できるようになり，特に SFC においては Software Defined Networking(SDN)[7] と連携し，VNF の組み合わせや連結の順序が動的に変更可能になる．NFV は，通信事業者の課題を解決することが可能な新しい技術として期待されている [8].

1.2　次世代のネットワークサービスプラットフォーム

クラウドコンピューティングの普及に伴い，アプリケーションのアーキテクチャや，展開・運用のプロセスが変化した．近代的なアプリケーションは，クラウド事業者が Platform as a Service(PaaS) 形式で提供するデータベースやストレージと

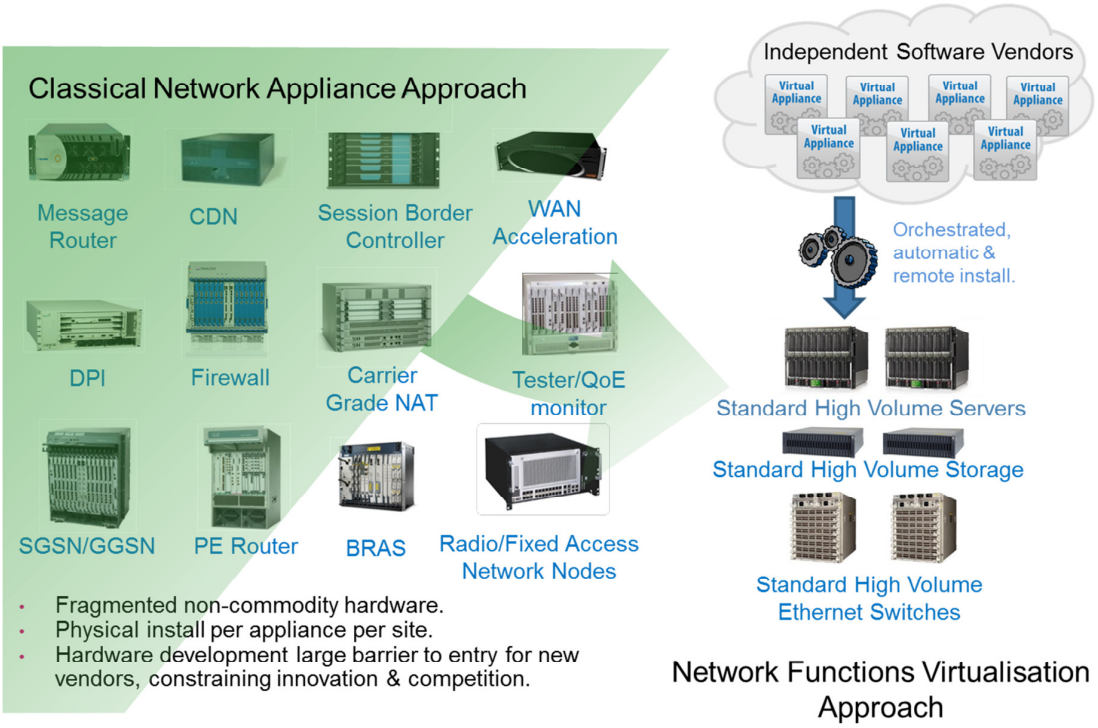


図 1.2: ETSI による NFV のコンセプトの概要 [6]

いった基礎的なインフラを組み合わせ，それらを操作するビジネスロジックを実装するマイクロサービスアーキテクチャが主流になってきている．また，これらのアプリケーションの構成はコードとして管理する Infrastructure as Code(IaC) によってプロビジョニングが自動化され，その表現も具体的なプロビジョニングの手続きをまとめた命令型の表現から，結果として実現される要件を記述する宣言型の表現に変化している．その結果，アプリケーションは，スケールアウトの自動化など，コードで宣言された内容によって動的に展開・運用されるようになった．このように，クラウドコンピューティング環境に最適化されたアプリケーションをクラウドネイティブと呼ぶ．

一方で，アプリケーションがネットワークに対して要求する通信品質は高度化しており，ネットワーク機能の配置を含むネットワークアーキテクチャの最適化が必要になっている．例えば，証券取引や自動運転といったアプリケーションは，遅延に対する要件を数十ミリ秒単位の要求から，1 ミリ秒以下の単位に変化させた．このような要求の変化は，パケット転送における優先度の最適化など，単純なネットワークの設定を変更するだけでは実現が困難であり，第 5 世代移動通信

システム (5G)[9, 10] で提案されているアプリケーションとネットワーク機能を物理的にエッジ側に配置する Mobile Edge Computing(MEC)[11, 12] のように，ネットワーク機能の配置を含むネットワークアーキテクチャの最適化が必要である．

アプリケーションのクラウドネイティブ化と，高度化する通信品質の要求を実現するためには，ネットワークの PaaS 化が必要である．アプリケーション全体から見れば，ネットワークはデータベースやストレージといった構成要素の一部である．したがって，アプリケーションの構成が宣言的なコードによって表現され展開・運用されるのと同様に，ネットワークも PaaS として取り扱うことによって，クラウドネイティブな展開と運用が可能になる．その結果，アプリケーションとネットワークの関係性は，ネットワークの構成にアプリケーションが最適化されてきたモデルから，アプリケーションの構成に対して最適化されたネットワークを動的に提供するモデルに変化する．すなわち，アプリケーションはクラウドネイティブなアーキテクチャを維持したまま，高度化する通信品質への要求を実現できる．

ネットワークの PaaS 化を実現するには，VNF の可搬性が新たな NFV の要件となる．PaaS 化されたネットワークでは，ネットワークの構成が宣言型の表現で定義されるため，VNF の最適な配置場所はアプリケーションが要求する通信品質によって異なるだけでなく，状況に応じて動的に変化する可能性がある．したがって，VNF はアプリケーションと同様にクラウドコンピューティングにおける高頻度かつ継続的な構成の変更と展開が必要になる．すなわち，VNF はアプリケーションと同様にハードウェアの構成に非依存で様々な環境で動作し，ソフトウェアによって記述されたアーキテクチャの構成に基づいて高速に展開され起動する可搬性が不可欠である．

1.3　本論文が解決する課題

従来の NFV は，仮想化基盤技術に仮想マシン (VM) の利用を前提としており，可搬性と性能の両立に課題がある．従来の NFV は，既存のネットワーク装置のハードウェアとソフトウェア全体を仮想化することに注力されており，VM を前提とした VNF の抽象化方法が採用されてきた．しかし，この手法において VM のパケット処理性能を向上させるには，NFVI(NFV Infrastructure) と呼ばれる VNF の動作環境に特化した VNF の最適化が不可欠である．その結果，市販されている VNF は，特定の NIC ハードウェアや，詳細な仮想 NIC の実装をシステムの動作条件として要求しており，VNF と NFVI の分離は困難であるのが現実である．すなわち，ハードウェアと NF の分離がもたらす NFV の価値を提供できておらず，

可搬性に課題があると言える．

これらの課題を解決するため，本研究ではコンテナ型 NFV アーキテクチャを提案する．提案するアーキテクチャは，VNF をオペレーティングシステムとは分離されたユーザスペースで動作するコンテナとして取り扱う．そのため，従来の VM を前提とした VNF と比較して，ハードウェアに非依存な VNF の可搬性を保証するとともに，軽量かつ高性能なパケット処理を実現する．また，既存のコンテナオーケストレーションシステムを利用した NFVI 全体における配置の最適化，スケールアウト，オートヒーリングなどが可能になり，ソフトウェアによる柔軟なネットワーク構成の変更を可能にする．その結果，5G の要件であるアプリケーションが要求する多様な通信品質を満たすネットワークアーキテクチャが実現可能になる．

1.4　本研究の貢献

コンテナ型 NFV アーキテクチャを実現するにあたって，本研究が貢献した具体的な成果は，VM を利用した NFV の評価と，プロセス型 VNF の提案である．

VM を利用した NFV の評価

VM を利用して VNF を実現するには，ネットワーク I/O を含む数多くの仮想化技術の組み合わせが必要である．本論文では，これらの組み合わせのパケット処理性能を評価し，市販の製品を利用した商用ネットワークの構築におけるベストプラクティスを導きだした．その結果を実証する場として，2015 年に開催された Interop Tokyo[2][13] の ShowNet[3][14] において virtual Customer Premises Equipment(vCPE)[15] サービスを設計・構築し，実運用に基づく評価をおこなった．また，VM を利用した NFV の評価から，VM を利用することの弊害を明らかにした．具体的には，VM によるセンシティブ命令の実行に伴うコンテキストスイッチ (VM_Exit) に対応するパケット処理性能の最適化と可搬性のトレードオフが課題であり，これらを解決する VNF のアーキテクチャが必要であることを示した．

[2]Interop Tokyo は毎年 6 月に開催されるネットワーク機器と技術の展示会である．

[3]ShowNet は，Interop に出展している企業が販売促進のためにネットワーク機器を提供し，それらを組み合わせて構築されるデモンストレーションネットワークである．ShowNet の役割は，最新のネットワーク装置に実装されている新しいネットワーク技術の実現性を示すと共に，実際に出展者と来場者に対してインターネットの接続性を提供し，それらが実際の商用ネットワークで運用可能であることを示すことである．

プロセス型 VNF の提案

VM を利用した NFV の評価から明らかになった課題を解決するため，プロセス型 VNF を設計・実装し，その優位性を評価した．プロセス型 VNF は，カーネルが提供する高速なネットワーク I/O の API を利用した VNF の実装方法であり，一般的な POSIX API を利用したユーザスペースのアプリケーションとして抽象化された VNF である．具体的には，Netmap API を利用したルータ，NAT，Firewall を実装し，従来の VM を利用した方式と比較した場合，単体の VNF では約 2 から 5.5 倍のパケット処理性能の向上を達成し，VNF の連結では最大で約 9 倍のパケット処理性能の向上を達成した．また，提案手法がコンテナおよびオーケストレーションシステムとの親和性が高いことを示した．

1.5　本論文の構成

2 章では，本論文が目指すネットワークの PaaS 化によって実現される次世代のネットワークサービスプラットフォームについて述べ，NFV の要件について議論する．3 章では本研究に関連する技術および研究について述べる．4 章では VM を利用した NFV を評価し，商用ネットワークにおけるベストプラクティスを述べ，VM を利用した NFV の課題をまとめる．5 章ではコンテナ型 NFV アーキテクチャを提案し，コンテナ型 NFV アーキテクチャの鍵となるプロセス型 NFV の設計・実装・評価について述べる．最後に 6 章で結論と今後の展望を述べる．

第2章　次世代のネットワークサービスプラットフォーム

本章では，アプリケーションが要求する通信品質の多様化に対して，最適化されたネットワークを提供する概念であるネットワークスライシングについて述べる．次に，ネットワークスライシングの実現に向けた課題を説明し，その解決方法としてネットワークサービスの Platform as a Service(PaaS) 化を提案する．

2.1　ネットワークスライシング

アプリケーションの多様化に伴い，ネットワークにおける通信品質への要求は高度化している．図 2.1 に，ITU-T の IMT-2020(International Mobile Telecommunication system for Year 2020) が定めた第 5 世代の移動体通信システム (5G) における通信の要件とアプリケーションの関係を示す．IMT-2020 が定めた主な通信の要件は，「超高速 (eMBB)」，「多数同時接続 (mMTC)」，「低遅延・高信頼 (URLLC)」の 3 点である．それぞれの要件に対する代表的なアプリケーションは，eMBB に対しては大容量のデータ通信を必要とする 4K/8K などの高品質な映像伝送，mMTC に対しては多数のセンサーの接続を必要とする IoT や M2M，URLLC に対しては高信頼かつ低遅延の通信を必要とする自動運転などが挙げられる．一般的にこれらの要求事項はトレードオフであり，個々のアプリケーションが要求する通信品質に最適化されたネットワークは異なる．そのため，共通のインフラストラクチャ上で様々なアプリケーションの要求に対応可能な超柔軟性 [16] が要求されている．

ネットワークスライシング [18, 19] は，ネットワークの超柔軟性を実現するにあたって重要な概念である．図 2.2 にネットワークスライシングの概要を示す．ネットワークスライシングでは，アプリケーションの要求ごとに独立したインフラストラクチャ (コンピューティング・ストレージ・ネットワーク機能) の集合体をネットワークスライスと呼ぶ．すなわち，ネットワークスライスとは，異なる要求を持つアプリケーションごとに最適化された仮想的なネットワークである．ネットワークスライシングは，アプリケーションの要求する通信品質に応じて，ネット

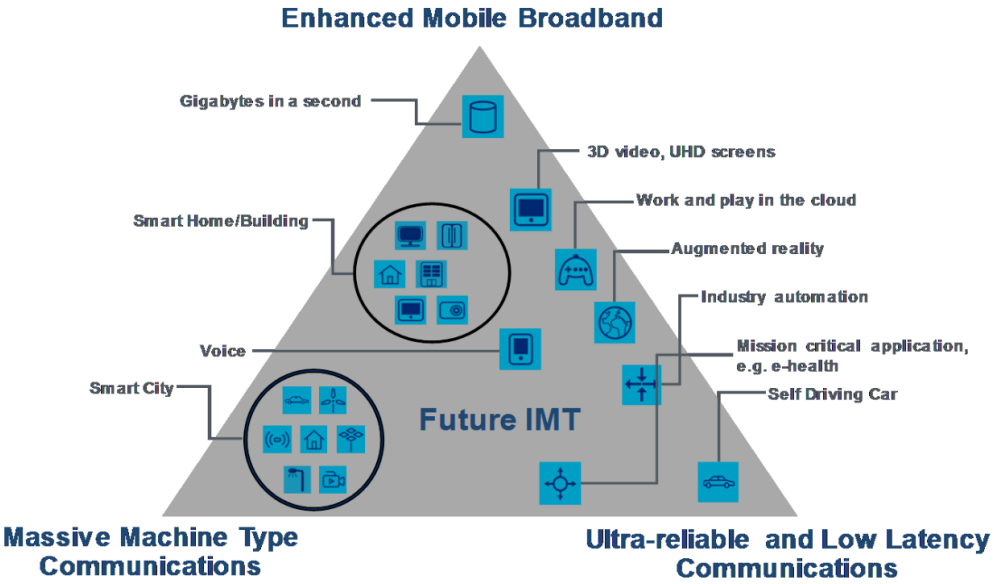


図 2.1: 2020 年のネットワークへの要求とユースケースの関係 [17]

ワークスライスの構成を動的に変更する機能を提供する機能を併せ持つ．つまり，ネットワークスライシングとはそれらを動的に割り当て可能なオンデマンドサービスと言える．その結果，アプリケーションとネットワークの関係性は，アプリケーションが既設のネットワークが持つ帯域幅・遅延・信頼性といった制約に最適化していくモデルから，アプリケーションの要求に対して最適なネットワークを動的に構築していくモデルに変化する．

ネットワークスライシングは，アプリケーションごとに異なるネットワークアーキテクチャを実現可能にし，ネットワーク機能の最適な配置を可能にする．図 2.3 に，5G のモバイルネットワークで定義されている 3 つの通信品質ごとに最適化されたネットワーク機能の配置例を示す．モバイルネットワークで代表的なネットワーク機能は，モバイル通信の制御機能 (Signaling)，端末の認証機能 (AAA)，データ通信の中継機能 (Gateway) である．モバイルネットワークでは，これらのネットワーク機能に加えて，DPI や WAF といった，特定のアプリケーションに特化したネットワーク機能を組み合わせて構成される．これらのネットワーク機能は，4G では Evolved Packet Core(EPC) と呼ばれ，一箇所に集中して配置されていた．5G ではこれらのネットワーク機能は分散可能になり，アプリケーションが要求する通信品質に最適化された配置が可能になった．例えば，URLLC のような高信頼かつ低遅延な通信を必要とするアプリケーションでは，アプリケーション自体を

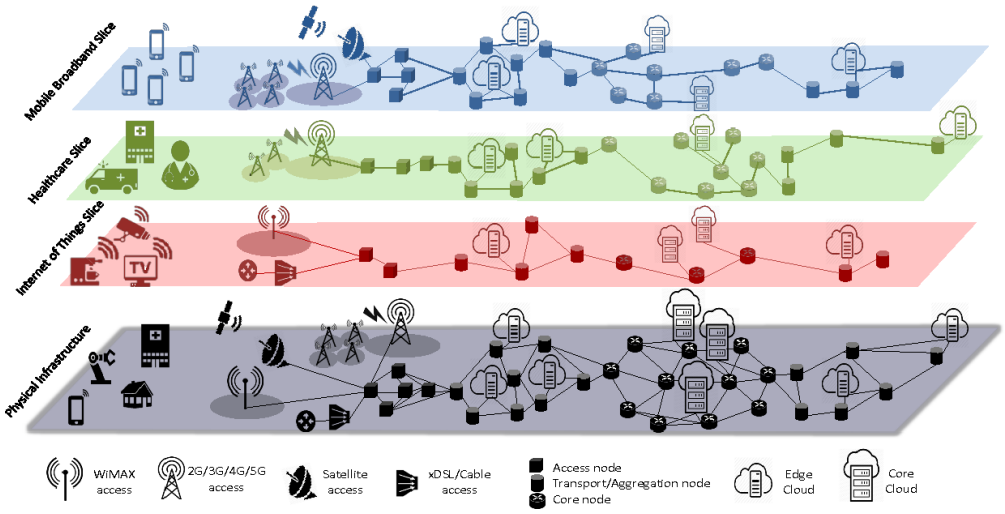


図 2.2: ネットワークスライシングの概要 [18]

より端末の近くに配置する必要がある．その結果，Gateway とアプリケーションそのものをネットワークのエッジ側に配置する Mobile Edge Computing(MEC) が提唱されている．また，mMTC では IoT のセンサーデバイスなど，端末数の急激な変化に対応する必要があり，AAA や Signaling といった機能は，需要の変化への弾力性があるパブリッククラウドで動作させる必要がある．このように，アプリケーションの要求する通信品質を実現するためには，ネットワーク機能の配置を最適化することが重要である．

しかしながら，ネットワークスライシングの具体的な実現方法は，その概念から乖離した部分があり，アプリケーションの要求に対して最適なネットワークを動的に構成する機能の実現に課題がある．例えば，3GPP におけるネットワークスライシングの標準化 [20] におけるネットワークスライスの構築とは，通信事業者の視点におけるエンドツーエンドのネットワークを構築することを主眼としており，Radio Area Network(RAN)，モバイルコア，両者を接続するトランスポートネットワークを連結した仮想ネットワークをネットワークスライスとして定義しており，これらを結びつけることによって提供されるネットワークをカタログ化し，ネットワークの利用者に提供する方式が検討されている．例えば，3GPP では mMTC におけるモバイル IoT のシナリオを定義し，それぞれのシナリオには回線速度や遅延時間の目安は定義されている [21]．これらのシナリオは同一のインフラストラクチャ上に異なる要件を満たす複数のネットワークを構築を可能にする．しかし，アプリケーションの要求，すなわちアプリケーション開発者のネット

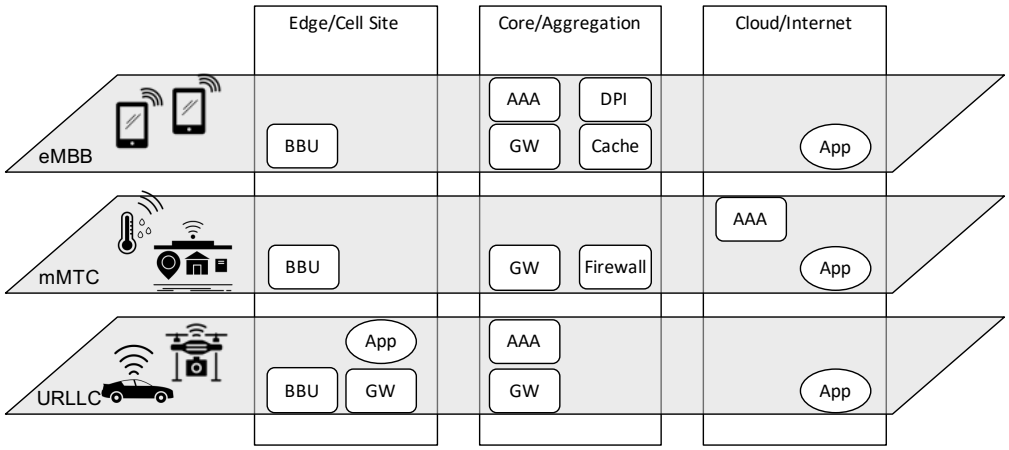


図 2.3: 5G の通信要件ごとに最適化されたネットワークアーキテクチャ

ワークに対して通信品質要求に対して最適化しているわけではなく，むしろアプリケーション開発者はシナリオとして与えられたネットワークの通信品質に最適化したアプリケーションの開発を余儀なくされる．そのため，既存のネットワークスライシングの実現方法は，ネットワークスライシングの概念である「アプリケーションの要求に対して最適なネットワークを動的に構築していくモデル」とは乖離があり，進化の速度が早いアプリケーションの要求に対して最適化が困難である．

2.2　ネットワークサービスの PaaS 化

本研究では，ネットワークスライシングの概念において重要な要素である，アプリケーションが要求する通信品質に対して最適なネットワークを動的に構成する機能の実現に向けて，ネットワークサービスの Platform as a Service(PaaS) 化を提案する．PaaS は，本来アプリケーションを実行するプラットフォームを提供するクラウドコンピューティングにおけるサービスの形態である．このプラットフォームは，データベースやストレージといったアプリケーションを構成する機能を，アプリケーションの要求に応じて動的に供給する．また，これらの構成要素は，その供給量や配置を宣言的に記述したコードによって管理・運用される．したがって，アプリケーションの利用者数や，コンピューティング資源の利用率などに基づいて，動的に供給量や配置が変化する．このような PaaS の概念を，ネットワークサービスに適用することによって，通信事業者が提案するネットワーク

スライシングの実装方法では困難なアプリケーションの要求に対して最適なネットワークを動的に構築していくモデルを実現する．

図 2.4 に，PaaS 化されたネットワークサービスの概要を示す．PaaS 化されたネットワークサービスでは，アプリケーションが自身を構成する要素の一部として，利用するネットワーク機能を組み合わせたネットワークの構成と通信品質の要件を宣言的に記述する．この記述は，アプリケーションを構成するデータベースやストレージといった計算資源と同列に取り扱われ，単一のオーケストレーションシステムによって具現化される．オーケストレーションシステムは，アプリケーションを利用する端末数の増減や輻輳といったアプリケーションから見たネットワークの状況を監視し，宣言的に定義されたネットワークの通信品質要件に基づき，最適化されたネットワーク機能の配置や展開する数量を変化させる．具現化されたアプリケーションの構成要素群は，サービスチェイニングによって任意の順序で連結され，アプリケーションに特化したトラフィックの制御がおこなわれる．

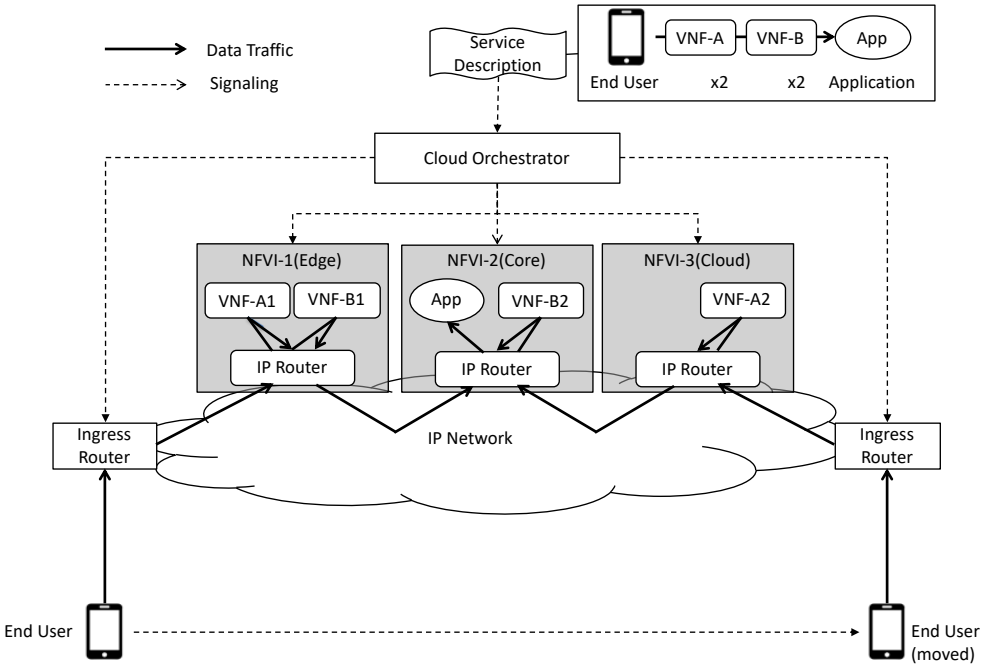


図 2.4: PaaS 化されたネットワークサービスの概要

ネットワークの PaaS 化を実現するには，クラウドコンピューティング，SDN，NFV の技術領域における新たな要件と，それらが協調するオーケストレーションシステムが必要になる．図 2.5 に各技術領域の関係性と，新たに必要になる要件を示す．まず，クラウドコンピューティングの領域において，アプリケーションの

要求する通信品質に最適化されたネットワーク機能の配置を実現するには，ネットワーク機能を含むアプリケーションの構成要素を，通信事業者とクラウド事業者の両者に配置可能なハイブリッドクラウド[1]を導入する必要がある．また，SDN の領域において，具現化されたアプリケーションの構成要素群を接続するサービスチェイニングは，異なる管理・運用主体のネットワークドメインにおいても制御可能にするため，SRv6 など IP を利用した経路制御が必要である．そして，NFV の領域において，ネットワーク機能が配置される場所，すなわち VNF の動作環境である NFVI の構成に非依存で動作する可搬性と，専用のハードウェアで動作するネットワーク機能と同等のパケット処理性能の両立が必要になる．これらクラウドコンピューティング，SDN，NFV の領域における要件を満足させることにより，それらが協調するオーケストレーションシステムが実現可能になる．

図 2.5: ネットワークサービスの PaaS 化に向けた技術領域の関係性

[1]ハイブリッドクラウドは，オンプレミスのプライベートクラウドと，パブリッククラウドの間で，アプリケーションを移動可能にする概念である．この概念は，アプリケーションの需要に応じて，最適な動作環境を選択可能にする．例えば，ハイブリッドクラウドでは，開発段階やリリース直後はオンプレミスでアプリケーションを動作させ，オンプレミス環境では処理しきれない突発的な利用者の増加に対してパブリッククラウドを利用することができる [22, 23].

2.3　本章のまとめ

5G のネットワークは，アプリケーションが要求する多様な通信の品質を，単一のインフラストラクチャで実現するネットワークスライシングを提供する．ネットワークスライシングでは，アプリケーションの要求に応じて，ネットワーク機能の配置を含めたネットワークアーキテクチャの最適化が必要である．ネットワーク機能の配置を最適化するために NFV が新たに果たす役割として，VNF の可搬性が重要である．

第3章　関連研究

本章では，はじめにETSIが標準化しているNFVのアーキテクチャについて述べる．次に汎用的なハードウェアとオペレーティングシステムにおける高速なネットワークI/O技術について説明し，それらをNFVの観点からVMに適用した先行研究について述べる．その上で，VMの利用を前提とした従来のNFVが潜在的に抱えている課題について述べる．

3.1　NFVのアーキテクチャ

ETSI NFV ISG(Industory Specification Group)では，NFVの概念的な議論に加えて，アーキテクチャの標準仕様について議論がされている．図3.1にETSI NFV ISGで標準化されているNFVのアーキテクチャを示す．本研究が主に着目するNFVの構成要素は，Management and Orchestration(MANO)，NFV Infrastructure(NFVI)，Virtual Network Function(VNF)の3点である．MANOは，NFV Orchestrator(NFVO)において，Operation Support System(OSS)/Buisiness Support System(BSS)に登録されている顧客に対応したVNFの組み合わせが記述されたサービスカタログを保持している．サービスカタログは，NFVIで利用されている技術や構成に非依存な抽象度の高いサービスの定義である．NFVOはこれらのサービスカタログを元に，VNFを構成するVMの生成とVNFの設定を，Virtualized Infrastructure Manager(VIM)とVNF Manager(VNFM)に依頼する．したがって，この依頼の抽象度の高さは，NFVのシステム全体の抽象度の高さに密接な関わりがあり，NFVの可搬性を左右する要素だと言える．

このアーキテクチャにおけるVNFの管理では，VNFの動作環境としてVMの利用を前提としている．VIMおよびVNFMが，NFVOに対して提供するインターフェースには，NFVの利用に特化した変更を加えたOASIS TOSCA[25, 26]が利用される．OASIS TOSCAは，クラウドコンピューティングにおいて，システム構成情報を定義するための標準仕様であり，様々なクラウドコンピューティング環境で動作可能な抽象度の高い記述が可能な言語である．NFVに特化したOASIS TOSCAの仕様は，VMの利用を前提としている．例えば，Virtual Deployment

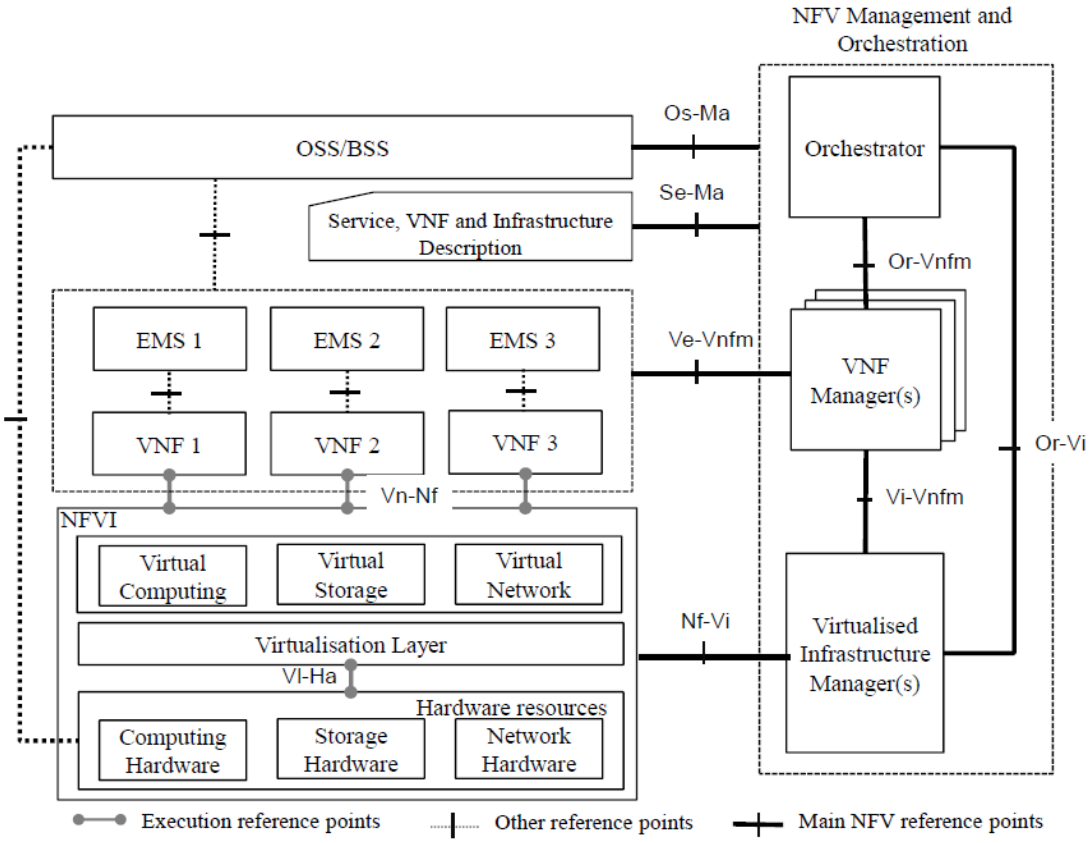


図 3.1: ETSI NFV ISG で標準化されている NFV のアーキテクチャ [24]

Units(VDU) と呼ばれる VNF の最小構成単位には，Connection Point Description，Virtual Computing，Virtual Storage といった VM の構成要素に対応する記述が必要になる．したがって，NFVI は KVM などのハイパーバイザに相当し，VIM はその管理をおこなう OpenStack[27] などの Infrastructure as a Service(IaaS) における VM の管理ソフトウェアに相当する．

3.2　高速ネットワーク I/O 技術

PC サーバ向けの 10 ギガビット Ethernet カードの普及と，マルチプロセッサ化およびマルチコア化に伴い，オペレーティングシステムにおける高速なネットワーク I/O の研究が重点的に実施されてきた．Linux カーネルでは，大量のパケット処理における割り込みに起因するコンテキストスイッチの負荷を低減するため，割

込みとポーリングをハイブリッドに運用する NAPI[28]，マルチキューを利用して割り込みを複数の CPU コアに分散する RSS(Receive Side Scaling)[29]，割込みとパケット・プロセスを同じ CPU で動作させる RFS(Receive Flow Steering)[30] を採用している．このような基礎技術を応用したパケット転送処理の並列化，バッチ処理，パイプライン処理を実現した RouteBricks[31, 32] や，CPU よりも高密度なマルチコア化を実現している GPU を利用した PacketShader[33] は，ベアメタルサーバを利用した高速なソフトウェアルータの代表例である．

次に，汎用的なハードウェアでの高度な NF の実現に向けて，ユーザスペースにおけるパケット処理の高速化が注目されるようになった．ユーザスペースにおいてパケット処理をおこなう場合には，カーネルにおけるネットワークスタックや，カーネルとユーザスペース間のデータコピーがオーバーヘッドとなっている．この課題を解決するため，ユーザスペースでデバイスを操作するためのフレームワークを利用してユーザ空間に NIC のデバイスドライバを実装する Intel Data Plane Development Kit(DPDK)[34] や，カーネル内のデバイスドライバが書き込むパケットバッファーをユーザスペースから参照可能にする API を提供する Netmap[35, 36] が提案された．これらの技術は，ユーザスペースで動作する多様なライブラリの利活用と，高速なネットワーク I/O を両立し，より複雑なパケット処理を伴う NF を実現する．例えば，SSL の暗号化・復号化や，ビデオのトランスコードなど，カーネルで実現するには多大な開発コストを伴う NF を，既存のユーザスペースのライブラリを活用して容易に実装できるようになった．また，バイパスしたネットワークスタックをユーザスペースで実装することによって，カーネルで実装するよりも高速な通信が可能な事例 [37] も示されている．

3.3　NFV への適用

これらの高速なネットワーク I/O 技術の適用領域は，NFV が提唱されたことによって仮想マシンにも拡大された．仮想マシンは，ユーザスペースで動作するプロセスであり，仮想 NIC へのレジスタアクセスに基づくコンテキストスイッチや，カーネルとユーザスペース間のデータコピーがネットワーク I/O のオーバーヘッドになる．これらのオーバーヘッドを回避する方法の一つは，サーバのリソースの物理的なパーティショニングである．物理 NIC をキュー単位で仮想化する SR-IOV[38] と，PCI デバイスを仮想マシンに直接アタッチする PCI Pass Through の組み合わせによって，仮想マシンが NIC のキューを占有する．その結果，仮想マシンはハイパーバイザを介さずに NIC のキューにアクセス可能になり，DPDK を利用したユーザスペースで高速なパケット処理を実施する VNF 製品は広く普及し

ている．

しかしながら，このような実装は，サービスチェイニングと，ハードウェアとソフトウェアの分離の観点で，NFV が目的とする柔軟性を満たしているとは言えない．サービスチェイニングにおいては，SR-IOV が PCI 内部で保持しているパケットスイッチング機能は，ハードウェアに依存しており，ソフトウェアによる外部からの操作が困難である．ハードウェアとソフトウェアの分離においては，ゲスト OS および VNF におけるパケット処理性能の最適化に関しては，ハードウェアへの依存性が高くならざるを得なくなっている．例えば，ゲスト OS は，仮想マシンに直接アタッチされた NIC のハードウェアに合致したデバイスドライバを保持しなければならない．また，VM 上で発生する I/O に由来する割り込みは，プロセッサによる VM への割り込みマッピングにおいても，ハイパーバイザへのコンテキストスイッチを必要とするため，CPU コアと VM の紐付けに基づく複雑な割り込みの制御が必要である [39]．DPDK においては，ユーザ空間に実装されるデバイスドライバに専用の CPU コアを割り当てたビジーウェイトなポーリングによる実装を採用しており，CPU コアと PCI バスを同じプロセッサに割り当てるなど，複雑な最適化が必要である．

これらの課題を解決するため，同一のハイパーバイザ内における VNF 間を接続する高速な仮想スイッチおよび仮想マシンとの接続方法が提案されている．様々なヘッダ情報に基づく柔軟なパケットスイッチングと，高速なパケット処理を両立するため，データプレーンの実装に DPDK を活用した Open vSwitch[40, 41] が提案されている．しかしながら，ソフトウェアスイッチと仮想マシンとの接続においては，依然としてデータコピーがオーバーヘッドになるため，DPDK を利用した共有メモリ型のパケットバッファを提供する NetVM[42] や，Netmap を利用した仮想 NIC のバックエンドを提供する ClickOS[43] が提案されている．

3.4　関連研究の課題

これらの VM を利用した NFV は，ネットワーク I/O 性能の最適化に伴ってハードウェアとソフトウェアの依存関係が密結合になる傾向にある．NetVM で利用されている DPDK は，既存のオペレーティングシステムと VM が抱えるコンテキストスイッチングの課題を，NIC のポートごとに CPU コアを占有したビジーウェイトなポーリングによって解消する．しかし，この手法はハードウェアを物理的にパーティショニングしているに過ぎず，仮想スイッチと組み合わせた運用では，本来 VNF が実施するべきアプリケーションに特化したパケット処理に利用できる CPU コア数を著しく制限する．また，ClickOS は同様の課題をハイパーバイザ

コールを直接呼び出し可能な準仮想化を活用して回避している．しかし，この手法は VNF のオペレーティングステム側に特別なカーネルを利用する必要がある．したがって，ETSI が当初の NFV の目的としていたハードウェアとソフトウェアの分離を実現しているとは言えない．

また，これらのハードウェアとソフトウェアの密結合は VNF の抽象度を下げ，MANO を実装する側に複雑な設計を要求する．例えば，VM が SR-IOV を活用する場合，ハイパーバイザが SR-IOV に対応した NIC を保持し，さらにデバイスドライバの設定で VF を有効にしていることを確認した上で，他の VM に未割り当てな VF を予約するプロセスが必要である．また，DPDK では NIC が接続されている PCI-Express と CPU の NUMA ノードの整合性を確認した Huge メモリの割り当てが必要である．確かに，OpenStack はこのような機能を有しているが，多様なハードウェアを前提とした環境で動作させるのは難しい．したがって，NFV の構成要素である MANO，NFVI，VNF を異なるベンダから調達してインテグレーションするのは困難である．

このような VNF の抽象度の低さと，MANO に対する設計・実装の負担は，結果として VNF の細分化と可搬性を低下させる．例えば，VNF には VM のオペレーティングシステムを含むため，VNF を起動するにはオペレーティングシステムを起動する時間が含まれるだけでなく，VNF イメージのファイルサイズも大きくなり，VNF の細分化が困難である．また，VNF の定義には VM の構成も含まれているため，仮想 NIC などを含む仮想デバイスとデバイスドライバが不可欠であり，VNF の可搬性を低下させる．すなわち，VM を利用した NFV は，次世代のネットワークサービスプラットフォームにおいて，NFV が期待されている役割を果たすことの障害になる．

3.5　まとめ

ETSI が標準化している NFV の仕様では，ハードウェアとソフトウェアの依存関係を疎結合にし，抽象化された VNF を MANO によってソフトウェアで制御するモデルが提案されている．しかしながら，汎用的なハードウェアとオペレーティングシステムを利用したネットワーク I/O の高速化は，VM の利用を前提とした場合に，ハードウェアとソフトウェアの依存関係が密結合になる傾向にある．その結果，MANO による VNF の管理において，複雑なシステムの設計を余儀なくされる．したがって，ネットワークサービスプラットフォームにおいて，NFV が期待されている役割を果たすのは困難である．

第4章　VMを利用したNFVの評価と実践

本研究では，市販されている VNF と NFVI の製品を利用した商用ネットワークを構築するため，VM を利用した NFV の評価を実施した．本章ではその評価結果および導き出されたベストプラクティスについて述べる．そして，ベストプラクティスに基づいて設計した vCPE サービスである FlowFall について説明し，FlowFall のアーキテクチャに基づいて構築・運用した商用ネットワークにおける vCPE サービスの実証実験について述べる．そして，実証実験を通して得られた知見に基づく VM を利用した NFV の課題を述べる．

4.1　背景

本研究では，市販されている NFV の関連製品を組み合わせて商用ネットワークが構築･運用できることを実証するため，2013 年から Interop Tokyo[13] の ShowNet[14] において，NFV を活用した出展者の収容ネットワークの仮想化に取り組んできた [44].

ShowNet の重要な役割の一つは，複数の出展者から ShowNet に提供される様々な製品を組み合わせてネットワークを構築し，最新の製品の相互接続性を実証することである．すなわち，NFV における ShowNet の役割は，様々なネットワークベンダが提供する NFVI と VNF を組み合わせてネットワークが構築・運用可能なことを実証し，実際の商用ネットワークでもハードウェアとソフトウェアを分離可能なことを示すことである．したがって，出展者に対して十分なネットワークの性能を提供するだけでなく，高い柔軟性を基盤とした相互接続性が必要になる．

しかしながら，NFV にサーバの仮想化技術で利用されてきた技術を適用した場合，パケット処理性能において課題がある．実際問題として，Interop Tokyo 2014 の ShowNet に提供された VNF ベンダにおいて，共通してサポートする NFVI 環境を利用した SFC の構成では，VNF の連結数が増えるにしたがって，個々の VNF におけるパケット処理性能が低下することを確認した．その対策として，VNF を

構成する VM に CPU コアやメモリなどの資源を追加して性能の向上を試みたが，パケット転送性能の十分な向上が得られなかった [45].

そこで，本研究では産学連携コンソーシアムにおける市販製品を評価するとともに，実際の商用ネットワークの構築・運用の実証実験をおこなった．本研究では，NFV を前提としたネットワークのアーキテクチャを検討するため，次世代 Network Service Platform(NSP) コンソーシアムを発足した．次世代 NSP コンソーシアムは，慶應大学が主催し，通信事業者，測定器ベンダ，NFV に関係する製品を販売するネットワークベンダ，合計 33 社を取りまとめた産学連携のコンソーシアムである．次世代 NSP コンソーシアムでは，NFV に関係する技術の検証方針を策定し，参加者合意のもとで実際に市販されている NFVI と VNF の組み合わせたパケット処理性能や，それらの管理手法の柔軟性を評価した．この評価に基づいて，VNF を連結してネットワークサービスを構成する SFC として FlowFall を設計し，Interop Tokyo 2015 の ShowNet では実際に市販製品を用いて出展者に提供するネットワークを構築し，20 の出展者に対して 3 日間の商用インターネット接続性を提供した．

4.2　vCPE サービス

ShowNet における出展者の収容ネットワークの仮想化において，実際に提供したサービスは NFV と SFC を組み合わせた virtual Customer Premises Equipment(vCPE) サービス [15] である．vCPE サービスは，従来の CPE で動作していた様々なネットワーク機能を，通信事業者の VNF として動作させ，それらを SFC によって連結することによって各利用者が要求するネットワークを実現する技術である．図 4.1 に vCPE サービスのネットワーク構成を示す．vCPE サービスでは，VNF の動作する汎用ハードウェアが，利用者のアクセス回線を集約する場所に設置される．通信事業者は，利用者に対して Web ポータルなどの連結する VNF を制御可能なインターフェースを提供する．その結果，利用者は自身の運用するネットワークのポリシーに応じて，動的に VNF の組み合わせと順序を変更できるようになる．また，利用者は，VNF の運用を通信事業者へ委託可能になる．例えば，Firewall や IDS など，頻繁にシグネチャーのアップデートが必要なセキュリティ製品の運用を，通信事業者でまとめて運用することが可能になる．

vCPE サービスの実現には，一般的な NFV の要求事項 [46] であるセキュリティ，サービス保証，耐障害性，既存ネットワークとの共存，電力効率などを考慮する必要はあるが，ShowNet の構築・運用においては，先に述べた高い柔軟性を基盤とした相互接続性と，出展者に対して十分なネットワークの性能を提供するためのスケールアウトが要件となる．以下に本論文における相互接続性とスケールア

ウトの定義を示す．

相互接続性

ネットワーク機器のコントロールプレーンとデータプレーンに標準化された技術を利用し，様々なベンダのネットワーク機器を組み合わせてネットワークを設計できること．NFV の利点は，様々な VNF を組み合わせたネットワークの動的な構成変更であり，それを制限する特定ネットワークベンダに限定された技術や標準化が進行中の技術の利用は避ける必要がある．

スケールアウト

VNF のパケット転送性能が，VNF に割り当てる資源の追加によって段階的に増強可能なこと．vCPE サービスでは，VNF は集約されたアクセス回線分のパケット処理性能を実現する必要があり，従来のハードウェアで実装されたネットワーク装置と同等の性能が要求される．しかし，単一の VM で要求されるパケット処理性能を実現するのは困難であり，複数の VM を用いてトラフィックを分散処理する必要がある．

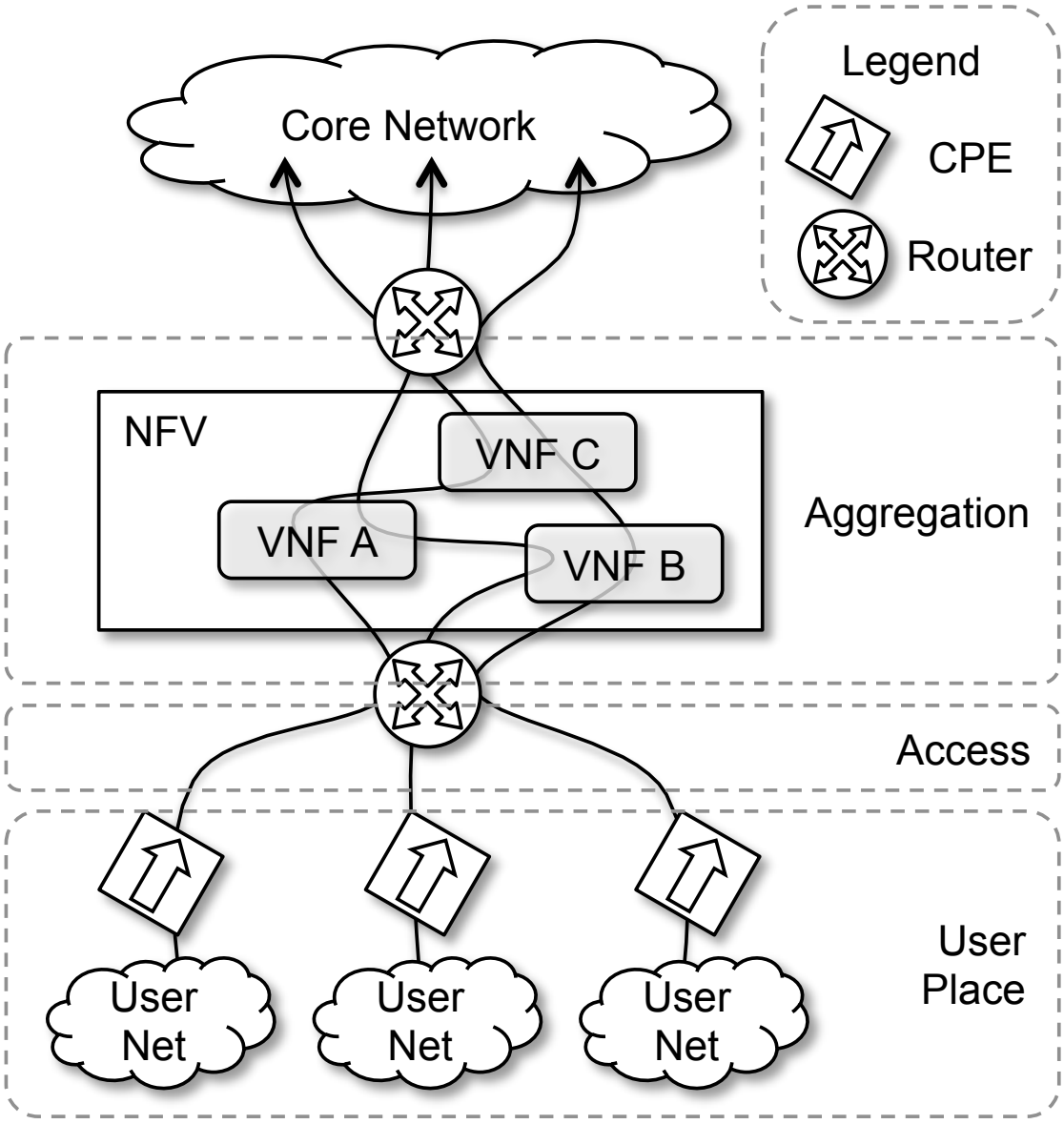


図 4.1: vCPE サービスのネットワーク構成

4.3　ShowNet における vCPE の位置づけ

ShowNet の出展者収容ネットワークの要求と構成は，vCPE サービスに求められるものと類似しており，NFV および SFC の実践的な概念実証に適している．ShowNet におけるネットワークの利用者とは出展者であり，出展者が ShowNet に対して要求するサービスとは，出展者のデモに応じたトラフィック処理方法である．例えば，セキュリティ製品のデモをおこなう出展者は，外部からの攻撃トラフィックを自社の出展者収容ネットワークに流入させるため，ShowNet に対して Firewall によるトラフィックフィルタの解除を要求する．こうした要求に耐えうる柔軟なネットワークの構成変更を実現するために，ShowNet は vCPE サービスと同様に利用者ごとに適用する VNF の組み合わせを動的に変更できる必要がある．すなわち，ShowNet の出展者収容ネットワークにおける要件は，本質的には通信事業者における vCPE サービスの要件と同じである．

また，ShowNet の出展者収容ネットワークの構成は，出展者ごとのアクセス回線をバックボーンの手前で集約して集中管理する．ShowNet は出展者収容ネットワークに対して 1Gbps のアクセス回線と，プライベートまたはグローバル IP アドレスのスタブネットワークを提供する．ShowNet 2015 における 138 の出展者収容ネットワークのうち，最終的には 20 の出展者収容ネットワークを vCPE サービスで収容することとなった．そのため集約したアクセス回線分のパケット転送性能 (20Gbps) を実現する必要があり，従来のハードウェアで実装されたネットワーク装置と同等の性能が要求される．すなわち，ShowNet の出展者収容ネットワークの構成は，通信事業者が vCPE サービスを実装する際のネットワーク構成と類似している．したがって，ShowNet の出展者収容ネットワークの構成で vCPE サービスを実現できれば，通信事業者における実際の商用サービスも実現できると考えられる．

そこで，Interop Tokyo 2015 の ShowNet における vCPE サービスの設計に先立って，全ての VNF ベンダが共通してサポートする NFVI 環境で可能な限り性能を出すために，様々な VM の構成技術の組み合わせを比較し，性能と柔軟性の観点から評価し，市販の VNF が共通して対応している汎用的な NFVI 環境のベストプラクティスを導き出す．

4.3.1　検討内容と評価環境

そこで本研究では，相互接続性を維持したままスケールアウトが可能な VNF 構成を検討するため，多くの VNF 製品が対応している VM のネットワーク I/O 技術

と，VM の連結方式および VM の資源割り当て方式の組み合わせを評価し，vCPE の構成要素間による資源競合が発生しにくい方式を検討した．

VM の連結方式

VM の連結方式では，表 4.1 に示す VM のネットワーク I/O 技術を比較し，資源競合によりパケット転送性能が低下しない方式を検討する．図 4.2 に，表 4.1 に示した VNF 構成におけるトラフィックの通過パスを示す．利用した VM のネットワーク I/O 技術は，ShowNet 2014 で利用したソフトウェアで用いた仮想スイッチである OpenVswitch(OVS)[40] と，準仮想 NIC(Virtio)[47] を組み合わせた構成に加えて，PCI カードのハードウェア補助による資源競合の回避が期待できる SR-IOV を検討対象とした．SR-IOV は，VF(Virtual Function) と呼ばれる PCI カード内の仮想 NIC を持ち，VM は VF を PCI PassThrough によって直接利用する．VM の連結方式には，ShowNet 2014 で採用したハイパーバイザ内で動作する仮想スイッチ (OVS)，ハイパーバイザ外部のハードウェアスイッチ，SR-IOV をサポートする PCI カード内のスイッチを利用した VM の連結を対象にした．

VM の資源割り当て方式

VM の資源割り当て方式では，表 4.1 に示した VM のネットワーク I/O 技術と VM の資源割り当て方式を組み合わせた VNF 構成を比較し，資源の追加に対してパケット転送性能の向上率が高い方法を検討する．VM の資源割り当て方式には，ShowNet 2014 で採用した単一の VM に資源を集約して VNF を構成する方法 (以下，資源集約と記載する) と，複数の VM に資源を分散して VNF を構成する方法 (以下，資源分散と記載する) を対象とした．

表 4.2 に検討に使用した評価環境を示す．評価対象の VNF は，VM の OS 上で動作するカーネルに付属するパケット転送機能と iptables を利用して構成した．パケット転送性能の評価は，負荷試験機器として Netmap[35] に付属するトラフィック生成アプリケーションを利用して，フレームサイズ 64 バイトで 10 Gigabit Ethernet の理論値でトラフィックを送信し，通過できたパケット数を 10 秒間計測した．

表 4.1: 評価した VNF 構成で利用した技術

VNF 構成	VM のネットワーク I/O 技術
(1) Virtual-Switch_OVS	OVS[40] + Virtio-Net[47]
(2) Exter-Switch_OVS	OVS[40] + Virtio-Net[47]
(3) PCI-Builtin-Switch_SR-IOV	SR-IOV[38]
(4) PCI-External-Switch_SR-IOV	SR-IOV[38]

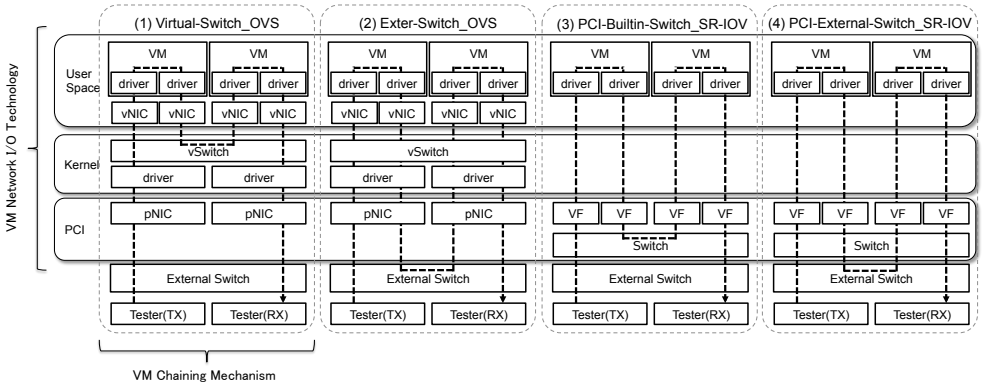


図 4.2: VNF 構成ごとのトラフィックの通過パス

表 4.2: 評価環境

プロセッサ	Intel(R) Xeon(R) E5-2650 2.00GHz
CPU コア数	8 コア
メモリ	メモリ 64G バイト
ホスト OS	Linux kernel 4.4.0-62
ハイパーバイザ	QEMU KVM 2.5.0
VNF OS	Linux kernel 4.4.0-62
トラフィック生成	Netmap
計測時間	10 秒

4.3.2　評価結果

VM の連結方式

図 4.3 に同一ハイパーバイザにおける VM の連結数と性能低下率の関係を示す．評価対象は，表 4.1 に記載した VM のネットワーク I/O 技術と VM の連結方式の組み合わせた VNF 構成である．縦軸は VM が 1 個の場合のパケット転送性能の低下率を示し，横軸は VM の連結数を示す．

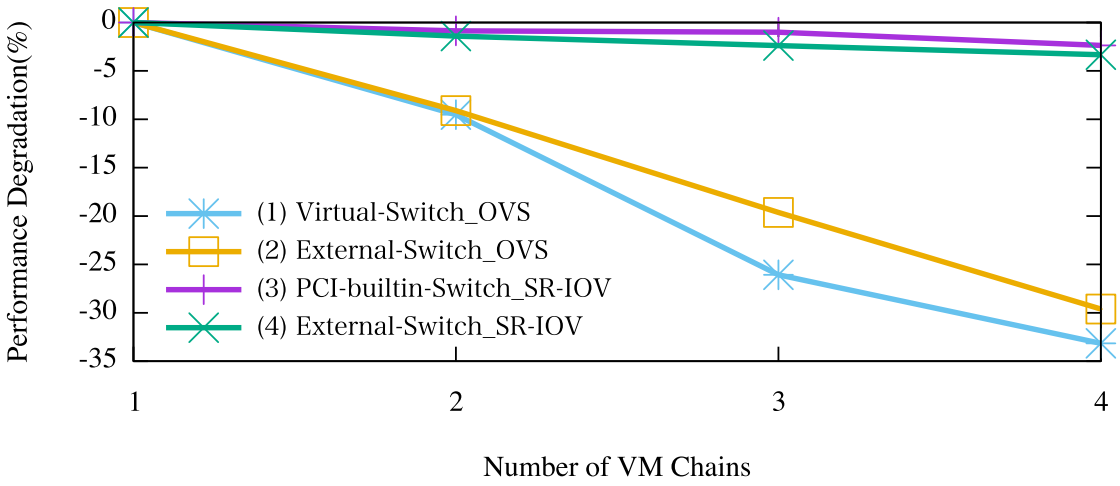


図 4.3: VM 連結方式の評価

仮想スイッチと仮想 NIC を利用した場合は，ハイパーバイザ内部の仮想スイッチ，ハイパーバイザ外部のスイッチのどちらを利用した場合 (図中 (1) および (2)) も，連結する VM 数が増加するとパケット転送性能が大きく低下する．一方，SR-IOV を利用した場合は，PCI カード内部のハードウェアスイッチ，ハイパーバイザ外部のハードウェアスイッチのどちらを利用した場合 (図中 (3) および (4)) でも，パケット転送性能は低下しない．

仮想スイッチと仮想 NIC を利用した場合に性能が低下する理由は，仮想スイッチから VM の仮想 NIC にパケットのデータを転送する際に発生するカーネルとユーザスペースでのデータコピーや，ハイパーバイザからのコンテキストスイッチのオーバーヘッドがあるためである [35]．

検討の結果，SR-IOV を利用してハイパーバイザ外部のハードウェアスイッチもしくは PCI カード内部のハードウェアスイッチを利用した VM の連結は，ShowNet 2014 における VNF 構成 (同一ハイパーバイザ内で仮想スイッチを利用した VM の

連結) と比較して，VNF の連結による性能低下の抑制に適した接続方式であることがわかった．

VM の資源割り当て方式

図 4.4 に VM に割り当てる資源の量とパケット転送性能の関係を示す．評価対象は，図 4.2 の (1) および (3) の VNF 構成かつ VM の連結数が 1 の場合である．縦軸は 1 つの VM に対して，1 つの CPU コアを割り当てた場合を 100 としたパケット転送性能を示し，横軸は VM に割り当てた CPU コア数を示す．

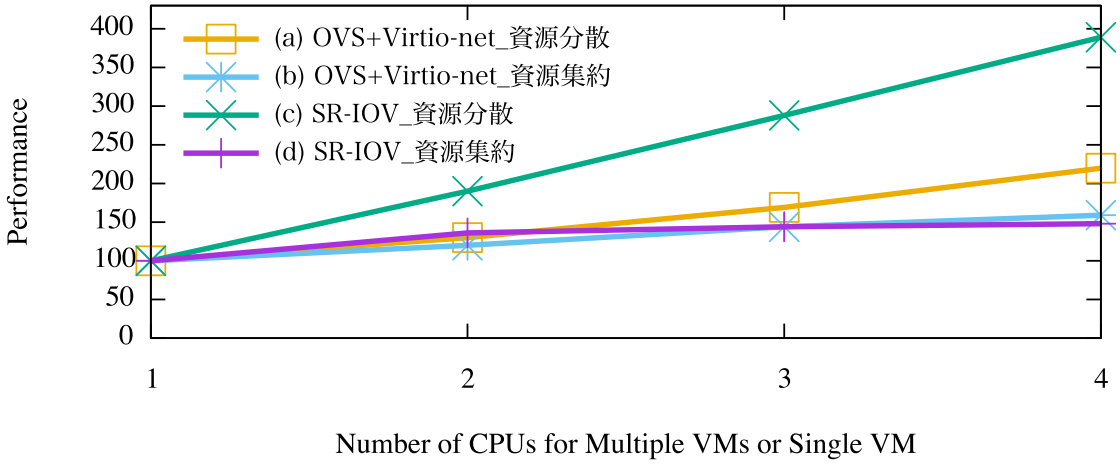


図 4.4: VM 資源割り当て方式の評価，IP Forwarding

仮想スイッチと仮想 NIC を利用した資源分散をおこなった場合 (図中 (a))，パケット転送性能が線形に向上した．しかし，資源集約 (図中 (b)) をおこなった場合はパケット転送性能が線形に向上しなかった．一方，SR-IOV を利用した資源分散をおこなった場合 (図中 (c))，パケット転送性能が線形に向上した．しかし，資源集約をおこなった場合 (図中 (d)) は CPU コア 3 個以上になった時点からパケット転送性能が向上しなかった．SR-IOV を利用した資源分散をおこなった方が，仮想スイッチと仮想 NIC を用いた資源分散と比較して，性能の向上率が高かった．

SR-IOV を利用した資源集約をおこなった場合に性能が線形に向上しなかった理由は，実験に利用した PCI カードの VF に対して割り当てるキューが 2 個であり，3 個目以降の CPU コアが Receiver Side Scaling(RSS)[29] による負荷分散の対象にならないためである．なお，このキュー数はデバイスドライバの実装に依存する．SR-IOV を利用した資源分散の方が，仮想スイッチと仮想 NIC を利用した資源分

散と比較して性能の向上率が高かった理由は，前述の SR-IOV には仮想スイッチと仮想 NIC を利用した際に発生するオーバーヘッドがないためである．

図 4.5 に，特定のアプリケーションやプロトコルに特化した VNF 利用を想定し，VM で Firewall 機能を有効にし，100 個のルールを設定した場合の VNF 割り当て資源量とパケット転送性能の関係を示す．これらのルールは，全てのトラフィックで評価対象となるが，マッチしない条件設定としている．図 4.4 と同様に，資源分散をおこなった場合 (図中 (a)(c)) は性能が線形に向上し，資源集約をおこなった場合 (図中 (b)(d)) はスケールアウトに適した結果にはならなかった．

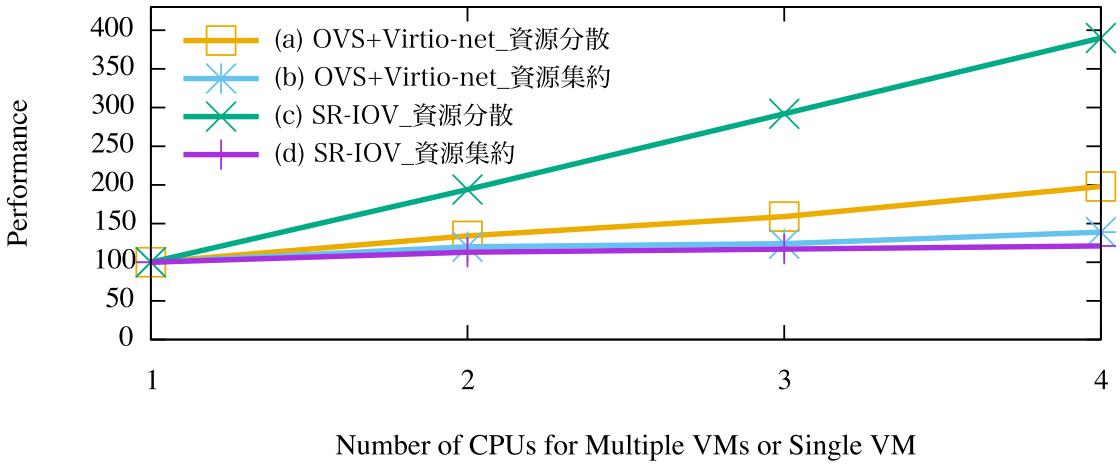


図 4.5: VM 資源割り当て方式の評価，Firewall

検討の結果，SR-IOV を利用した資源分散は，特定のアプリケーションやプロトコルに特化した VNF を利用した場合であっても，ShowNet 2014 における VNF 構成 (仮想スイッチと仮想 NIC を利用した資源集約) と比較して，スケールアウトに適した VM の資源割り当て方式であることがわかった．

4.4　FlowFall の設計・実装

前節で得られたスケールアウト可能な VNF 構成の知見に基づいて検討した FlowFall の設計・実装について述べる．FlowFall は vCPE サービスを実現するネットワークアーキテクチャである．FlowFall が vCPE の要件を満たしていることを示すため，アーキテクチャとトラフィック制御について説明すると共に，プロトタイプ実装による相互接続性とスケールアウトの検証結果を示す．

4.4.1　アーキテクチャ

FlowFall は，相互接続性の要件を満たすため，OpenFlow を利用してネットワークを構成する．図 4.6 に FlowFall のネットワーク構成を示す．FlowFall のネットワークは，複数の VNF レイヤ，バイパスリンク，CPE，アグリゲーションルータによって構成される．VNF レイヤは，複数の VM，ハイパーバイザ，OpenFlow スイッチによって構成され，複数の VNF レイヤを重ねることで複数のネットワーク機能の連結した SFC を構成する．バイパスリンクは，サービスチェイニングにおいて特定の VNF レイヤを通過する必要がないトラフィックの転送に利用するリンクである．アグリゲーションルータは，利用者ネットワークのアクセス回線を集約し，連結した VNF にトラフィックを転送する．CPE は利用者ネットワークのデフォルトゲートウェイとして動作し，利用者の要求するサービスに基づいて後述する IP ヘッダの Type-of-Service(ToS) フィールドを用いたサービス識別子を記述する．vCPE サービスの内部では，このサービス識別子に基づいてトラフィックが通過する VNF レイヤを決定する．

VNF レイヤは，スケールアウトの要件を満たすため，5.4 章の検討で得られた知見に基づいた構成となっている．図 4.7 に，ハイパーバイザにおける VM の構成を示す．各ハイパーバイザはアップリンクとダウンリンクの物理 NIC を搭載する．VNF を構成する VM のネットワーク I/O は，SR-IOV で分離された仮想 NIC(VF) を利用し，それぞれ異なる物理 NIC に所属する VF を割り当てる．VM の連結とトラフィックの分散は，物理 NIC が接続されている OpenFlow スイッチでおこなう．各 VM の CPU コアおよびメモリは，他の VM と重複しないように割り当てる．このような VM の構成によって VM 間の資源競合を排除し，複数の VM を用いたトラフィックの分散処理が可能になるため，VNF レイヤのスケールアウトが実現できる．この分散処理はハイパーバイザの追加にも適用可能なため，1 台のハイパーバイザが保持する CPU コア数やメモリ量に依存せず，VNF レイヤのパケット転送性能を向上できる．

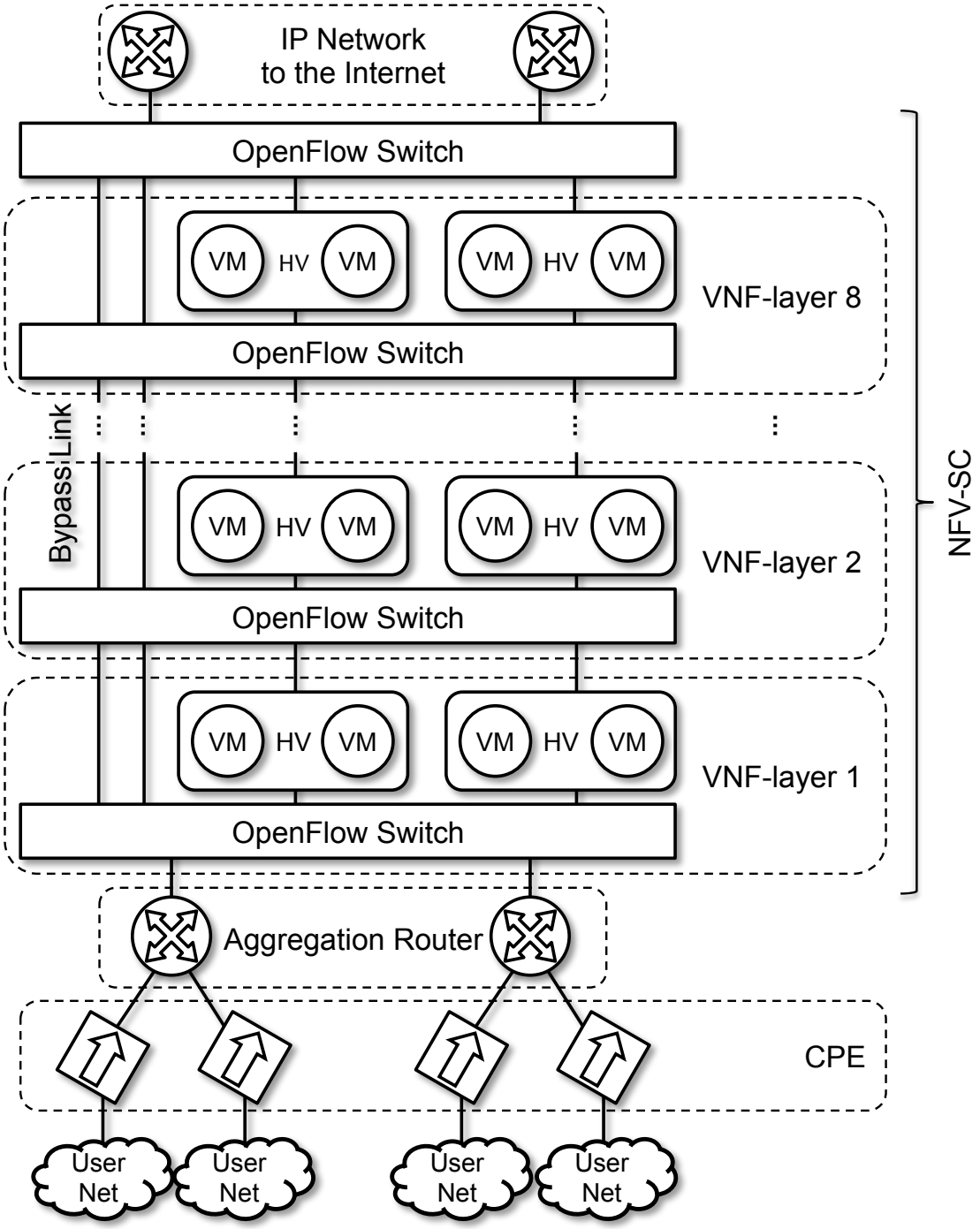


図 4.6: FlowFall のアーキテクチャ

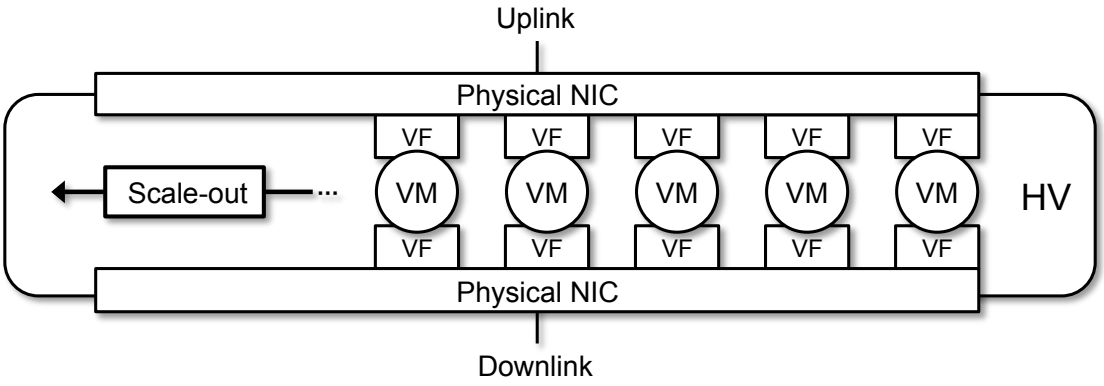


図 4.7: VNF レイヤにおける VM のネットワーク構成

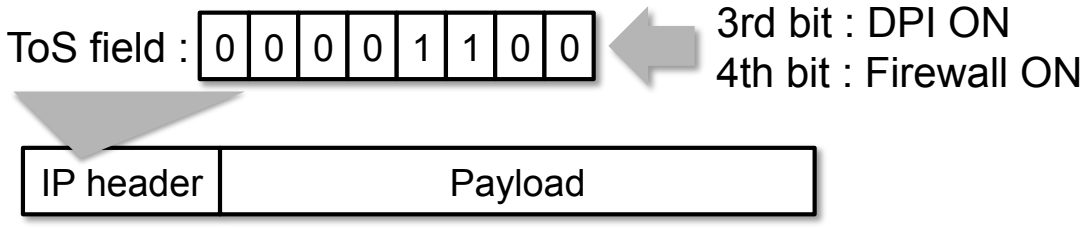


図 4.8: ToS フィールドをビットマップとしたサービス識別子

4.4.2　トラフィック制御

FlowFall におけるトラフィック制御は, 利用者ごとのサービスチェイニングを実現するため, IP ヘッダの ToS フィールドをサービスの識別子として用いる．OpenFlow スイッチは，ToS フィールドによってトラフィックの転送先を VM にするかバイパスリンクにするか，つまり各 VNF レイヤでネットワーク機能を適用するか否かを決定する．FlowFall におけるサービス識別子に ToS フィールドを選択した理由は，トラフィックごとに適用すべきサービスを識別でき，相互接続性を確保できるためである．例えば，ToS フィールドの書き換えは多くの CPE で可能であり，OpenFlow のマッチフィールドとしても利用可能である．

図 4.8 に，FlowFall における ToS フィールドを利用してサービスの識別子を記述する方法を示す．ToS フィールドは IP ヘッダに含まれる 8 ビット長のフィールドである．FlowFall における ToS フィールドの各ビットは，各 VNF レイヤの適用の有無を示す．例えば，DPI と Firewall サービスを通過するパケットでは，ToS フィールドの対応する 3 ビット目と 4 ビット目を 1 にする．利用者ネットワークを収容する CPE は，通過するパケットの ToS フィールドに利用者の選択したサービスに対応したビット列を設定する．

この ToS フィールドを用いたサービス適用の可否判断と VNF を構成する複数 VM へのトラフィックの分散のために，FlowFall は OpenFlow スイッチのポートをトラフィック制御の観点から 4 種類に分類して管理する．図 4.9 に FlowFall における OpenFlow のスイッチポートの分類を示す．Bypass Up(BU) ポートは上位 VNF レイヤの OpenFlow スイッチに，Bypass Down(BD) ポートは下位 VNF レイヤの OpenFlow スイッチに接続する． VNF Up(VU) ポートは上位 VNF レイヤのハイパーバイザに，VNF Down(VD) ポートは下位 VNF レイヤのハイパーバイザに接続する．同じ分類のスイッチポートが複数ある場合，OpenFlow スイッチはトラフィックの分散処理をおこなう．OpenFlow コントローラは，各 VNF レイヤに属する VM の接続情報を，OpenFlow スイッチのポートと宛先 MAC アドレスのペアで表現する．例えば，VU ポートではハイパーバイザと接続されたポートと，そのハイパーバイザ上に存在する VM の MAC アドレスによって表現する．これらの情報は，VM の起動と終了に応じて動的に設定する．

出展者収容ネットワークからインターネットに向けたトラフィックの転送は，OpenFlow コントローラが予め設定された出展者収容ネットワークの IP アドレスと ToS フィールドの値を用いて，OpenFlow スイッチにフローエントリを設定することで実現する．OpenFlow コントローラは，パケットの受信ポートが VD もしくは BD ポートかつ，受信パケットの ToS フィールド上で該当する VNF レイヤのビットが 1 の場合，VM の接続情報から IP アドレスの MD5 ハッシュ値に基づい

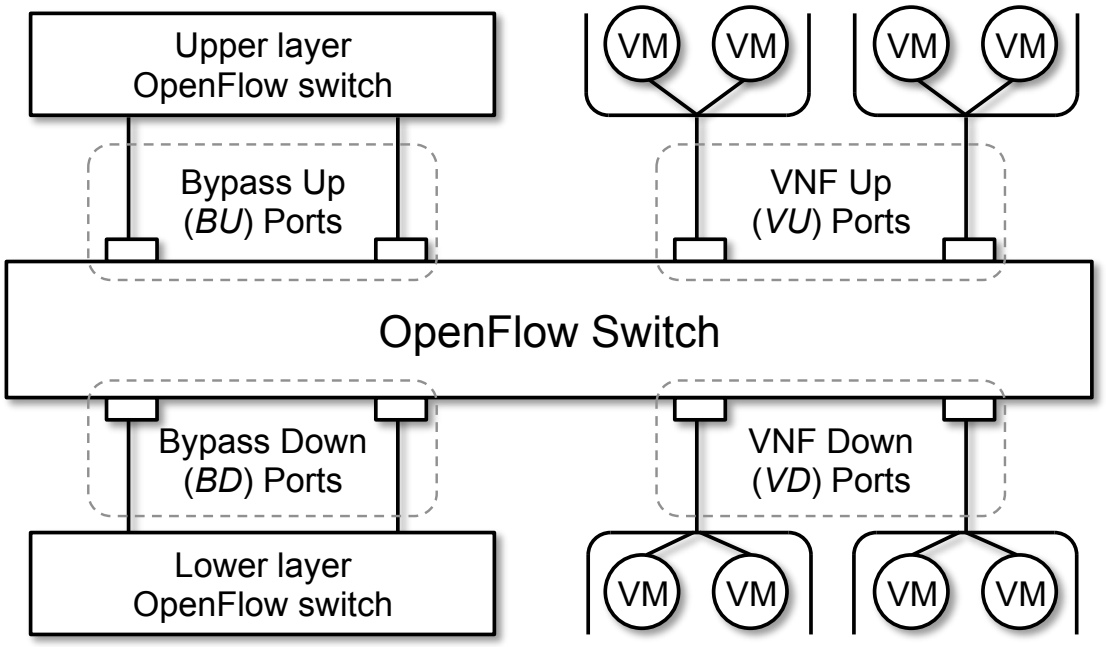


図 4.9: FlowFall における OpenFlow スイッチポート

て出力先 VU ポートと宛先 MAC アドレスを選出する．その後，受信ポート，ToS フィールド，送信元 IP アドレスをマッチフィールドとし，宛先 MAC アドレスを選出された MAC アドレスに書き換え (set-dl-dst)，選出された VU ポートに出力するアクションを持つフローエントリを設定する．受信パケットの ToS フィールド上で当該 VNF レイヤのビットが 0 の場合は，バイパスリンクに対して同様の処理をおこなう．

インターネットから出展者収容ネットワークに向けたトラフィックの転送は，出展者収容ネットワークからインターネットに向けたトラフィックの転送に利用したフローエントリの送信元 IP アドレスおよび MAC アドレスを宛先とし，VD または BD ポートを出力先としたフローエントリを設定することで実現する．このように VNF レイヤごとに上下対称にフローを設定することで，あるフローが通過する VM の列が上下対称になる．これによって，セッション単位の処理が必要な Firewall や DPI といったネットワーク機能が利用可能となる．

4.4.3　プロトタイプ実装による検証

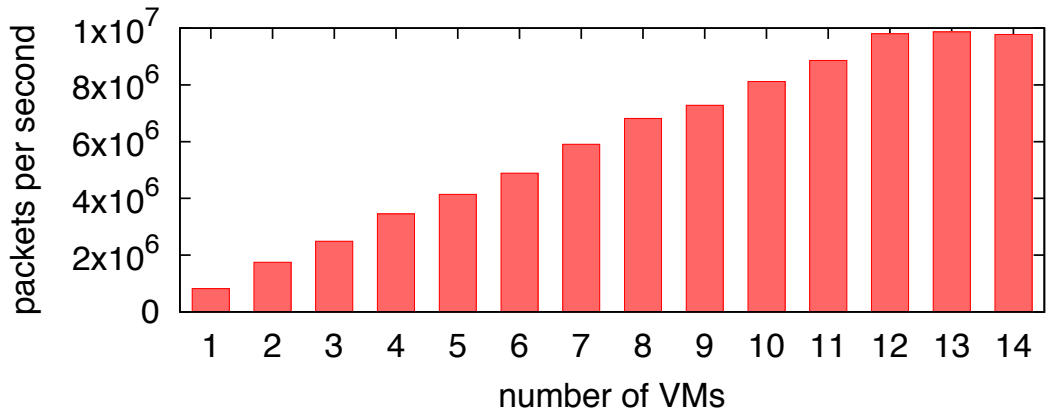
FlowFall の動作確認のためにプロトタイプを実装し，相互接続性とスケールアウトについて検証した．その結果，相互接続性については定性的，スケールアウトについては定量的な評価によって要件を満たしていることを確認した．

相互接続性

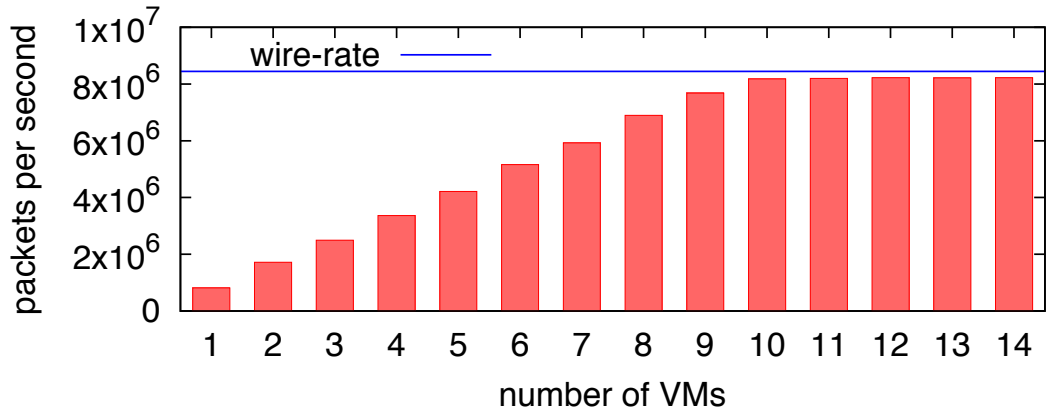
FlowFall は，汎用的なハードウェアと標準化された技術によって構成されており，プロトタイプ実装においても相互接続性を実現している．FlowFall のネットワークは，x86 アーキテクチャの汎用サーバと OpenFlow 1.3 に準拠した汎用 OpenFlow スイッチによって構成され，サービスチェイニングのトラフィック制御に用いた識別子は，利用者の IP アドレスと ToS フィールドである．また，VNF レイヤは，VM のネットワーク I/O 技術に SR-IOV を利用し，連結方式にハイパーバイザ外部のハードウェアスイッチを利用しており，一般的に入手可能な VNF 製品で動作可能な構成である．つまり，FlowFall が利用するデータプレーンおよびコントロールプレーンは標準化された技術であり，様々な製品を組み合わせてネットワークを構成することができる．

スケールアウト

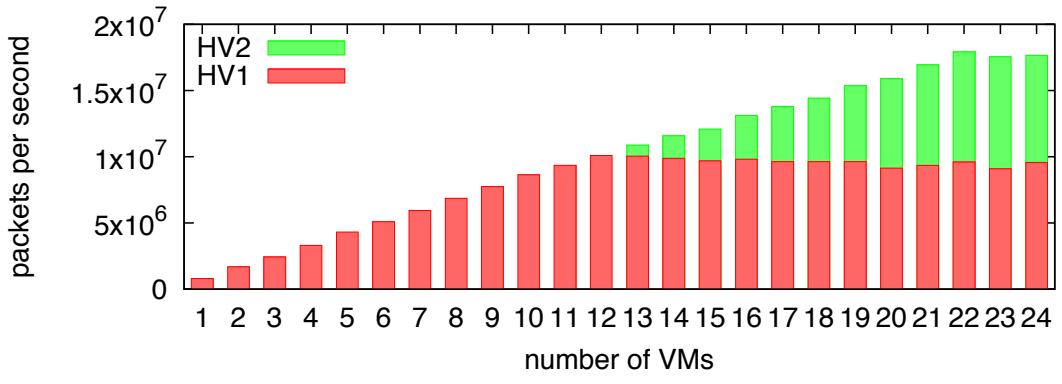
FlowFall は，VNF レイヤへの VM およびハイパーバイザの追加によるスケールアウトを実現している．図 4.10 に，1 台のハイパーバイザにおいて起動する VM を増加させていった場合のパケット転送性能 (フレームサイズ 64 バイト，128 バイト) と，ハイパーバイザを追加した場合のパケット転送性能 (フレームサイズ 64 バイト) を示す．1 台のハイパーバイザにおいて起動する VM を増加させていった結果，フレームサイズ 64 バイトでは，12 台の VM まで性能が線形に向上し，約 7Gbps のパケット転送性能を達成した．フレームサイズ 128 バイトでは，10 台の VM まで性能が線形に向上し，10 Gigabit Ethernet の理論値性能を達成した．また，ハイパーバイザを追加することによって，パケット転送性能が線形に向上することを確認した．



(a) ハイパーバイザ 1 台，フレームサイズ 64 バイト



(b) ハイパーバイザ 1 台，フレームサイズ 128 バイト



(c) ハイパーバイザ 2 台，フレームサイズ 64 バイト

図 4.10: 単一 VNF レイヤにおける VNF 数とハイパーバイザ数を増加させた場合のパケット転送性能

4.5　ShowNet における FlowFall の構築・運用

本節では，ShowNet 2015 で構築した FlowFall のネットワーク構成，出展者収容ネットワークとして運用した結果から見えてきた課題について述べる．

4.5.1　ShowNet における FlowFall の設計と構築

ShowNet 2015 の出展者収容ネットワークは，全て市販されている汎用サーバ，VNF 製品，ネットワーク装置によって構成された．表 4.3 に ShowNet 2015 の出展者収容ネットワークに利用したネットワーク機器の一覧を示し，図 4.11 にそれらを用いたネットワーク構成を示す．利用したハードウェアは，OpenFlow スイッチと x86 アーキテクチャの汎用サーバである．OpenFlow スイッチには NEC PF5248 と PF5459 を利用し，ハイパーバイザとして用いる汎用サーバには Dell PowerEdge R630，Huawei FusionServer X6800，Dell PowerEdge C6220 を利用した．これらのハードウェアを用いて，3 つの VNF レイヤから成る FlowFall を構築した．VNF レイヤは 3 つの機能によって構成した．1 つ目の VNF レイヤは Palo Alto Networks の PA-VM を利用したアプリケーションレベルでのトラフィック解析機能，2 つ目の VNF レイヤは Cisco Systems の CSR1000v を利用した Firewall 機能，3 つ目の VNF レイヤは A10 Networks の Thunder 6536 TPS を利用した DDoS 対策機能である．Thunder 6536 TPS は専用ハードウェアを利用したネットワーク機器であるが，FlowFall のアーキテクチャでは VNF がハードウェアか VM かを区別する必要はないため，動作確認のために導入した．

CPE には Juniper Networks の vSRX を用いて，NAT 機能と，出展者の適用するサービスを ToS フィールドに設定する機能を実装した．NAT 機能を VNF レイヤではなく CPE レイヤに実装した理由は，FlowFall におけるトラフィック制御を容易にするためである．FlowFall は，インターネットから出展者収容ネットワークに向けたトラフィックの識別に，送信元 IP アドレスを利用するため，NAT が送信元 IP アドレスを変更した場合，FlowFall のトラフィック制御で利用する送信元 IP アドレスが 2 種類になり，OpenFlow コントローラにおける管理が複雑になる．そのため，パケットの送信元 IP アドレス，ポート番号を変換する NAT は CPE に実装した．

出展者ごとのサービス要求に基づいた動的なネットワーク構成変更への対応については，出展者収容ネットワークに対して利用する VNF を制御可能な Web ポータルを用意した．出展者が利用するサービスを Web ポータルから指定することで，トラフィック制御に利用する ToS フィールドの値を決定し，vSRX に対して Netconf

を利用して設定変更を行い，出展者収容ネットワークから出力されるトラフィックに ToS フィールドの値を設定する．

表 4.3: ShowNet 2015 における FlowFall で利用したネットワーク機器

装置名	タイプ	役割	ハイパーバイザ
A10 Thunder 6435 TPS	Hardware	DDos Mitigation	N/A
Cisco CSR1000v	VNF	Firewall	Dell PowerEdge C6220
PaloAlto PA-VM	VNF	DPI	Huawei Fusion Server X6800
Juniper EX4600	Hardware	Aggregation Router	N/A
Juniper vSRX	CPE	NAT	Dell PowerEdge R630

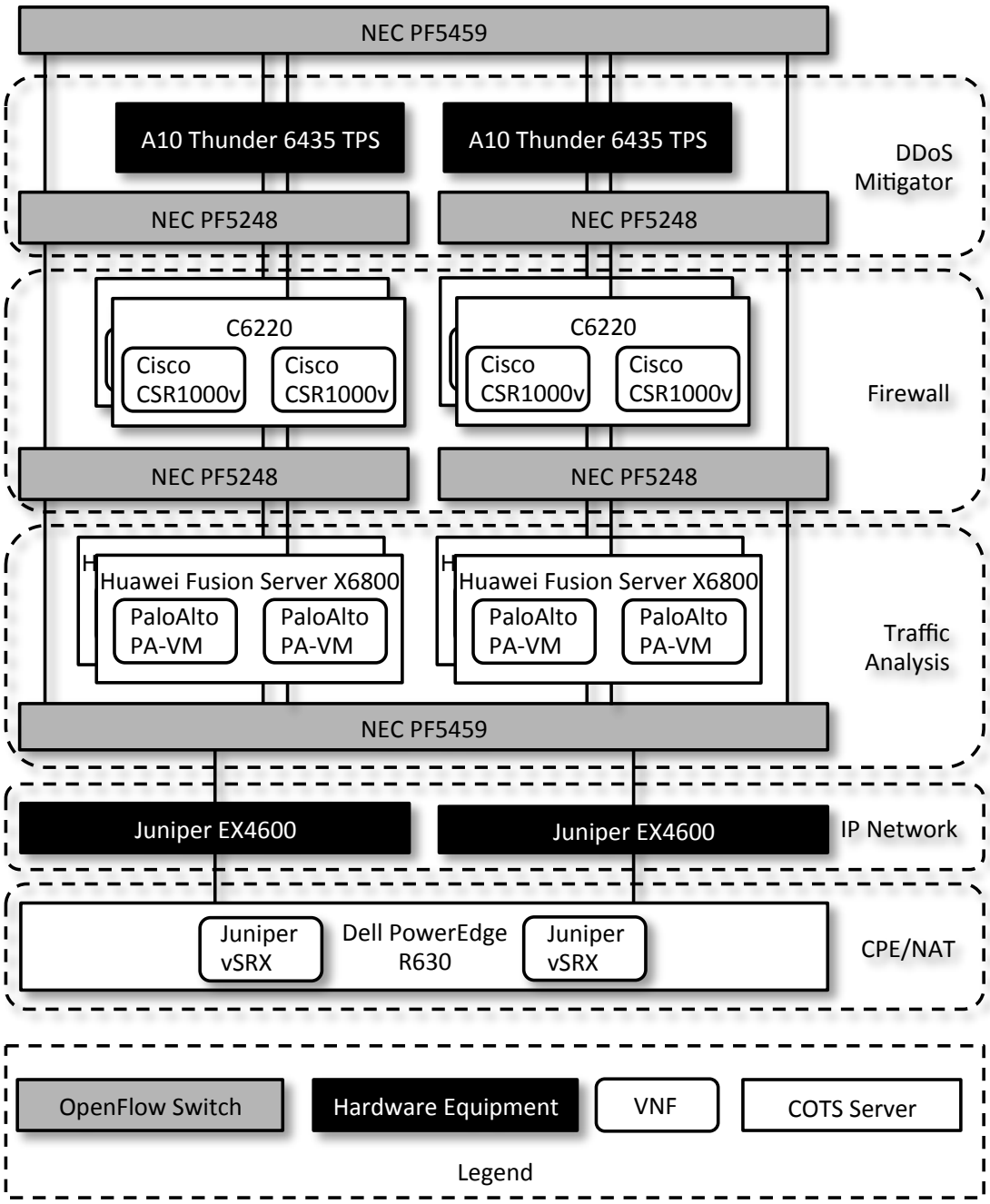


図 4.11: ShowNet 2015 における FlowFall のネットワーク構成

4.5.2　ShowNet における FlowFall の運用課題

FlowFall は ShowNet における vCPE サービスの要件である相互接続性とスケールアウトを満たし，ShowNet 2015 での運用を通して実用性を示すことができた．しかし，ShowNet において FlowFall をより安定して運用するためには，少なくとも FlowFall に特化した死活監視機能と効率的なトラフィックの分散が必要である．

FlowFall に特化した死活監視機能

FlowFall では，スケールアウトを実現するトラフィックの分散に IP アドレスのハッシュ値を用いている．このため，外部のネットワークから任意の VM に対して Ping などを用いた一般的な IP ネットワークにおける死活監視の手法を適用できない．

この課題については，Link Layer Discovery Protocol(LLDP) を用いた OpenFlow ネットワークにおける死活監視手法 [48] を VNF およびリンクの監視に応用することで解決できると考えている．このような手法を FlowFall に実装するためには，OpenFlow コントローラが packet-in および packet-out メッセージを用いて OpenFlow スイッチ経由で LLDP パケットを送受信することで，リンクの状態を管理する必要がある．

効率的なトラフィックの分散

FlowFall では，パケットの送信元 IP アドレスをキーとしたハッシュアルゴリズム (MD5) を利用して，あるフローを処理する VM を決定している．このため，フローごとのトラフィック量が異なる実際のネットワークでは，同一 VNF レイヤ内において均等にトラフィックを分散できない場合がある．

この課題については，ネットワーク機器に入力されるトラフィックの傾向を解析し，動的に転送先を変更するトラフィック分散手法 [49] を，同一 VNF レイヤの VM に対して入力するトラフィック分散に応用することで解決できると考えている．このような手法を FlowFall に実装するためには，sFlow[50] に対応した OpenFlow スイッチを利用し，OpenFlow コントローラでトラフィックの傾向を解析した結果に基づいて転送先の VM を変更する必要がある．

4.6　本研究によって得られた知見

本研究によって得られた vCPE における相互接続性とスケールアウトに関するプラクティスと，それに伴うトレードオフについて述べる．

4.6.1　相互接続性とスケールアウトに関するプラクティス

FlowFall の設計・実装・運用から得られた vCPE における相互接続性とスケールアウトに関するプラクティスを示す．

相互接続性

vCPE で相互接続性を実現するには，VM のネットワーク I/O，VM の連結方式，VNF 間のトラフィック制御について考慮する必要がある．

(a) **VM のネットワーク I/O:** ハイパーバイザと VNF の両方が対応している技術が必要である．SR-IOV や Virtio-Net などの標準化もしくは仕様が公開された仮想デバイスの利用が有効である．

(b) **VM の連結方式:** VM のネットワーク I/O 技術と接続可能で，外部からトラフィック制御のシグナリングが可能な技術が必要である．OVS や OpenFlow スイッチなど，標準化されたコントロールプレーンプロトコルに対応した SDN スイッチの利用が有効である．

(c) **VNF 間のトラフィック制御:** VNF と CPE を含めた全てのネットワーク構成要素が対応しているデータプレーン技術が必要である．ToS フィールドなど標準化されたパケットヘッダのサービス識別子への利用が有効である．

スケールアウト

vCPE でスケールアウトを実現するには，VM 間における資源競合を回避し，複数の VM でトラフィックを分散処理可能な VM の連結方式，VM の資源割り当て方法について考慮する必要がある．

(a) **VM の連結方式:** 仮想 NIC や仮想スイッチなど，ハイパーバイザ内部におけるソフトウェアパケット処理による資源競合を回避可能な技術が必要である．SR-IOV とハイパーバイザ外部のハードウェアスイッチを利用したハイパーバイザ内のソフトウェアパケット処理のバイパスが有効である．

(b) **VM の資源割り当て方式:** VM のネットワーク I/O 技術が持つキュー数に非依存で，VM 内の資源競合を回避可能な技術が必要である．VM ごとに必要最小限の CPU コアを割り当てる資源分散が有効である．

4.6.2　トレードオフ

FlowFall には，vCPE サービスの要件を満たすことを優先した結果，トレードオフとして VNF の数，VNF の種類，VNF の順序に関して制約が発生した．

VNF の数

FlowFall では 8 ビットの ToS フィールドをサービス識別子に用いて，ToS フィールドの 1 ビットが 1 個の VNF を示す設計としたため，連結可能な VNF の数 (最大 8 個) に制約がある．ShowNet 2015 において VNF は 8 個で十分であったが，それ以上の数の VNF を利用するにあたっては，新たなネットワーク制御手法を検討する必要がある．

VNF の種類

FlowFall では ToS フィールドをトラフィック制御のサービス識別子に用いたため，IP ヘッダを書き換えるネットワーク機能が VNF レイヤで利用しにくい制約がある．例えば送信元 IP アドレスとポート番号を変換する Network Address Translation(NAT) 装置が ToS フィールドの値を書き換える場合，FlowFall のトラフィック制御で利用する送信元 IP アドレスが 2 種類になり，OpenFlow コントローラにおける管理が複雑になるため，ShowNet では NAT 機能は CPE 側で実装した．

VNF の順序

FlowFall では，VM のネットワーク I/O 技術にハードウェアによる補助 (SR-IOV) を用いたため，VNF レイヤ間の接続はハードウェアスイッチを介した静的なものとなった．そのため，VM 間の資源競合の排除によるスケールアウトを実現できたが，適用する VNF の順序は VNF レイヤの順序に依存し変更することができない．一方，仮想スイッチを用いれば，自由な順序での VNF の連結を実現できるが，性能は犠牲となる [51]．このように，ハードウェアによる補助を用いてスケールア

ウトを実現するか，パケットのソフトウェア処理を用いて柔軟性を実現するかは，トレードオフの関係にある．

4.7　VM を利用した NFV の課題整理

NFV の評価・検証および FlowFall の構築・運用から得られた知見にもどつき，VNF の可搬性およびパケット処理性能の観点から，VM を利用した NFV の課題を整理する．

4.7.1　VNF の可搬性

市販されている VNF は，VM のイメージとして配布されており，標準化されている NFV のフレームワークに対して，VNF は NFVI との境界を逸脱している．図 4.12 に，ETSI の NFV ISG が標準化している NFV のフレームワーク [5] と市販されている VNF の関係性を示す．標準的な NFV のフレームワークでは VM は NFVI を構成する要素の一部である．しかし，市販の VNF は VM のイメージを含んでおり，VNF と NFVI との境界を逸脱している．VM のイメージには，VNF の実行ファイルに加えて，オペレーティングシステムと VM の構成情報が含まれている．VM の構成情報には，仮想 CPU や割り当てメモリ量，仮想デバイスなどが記述されている．この記述には，オペレーティングシステムが保持するハードウェアリソースの識別子が利用される．

この境界は，VNF と NFVI のインターフェースとして Vn-Nf が定義されているが，実質的には形骸化している．Vn-Nf は，ハードウェアに非依存に VNF の要件 (ライフサイクル管理や性能など) を保証するものである．しかし，具体的な技術や仕様は定義されておらず，VNF が VM のイメージとして配布されている現状では，VNF の利用者が Vn-Nf インターフェースを意識することはなく，その役割を果たしているとは言えない．

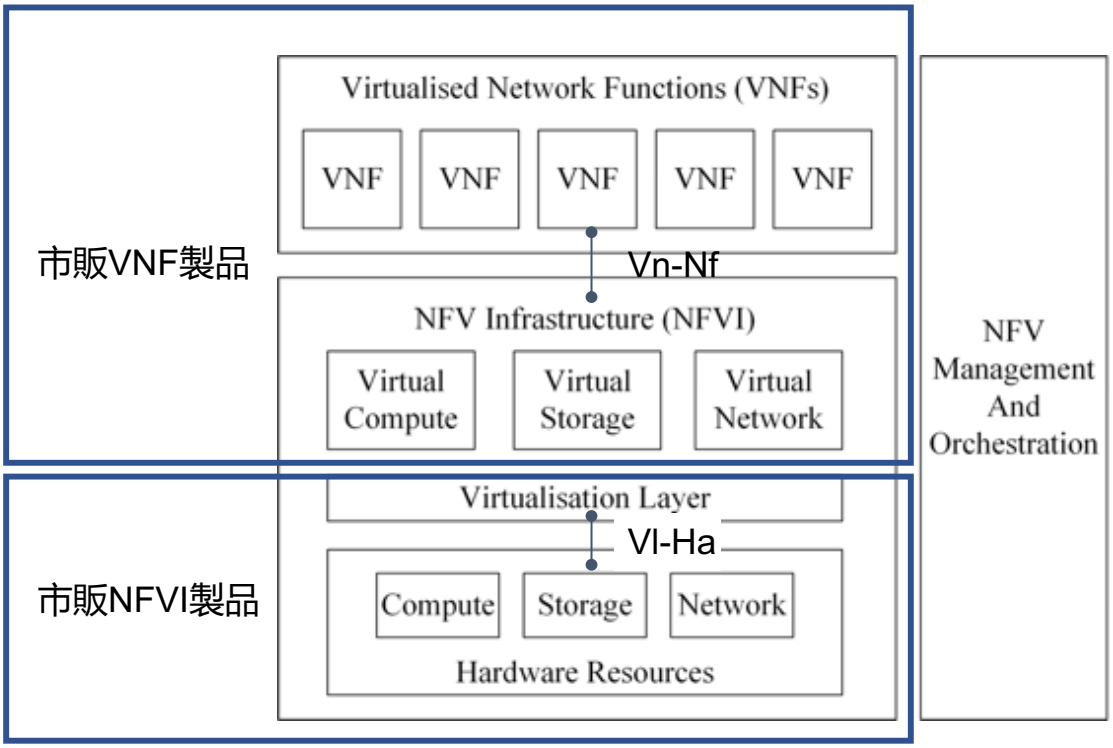


図 4.12: ETSI の NFV ISG が標準化している NFV のフレームワークと市販されている VNF の関係性 [24]

VNF を VM イメージとして配布することによって，VNF が NFVI との境界を逸脱し，Vn-Nf インターフェースが機能していない状況では，本来の NFV で期待されているハードウェアとネットワーク機能の分離が実現できていない．図 4.13 に理想的な NFV と VNF を VM イメージで抽象化した NFV の比較を示す．理想的な NFV は，異なるハードウェアの差分を，仮想化レイヤが抽象化して統一的なインターフェースを提供することによって，VNF に可搬性を提供する．しかし，VNF を VM イメージとして配布することによって，VNF の利用者が VM の構成情報を明示する必要が発生する．そのため，SR-IOV や DPDK といったハードウェアに特化したパケット処理性能の最適化を実装した場合，VNF はハードウェアと分離されているとは言えない．すなわち，VNF の可搬性が失われた状態になる．

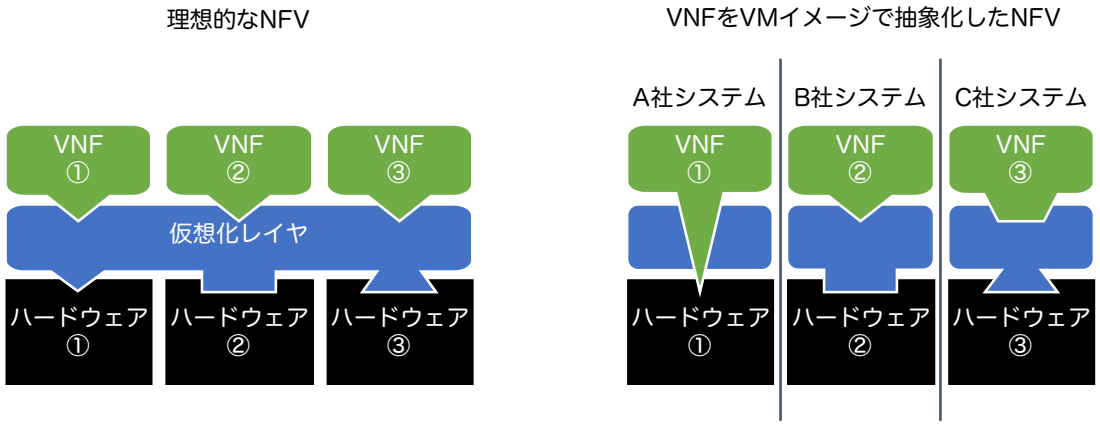


図 4.13: 理想的な NFV と VNF を VM イメージで抽象化した NFV の比較

4.7.2　VNF のパケット処理性能

VNF の可搬性を上げるため，ハードウェアの構成に非依存な仮想 NIC を利用した場合，VM を利用することによって発生するオーバーヘッドによって，高いパケット処理性能を提供するのが困難になる．その根本的な原因は，カーネルとユーザスペース間のデータコピー，ハイパーバイザとゲスト OS のコンテキストスイッチ，VNF に対して冗長なゲスト OS 機能である．以下にこれらの原因について述べる．

カーネルとユーザスペース間のデータコピー

VM に接続される仮想 NIC のバックエンドは，ホスト OS 上のカーネル内に実装される場合がある．一方で，VM はユーザスペースのプロセスである．したがって，仮想 NIC に対してパケットを入出力するためには，カーネルとユーザスペース間でメモリのデータコピーが発生する．

ハイパーバイザとゲスト OS のコンテキストスイッチ

ゲスト OS 上の仮想 CPU 上でセンシティブ命令が発生した場合，VM はハイパーバイザに対して VM_Exit を発行し，ゲスト OS からハイパーバイザへのコンテキストスイッチが発生する．その間，VM の仮想 CPU は停止状態になるため，その処理が完了するまではハイパーバイザ側の処理を待たなくてはならない．センシ

ティブ命令には，VM のネットワーク I/O における仮想 NIC のレジスタアクセスや，メモリマップド I/O，割り込み制御が含まれる．大量のパケットを送受信する事が想定される通信事業者のミドルボックスの利用には不向きであると言える．

VNF に対して冗長なゲスト OS 機能

通常，VM 上で動作するゲスト OS はホスト OS と同じソフトウェアが用いられる．本来 OS の役割は，ハードウェアの資源管理，アプリケーション（プロセス）間のメモリ空間の分離，マルチタスクのスケジューリングなどである．しかし，VM の実運用シナリオとして，単一アプリケーション機能のみが実装される場合も多く，OS の一部の機能は不要であり，オーバーヘッドとなっている場合がある．

4.8　本章のまとめ

VM は計算機という点では x86 アーキテクチャで抽象化されているが，ネットワークという点では NFVI が提供する多様な仮想 NIC の実装があり，その構成によって最適化方法が異なる．特に Virtio-Net を利用した純仮想 NIC の実装では，ネットワーク I/O が集中した場合の資源競合を抑制するのが困難であり，実質的にはハードウェアのパーティショニングによる仮想化を利用しなければ，スケールアウトの実現は困難である．この知見を元に，本研究では FlowFall を設計・実装し，10Gigabit Ethernet の理論値に近いパケット処理性能を実現し，Interop ShowNet において商用の vCPE サービスを提供できた．しかし．SR-IOV によるハードウェアに特化したパケット処理性能の代償として，SFC の運用上における課題などが明らかになった．

上記の知見から得られる VM を利用した NFV の問題点は，VNF の可搬性とパケット処理性能の両立である．VNF の可搬性おいて，VM を利用する市販製品は標準的な NFV のフレームワークにおける VNF と NFVI の境界を逸脱しており，統一されたインターフェースによって抽象化がされていない．その結果，VNF が VM の構成に特化した最適化を実施した場合，可搬性が失われる．一方，VNF のパケット処理性能において，あらゆる NFVI で統一された仮想 NIC を利用し，VM 上のオペレーティングシステムで共通のデバイスドライバを利用する場合，様々なオーバーヘッドがあり，可搬性とパケット処理性能は両立しない．

第5章　コンテナ型NFVアーキテクチャの提案

本章では，VMを利用したNFVが根本的に抱える課題である性能と柔軟性の両立を解決するため，本研究の提案手法であるコンテナ型VNFアーキテクチャについて述べる．はじめに，アーキテクチャの概要について述べ，次にプロセスによるVNFの抽象化，VNFのコンテナ化，サービスチェイニング方法について述べる．

5.1　コンテナ型NFVアーキテクチャの概要

コンテナ型NFVアーキテクチャは，2章で述べたネットワークサービスのPaaS化を実現させるためのアーキテクチャであり，NFV領域における要件であるVNFの可搬性とパケット処理性能を両立させる技術である．本アーキテクチャでは，VNFをコンテナの集合体として取り扱い，コンテナのオーケストレータによってVNFのライフサイクルを管理する．ネットワークを利用するアプリケーションは，オーケストレータに対してネットワークスライスに対する操作 (新規作成，構成変更，削除) を要求し，オーケストレータはリアクティブに実施する．アプリケーションとオーケストレータ間のインターフェースでは，ネットワークトポロジや経路制御の情報など具体的な情報を含まず，アプリケーションの視点での抽象度の高いネットワークに対する通信品質の要求が交換される．

図5.1にコンテナ型NFVアーキテクチャの概要を示す．本アーキテクチャは，オーケストレータ，NFVI，VNF，SRv6 Router，IP Networkから構成される．オーケストレータは，VNFFGの入力をトリガーとしてNFVIに対してVNFコンテナの起動およびVNFに割り当てたIPv6アドレスの広報を依頼する．IPv6アドレスはVNFの種類に応じてユニークなIDを割り当てる．すなわち，同じ役割を持つVNFは，異なるコンテナでも同じIPv6アドレスを利用する．次にVNFFGの記述に基づき，あるフローが通過するべきVNFの順序をIngressとEgressのSRv6 Routerに設定する．この順序はSRv6 Headerを利用して記述される．

図5.2に，オーケストレータとNFVIの関係を示す．オーケストレータは，Kuber-

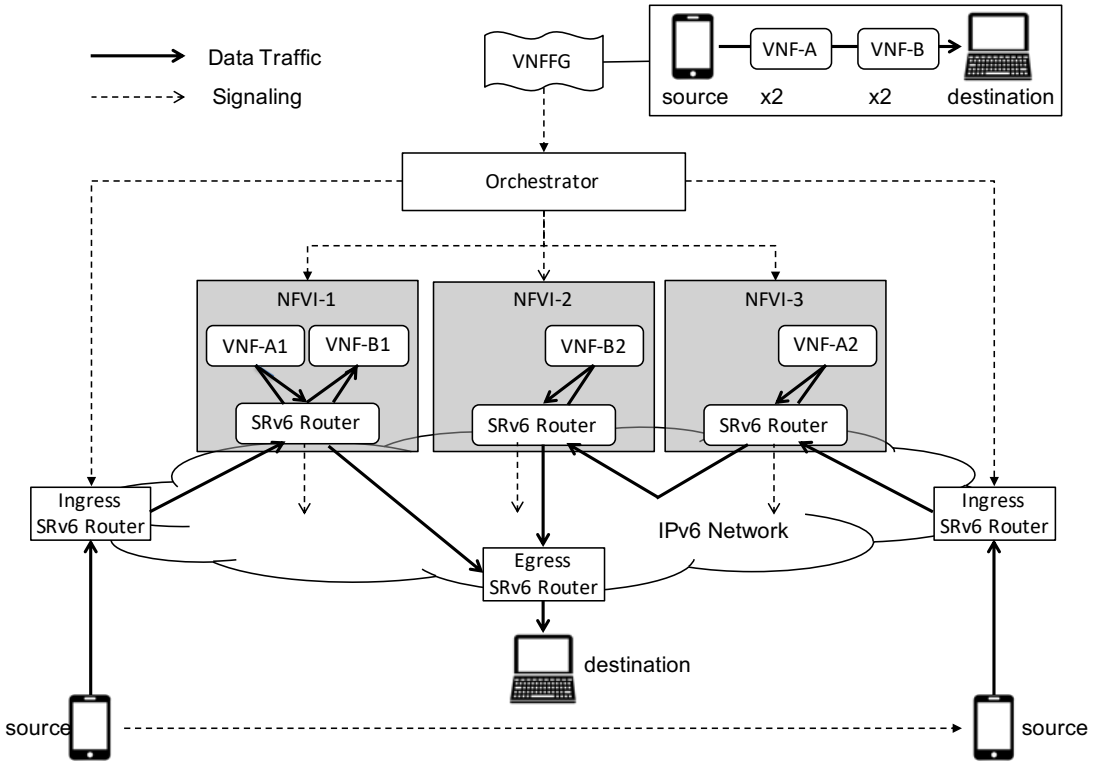


図 5.1: ネットワークのアーキテクチャ

netes など一般的に普及しているコンテナのオーケストレーションシステムを想定する．オーケストレータは，VNFFG から入力されたポリシー情報に基づき，コンテナを起動しネットワークの設定をおこなう．ポリシー情報には，コンテナの冗長性，自動回復，NFVI の条件などが記述されており，それらに基づいて適切な NFVI を選択してコンテナを起動する．これらの役割は，Scheduler と Manager によって実現される．コンテナの起動およびネットワークの設定は，Container Runtime Interface(CRI) や Container Network Interface(CNI) などの標準化されたシグナリングを利用して実施され，NFVI の実装に非依存な抽象化を提供する．

コンテナ型 NFV アーキテクチャの利点は，VNF の可搬性を確保するのが容易な点である．コンテナはプロセスの集合体であり，その実体はユーザスペースで動作するプログラムの実行ファイルである．ユーザスペースで動作するプログラムは，カーネルが提供するシステムコールを介して，カーネルが管理するハードウェアのリソースにアクセスする．したがって，ハードウェアのプラットフォームが同じであれば，Application Binary Interface(ABI) の互換性が保たれている限り，オペレーティングシステムの実装に非依存に実行することができる．また，昨

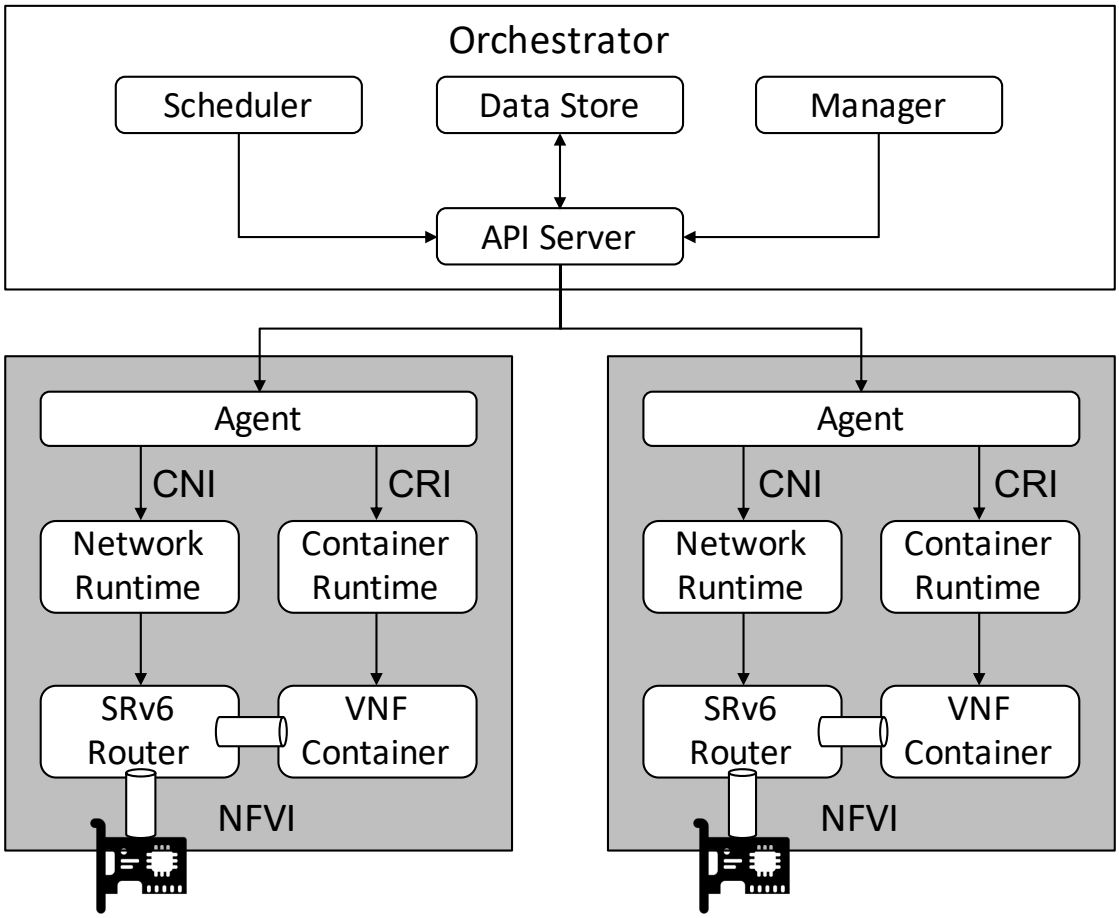


図 5.2: オーケストレータと NFVI の関係性

今のコンテナのデプロイシステムはマルチアーキテクチャ対応を進めており，異なるハードウェアプラットフォームにおいても，コンテナを登録しているレジストリから実行環境に合致したバイナリを取得することができる．したがって，オペレーティングシステムやハードウェアのプラットフォームに依存しない可搬性を確保できる．

もう一つの利点は，VM と密結合の VNF と比較して軽量な点である．従来の VM を利用した NFV と比較して，VNF の実行ファイルから，オペレーティングシステムを削除することによって，VNF を構成する実行ファイルのサイズを小さくすることができる．NFVI に対して VNF をダウンロードするために必要な時間を短縮できるだけでなく，より多くの VNF の実行ファイルを NFVI にキャッシュできる．その結果，VNF の運用方法が，あらかじめ起動しておくプロアクティブな手

法から，VNF が必要となるアプリケーションの通信の検出をトリガーとしたリアクティブな手法に変化し，資源の有効活用が可能になる．

図 5.3 に，NFVI のアーキテクチャを示す．NFVI は，VNF Container と SRv6 Router で構成される．VNF Container はプロセスの集合であり，ネームスペースや Cgroups などのオペレーティングシステムが提供するリソースの隔離・制限をする機能を利用してコンテナ化する．SRv6 Router は VNF Container に対して仮想 NIC を提供する．カーネルは仮想 NIC のパケットバッファに対してユーザスペースから直接アクセス可能な API を提供する．その結果，VNF Container で動作するプロセスは，オペレーティングシステムが持つネットワークスタックをバイパスし，カーネルからユーザスペースへのデータコピーを介さずに仮想 NIC にアクセスする．

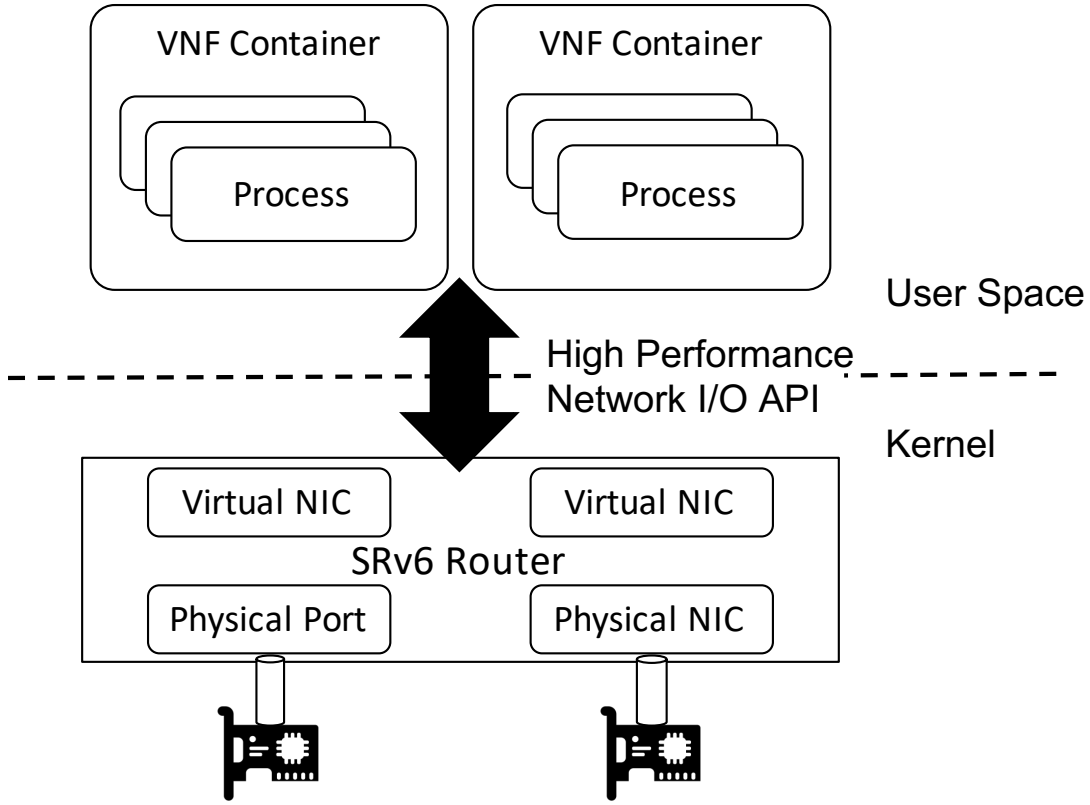


図 5.3: NFVI のアーキテクチャ概要

5.2　プロセス型 VNF の設計

本研究が提案するコンテナ型 NFV アーキテクチャにおける根幹である NFVI と VNF のアーキテクチャを実証するためにおこなった，プロセス型 VNF の設計について述べる．

コンテナ型 NFV アーキテクチャでは，VNF をオペレーティングシステムのプロセスの集合として抽象化する．図 5.4 にプロセス型 VNF の設計を示す．プロセス型 VNF では，ユーザスペースのプロセスとして動作する VNF と，カーネルで動作する Bridge で構成される．Bridge は，VNF に仮想ポートを提供しサービスチェイニングにおけるトラフィック制御を担当する．VNF は，仮想ポートから入力されたパケットに対してネットワーク機能としてパケット処理を担当する．VNF は，ユーザスペースでパケット処理をおこなう Pre-Processing，VNF-Specific-Processing，Post-Processing で構成される．VNF は Bridge の仮想ポートがカーネル内に持つパケットバッファと，自身がユーザスペースに持つリングバッファをマッピングし，パケットの入出力をカーネルとユーザスペース間のメモリコピーを介さない参照渡しで実施する．以下に各パーツの動作について説明する．

Bridge

VNF に対して仮想ポートを提供し，VNF 間もしく VNF と物理 NIC 間のトラフィックを中継する．VNF に対して，仮想ポートのパケットバッファを参照可能な高速ネットワーク I/O の API を提供する．また，ネットワークコントローラと接続するインターフェースを持ち，動的なトラフィック制御を実現するインターフェースを持つ．

Pre-Processing

Bridge が提供する高速なネットワーク入出力 API を利用して VNF に着信したパケットを取得し，そのパケットに対して転送，処理，破棄のいずれを実施するか判別する．パケット処理が必要な場合は，続く VNF-Specific-Processing を呼び出し，転送する場合は Post-Processing を呼び出す．

VNF-Specific-Processing

VNF 自身が提供するパケット処理を実行する．この機能は実装する VNF に特化した内容となるため，特に動作に関する定義を行わない．

Post-Processing

Pre-Processing もしくは VNF Specific Processing から，外部へ送信すべきものとして送られてきたパケットに対して，VNF の種別に基づき必要なプロトコルスタック処理をおこなった後，Bridge が提供する高速なネットワーク入出力 API を利用して，パケット送信処理をおこなう．

Pre-Processing，VNF-Specific-Processing，Post-Processing を複数のプロセスに分割し，並列処理を実施する場合は，実現する VNF のパケット処理モデルに応じて同期手法を検討する必要がある．例えば，テールドロップのようにパケットの受信キュー長を超えるパケットの入力を破棄するモデルでは，セマフォのような同期手法を利用する必要がある．パケットの受信から送信までの一連の処理にアトミック性を持たせる必要がある場合は，ミューテックスを利用する必要がある．このように，VNF のパケット処理に応じて，設計者が適切な同期手法を選択する必要がある．

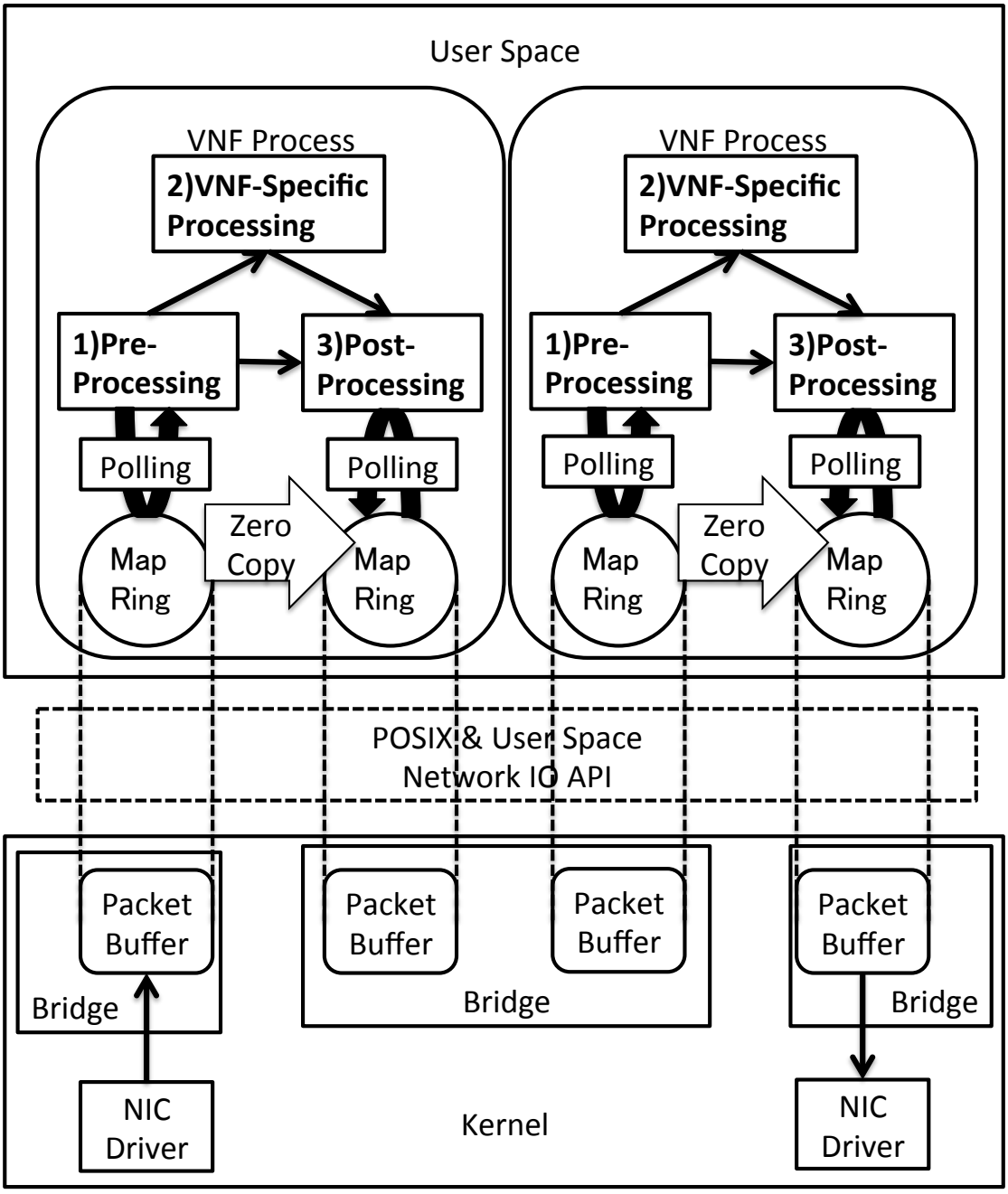


図 5.4: プロセス型 VNF アーキテクチャの概要

5.3　プロセス型 VNF の実装

本節では，プロセス型 VNF アーキテクチャをモデルとして，実際の VNF をプロセス上に実装する手法に関して説明する．

Bridge と高速ネットワーク I/O API

本研究ではプロセス型 VNF アーキテクチャの高速ユーザスペースネットワーク I/O 技術に Netmap を，本アーキテクチャの Bridge に VALE[52] を利用する．VALE はカーネル内のブリッジとして動作し，Netmap API をユーザスペースに対して提供する高速なユーザスペースネットワーク I/O 技術である．

VALE は Netmap API を利用してポーリングベースのパケット入出力インターフェースを提供するため，Pre-Process と Post-Process と接続が可能である．また，ゼロコピーが可能な事から，ネットワーク機能の抽象化とネットワーク I/O 性能を両立できる．

VALE は OpenFlow など Ethernet のスイッチング以外の機能も適用可能なため，サービス連結における経路制御を担当可能であり，VNF から複雑なサービス連結に関係する複雑な経路制御を分離できる．Netmap はユーザスペースに対してネットワーク I/O 以外のカーネルメモリ空間を解放しないため，意図せずプロセスがカーネルをクラッシュさせる心配がなく，マルチテナンシーに適合する．また，Netmap は専用ハードウェアのオフロード機能に非依存で，デバイスドライバが対応すれば，一般的なネットワークカードでも動作するため，プロセス型 VNF のネットワーク機能の抽象化と合致する．

VNF に特化したプロトコル処理

ユーザスペースのネットワーク I/O はカーネルをバイパスするため，プロトコルスタックの処理をユーザスペースの VNF 自身で実装する必要がある．本アーキテクチャでは VNF の種類や構成に依存せずモデルを保ち，Pre-Processing と Post-Processing に VNF の種類に特化した処理を当てはめることで，様々な VNF の実装に対応する．ミドルボックスが処理するプロトコルスタックの性質は，ミドルボックスがトラフィックの質に応じて，通信の方向性（片方向もしくは双方向），処理する通信の単位（パケット単位もしくはフロー単位）の違いがある．

表 5.1 に実際に対応するミドルボックスの種類と実装すべき Pre-Processing と Post-Processing の関係性を示す．

表 5.1: 通信の方向性，トラフィック処理単位と実装すべき機能

通信の方向	片方向	双方向	双方向	Pre-Process と Post-Process
処理の単位	パケット毎	パケット毎	フロー毎	
レイヤ 2 透過機器	スイッチ		DPI	FDB 探索
レイヤ 3 終端機器	ルータ	NAT		ARP 解決と MAC 入れ替え IP 経路表探索，TTL 減算 チェックサム計算
レイヤ 4 終端機器	Stateless Firewall	NAPT	Stateful Firewall	TCP の再構成
レイヤ 7 終端機器			プロキシ キャッシュ	アプリケーション プロトコルの終端

NAT 機能を例にした VNF の実装例

図 5.5 に本アーキテクチャの設計思想に基づいて作成した NAT VNF の概要を示す．

Pre-Process は VALE ポートを Open し，Netmap の機能を利用してポーリングによって受信リングバッファにデータが書き込まれたことを検知する．受信リングバッファのメモリ番地が格納されたユーザスペースのリングにアクセスする事で，実際のパケットデータにアクセスする．パケットの着信とともに IP の経路表が格納されているパトリシアツリーに対して，パケットの宛先プレフィックスを検索し，存在しなければパケットを破棄，存在すれば VNF Specific Process に進む．

次に，VNF Specific Process が NAT の変換表が格納されているハッシュテーブルに対して，変化すべき送信元 IP アドレスを検索し，存在しなければパケットを破棄，存在すれば送信元 IP アドレスを変換し，Post-Process に進む．

Post-Process ではレイヤ 3 終端機器としておこなうべき TTL の減算，宛先 MAC アドレスを次の VNF，もしくは外部のサービス連結先のものに変換，チェックサム計算をおこなったのち，送信処理をおこなう．送信処理は Netmap の機能を利用して受信バッファのメモリ番地を送信バッファのメモリ番地とスワップすることで，ゼロコピーでパケットデータを送信する．

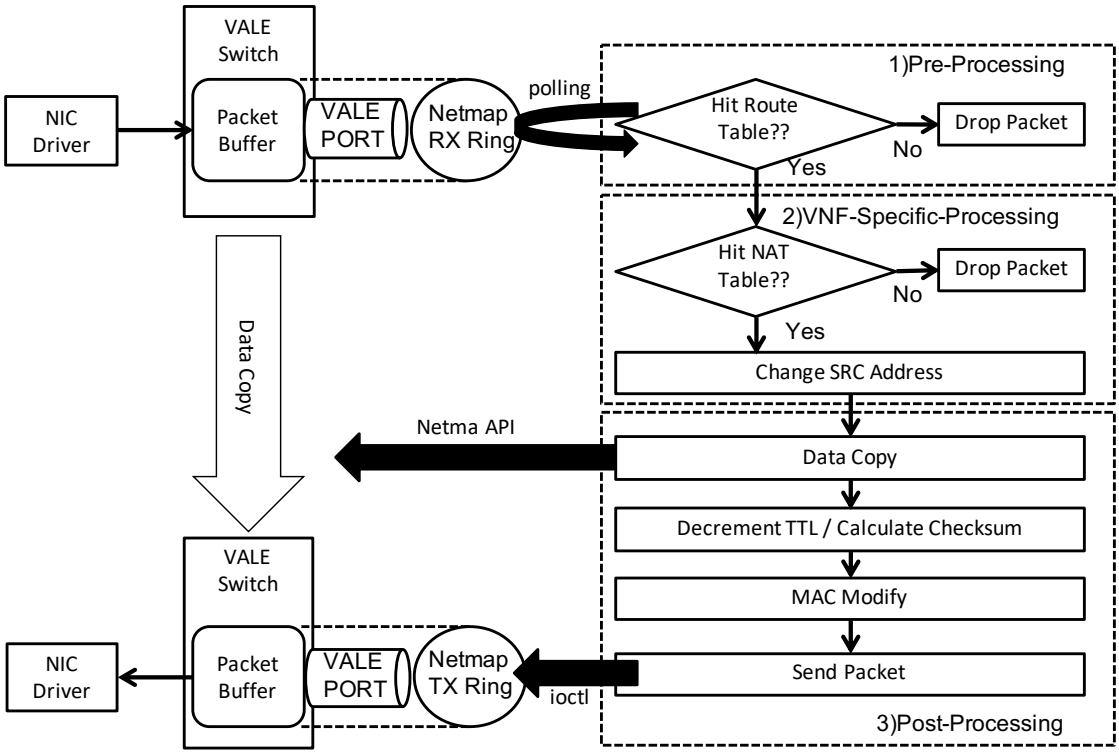


図 5.5: プロセス型 VNF における NAT の実装

5.4　評価

提案手法の評価をおこなうため，コンテナ型 NFV アーキテクチャと同等の可搬性を担保した従来方式との比較をおこなう．定量的な評価としては，VNF にルータ，NAT，Firewall を実装した場合のパケット処理性能を測定する．また，定性的な評価として，VM を利用した NFV と比較した可搬性の評価をおこなう．

5.4.1　評価環境

ハードウェアとソフトウェア

評価に利用したハードウェアは Cisco Systems 社の UCS C200 M2 である．Intel XEON E5670 2.97GHz 6 Core が 2 ソケット，DDR3 1333MHz 8GByte が 12 本搭載されており，ネットワークデバイスには 10Gbit Ethernet カードである Intel X520 DA2 を利用した．

利用した OS は Linux Kernel 3.13.0–37 である．カーネルの設定は Ubuntu14.04.1 の標準設定を利用した．10Gbit Ethernet カードのデバイスドライバには，VM 形式の場合は Linux Kernel 付属の ixgbe を，本提案の場合は Netmap 付属の ixgbe ドライバを利用した．ハイパーバイザには qemu-kvm バージョン 2.0 を利用した．

測定器には Spirent 社の SPT-2000 に 10Gbit Ethernet の測定モジュールである XFP-2001b を利用し，サーバと計測器は直接繋げた．

実験ネットワーク構成

図 5.6 に，サービス連結の性能測定に用いたネットワーク構成を示す．どちらの構成も物理ネットワークデバイスと VNF の接続，VNF と VNF 間の接続は仮想ブリッジを経由しておこなう．また，NFV の利点に仮想化による柔軟なトポロジ構成とハードウェア依存度が低いソフトウェアによる構成が挙げられる．そのため，動的なスケールアウトなどによる柔軟なネットワーク構成の変更が可能なネットワーク構成とした．

VM 形式の実験環境

VM の問題点として着目すべき点は仮想化に起因するオーバーヘッドである．すなわち VM がセンシティブ命令を実行する事によるハイパーバイザへのコンテキストスイッチが発生する問題である．そのため，VM 形式に対して他の原因で性

能劣化を起こさぬよう，この問題以外の部分のオーバーヘッドは可能な限り関連技術を用いて削減する方針をとる．

VM が利用する仮想ネットワークデバイスには vhost-net[53] を利用した．macvtap[54] の利用も検討した．しかし，サービス連結をおこなうためには，EVB(Edge Virtual Bridge)[55] で定義されているブリッジモード (VEB)，もしくは外部スイッチを経由した VEPA を利用する必要があり，どちらもハードウェア固有の機能を利用するため採用しなかった．

VM を利用した VNF の実装はゲスト OS のカーネル内に実装した物を利用する．ゲスト OS 上のプロセスとして実装した場合，仮想化のオーバーヘッドが存在しない分，本提案の方が高速なのは自明だからである．具体的に利用した機能は Linux の IP Tables を利用した Firewall 機能，NAT 機能である．

計測トラフィックパターン

計測トラフィックパターンは，64，512，1518 バイトの 3 種類の Ethernet フレームのサイズを利用した．測定器の送信側インターフェースより 10Gbps をワイヤレートのトラフィックを送信し，受信側インターフェースに通過して来たパケット数を基にスループットを算出した．

本提案で利用した Netmap 関連技術と，VM 形式で利用可能なマルチキューの性能の差異を避けるため，計測トラフィックは 1 フローとした．VM に割り当てる仮想 CPU 数に対してデータプレーン性能に優位な差が見られなかったためである．したがって，本提案手法，VM 利用のどちらも割り込みで利用される CPU コア数は 1 個である．

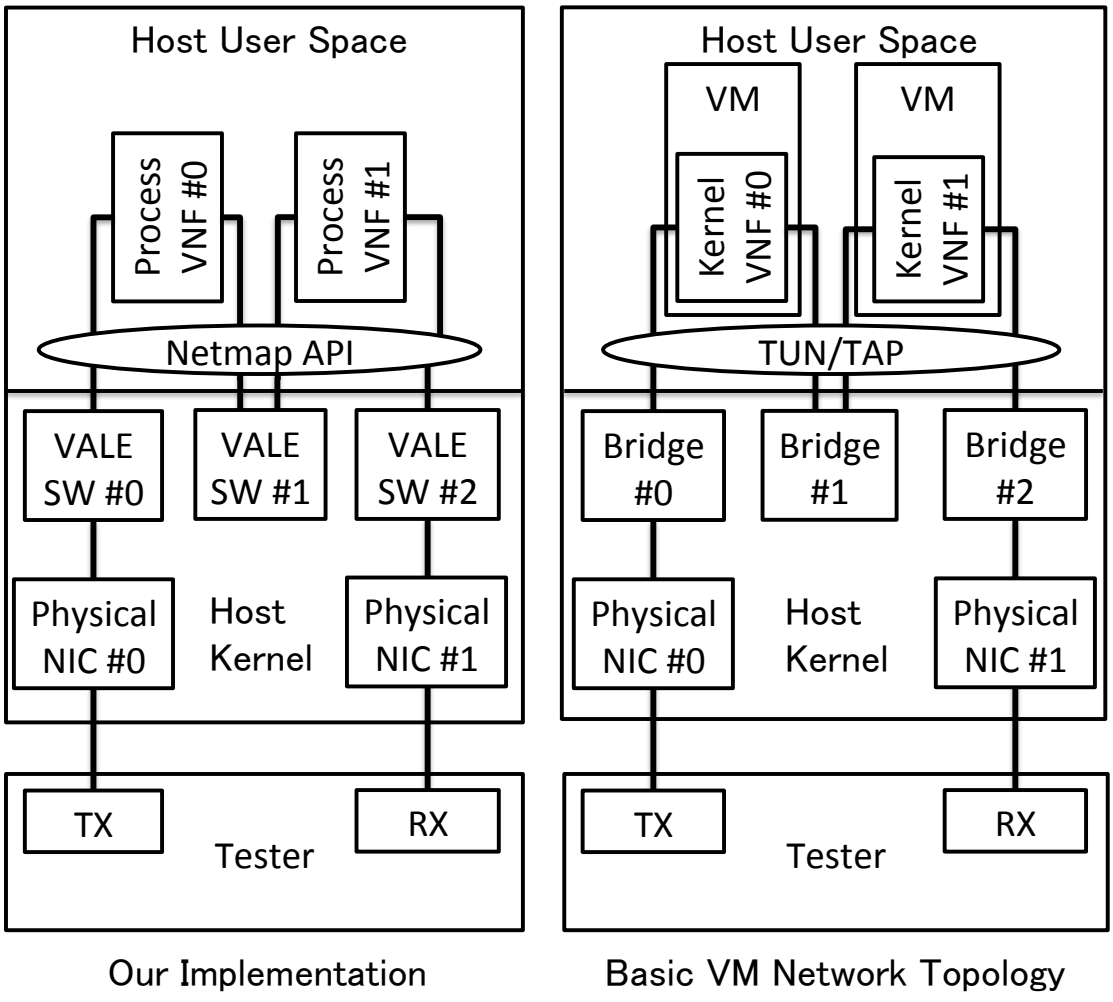


図 5.6: 評価実験ネットワーク構成

5.4.2　パケット処理性能の評価

VNF 単体の性能測定

図 5.7 は，本研究の提案手法と VM を使った VNF の 1 秒間に処理したパケット数を示している．本実験で利用した VNF の機能はルータ，NAT，Firewall の 3 種類である．同一機能に対して，約 2 から 5.5 倍の性能向上が見られた．また，小さいパケットサイズを利用した計測の方が性能向上率は高くなった．VM 形式では処理するパケット数が多い程，コンテキストスイッチの頻度が高まるためである．

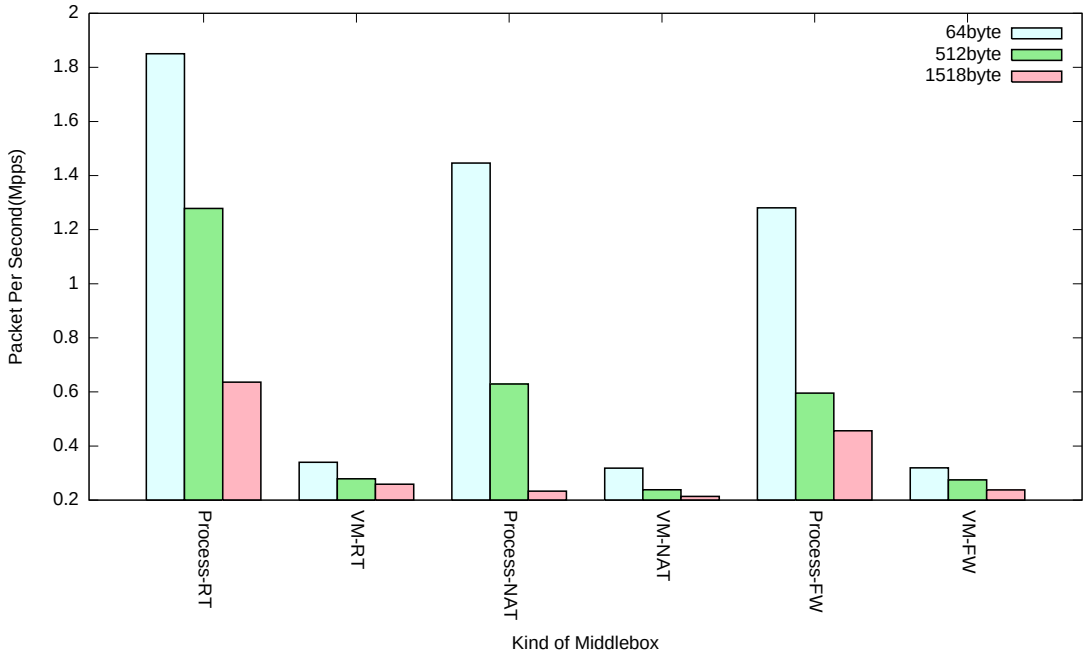


図 5.7: 提案手法と VM を使った VNF の 1 秒間に処理したパケット数

サービス連結時の性能測定

図 5.8 にサービス連結数による VNF の性能の変化を示す．利用した VNF の機能はルータで，経路表の検索，TTL の減算，チェックサムの計算，パケット転送をおこなう．同一の連結数に対して，約 3 から 9 倍の性能向上が見られた．また，ショートパケットを利用した計測の方が性能向上率は高くなった．VM 形式では処理するパケット数が多い程，コンテキストスイッチの頻度が高まるためである．

図 5.9 に VNF 単体性能を 100%とした際，連結数を増加させる事でどの程度性能が落ちていくかの割合を示す．連結数を 6 段まで増やす事により，提案手法は最大約 26%の性能劣化，VM 形式では 51%の性能劣化が観測された．性能劣化の傾向としては本提案手法の方が緩やかであることが示せた．

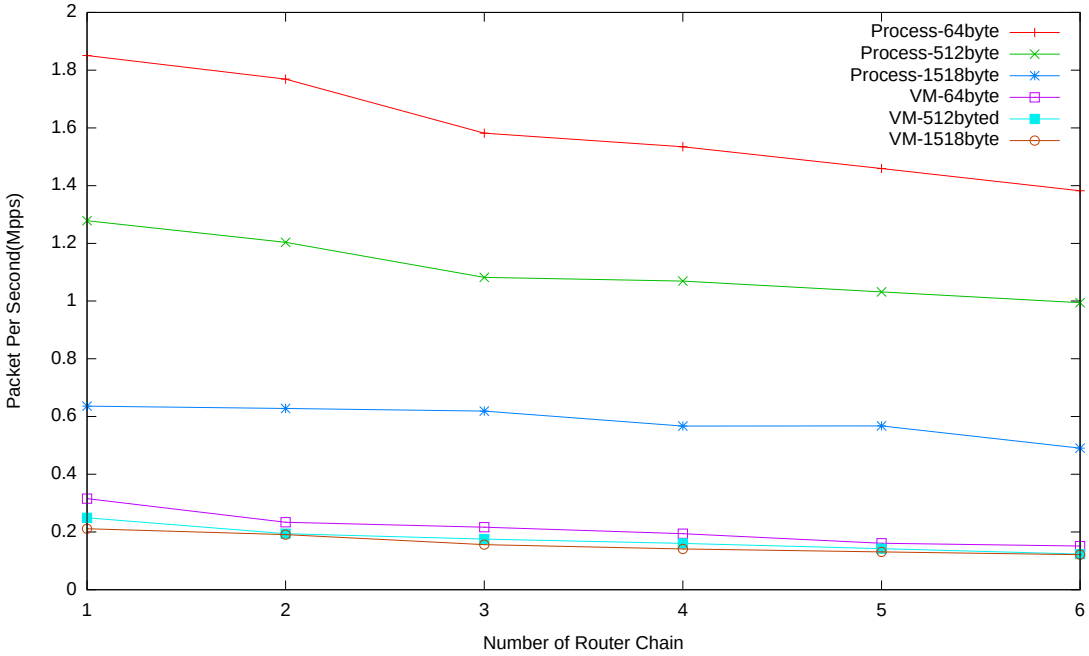


図 5.8: 連結数とフォワーディングの性能 (Mpps) の関係

5.4.3　ボトルネックの考察

提案手法と VM 形式の性能のボトルネックを CPU 利用率から考察する．先に述べた通り，計測トラフィックは 1 フローであるため原則として RSS は効果を発揮しない．そのため両者共に利用される softirqd は 1 つであり，CPU の利用率は 96%を超えて安定していた．

本提案手法ではサービス連結の 1 段目のプロセスでのみ CPU の利用率が 90%を超え，残りのプロセスでは 48%前後で安定した．softirqd がボトルネックとして処理しきれなかったパケット数を，VALE スイッチを経由して受信した 1 段目の VNF が主なボトルネックとなっているものと考えられる．

一方，VM 形式の場合は vhost-net のプロセス，qemu-kvm のプロセス共に 90%を超える CPU 利用率で安定した．そのため，カーネル内仮想ブリッジと VM との

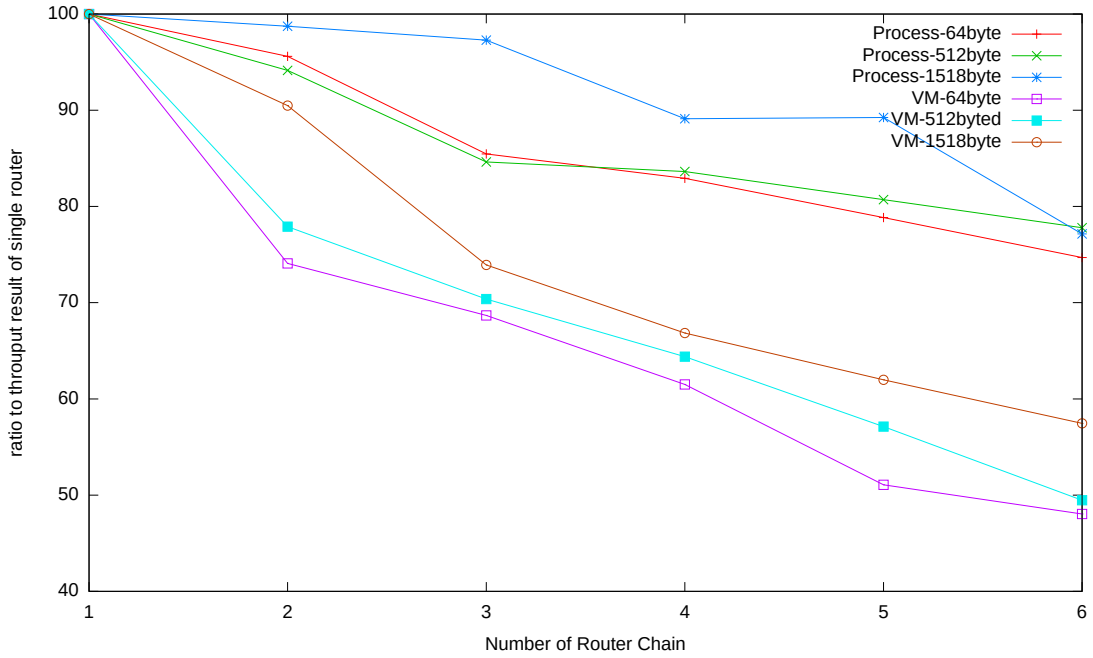


図 5.9: 連結数と IP ルーティグとフォワーディングの性能割合

間で発生するデータコピーのシステムコールがボトルネックになっていたと考えられる．

5.4.4　可搬性の評価

コンテナ型 NFV アーキテクチャの可搬性について，VNF の互換性，VNF の開発コスト，VNF のマルチテナンシーの観点から定性的な評価をおこなう．

VNF の互換性

コンテナ型 NFV アーキテクチャは，VNF をコンテナとして取り扱うことによって，VNF の実行ファイルからオペレーティングシステムを分離し，カーネルが提供する API レベルでのソースコードの互換性と，ABI レベルでのバイナリの互換性を提供する．したがって，同じ CPU アーキテクチャのプラットフォームを利用している限り，VNF は NFVI のハードウェアとオペレーティングシステムに非依存で動作する．また，提案手法の実装で採用した Netmap と VALE は特殊な NIC のハードウェアを必要とせず，実質的に FreeBSD と Linux で動作する．そのため，提案手法はハードウェアに依存しない VNF の実行環境が実現できる．

VNF の開発コスト

提案手法のようにユーザスペースの高速ネットワーク I/O 技術を利用した場合，プロトコル処理を VNF 側ですべて実装する必要がある．代表的なプロトコル処理としては，Ethernet，IP，TCP などであるが，これらの機能はライブラリ化が可能なため，時間とともに開発コストからは除外されていくと考えられる．また，ユーザスペースでのプロトコルスタック実装には，開発しやすさや，アプリケーションに最適化したプロトコル実装が可能な点など，一定の利点がある [56]．例えば，NetBSD はユーザスペースの TCP/IP スタックを動作させる機能をメインラインに取り込んでいる [57]．また，コンテナの名前空間分割機能のうち，ファイルシステムの分割機能を適用することができ，ライブラリ等を含めた VNF の実行環境を均質化することができる．

また，本研究の提案手法は，最新のオペレーティングシステムの実装とも親和性が高い．例えば，最新の Linux カーネルは，デバイスドライバ内で Berkley Packet Filter(BPF) を適用可能な eBPF と，ユーザスペースからカーネル内のパケット処理の変更を命令可能な XDP を提供している．これらは全てカーネルが標準的に提供するバイトコードであり，オペレーティングシステムや NIC ハードウェアに非依存で VNF から利用可能である．したがって，提案手法は将来的にも VNF の可搬性の確保に貢献する．

VNF の展開

提案手法は，コンテナのレジストリとオーケストレーションと連携することによって，プラットフォームごとにクロスコンパイルされたコンテナのイメージを配布することができる．近年の Kuberenetes の普及に伴って，Cloud Native Computing Foundation(CNCF) は，オーケストレータとコンテナランタイムおよびコンテナネットワーク間におけるインターフェースの標準化を進めており，Systemd，RanC，dockerd といった様々なランタイムを，単一のオーケストレータによって管理することができるようになっている．その結果，パブリッククラウドとオンプレミスのコンテナランタイムを透過的に管理できるようになり，異なる NFVI の構成における動作の保証が現実的になっている．

また，コンテナ型 NFV アーキテクチャは．従来の VM を前提とした NFV と比較して NFVI に要求するリソースのフットプリントが小さい．提案手法における VNF の最小構成単位は，オペレーティングシステムのユーザスペースで動作するプロセスであり，VNF の実行ファイルにオペレーティグシステムを含まないため，VNF の実行ファイルサイズを小さくできる．そのため，同じ機能を持つ VNF を大量に構築する場合の展開速度や，必要なネットワークの容量の点で優れている．

VNF のマルチテナンシー

同一ホスト OS で動作する VNF 間のセキュリティは，標準的なオペレーティングシステムが持つプロセスに対するアクセス権限によって担保することができる．元来プロセス間は異なる仮想メモリ空間を保持するため，VNF 間で互いのメモリ空間を破壊する事はできない．コンテナを利用して名前空間を分割する事で，VNF 間で互いの存在を認識する事ができなくなる．また，プロセスに対して実行可能なシステムコールの制限や，アクセス可能な資源をすることによって，同一のホスト OS で動作する VNF 間のセキュリティを担保することができる．

同一ホスト OS で動作する VNF 間の資源競合も，標準的なオペレーティングシステムが持つ資源割り当ての制限機能を利用することによって担保することができる．例えば Linux の cgroup を利用することによって，CPU 時間やメモリの割り当てを制限することができる．また，これらの制限は複数のサーバ間においても Mesos などに代表されるスケジューラを利用することによって管理可能である．したがって，同一サーバ内に異なるポリシーを持つ VNF 利用者を同居させる事が可能である．

5.5　本章のまとめ

本研究では VNF をホスト OS のプロセスとして実装し，一般的にシステムコールによっておこなわれるネットワーク入出力やプロトコル処理をユーザスペースで処理する事で，VM 形式よりも高速かつ軽量に実現できる VNF を設計・実装・評価した．従来の VM を利用した NFV と比較した結果，提案手法は高いパケット処理性能を有していることを示した．ネットワーク機能単体としては約 2 から 5.5 倍の性能を達成し，ネットワーク機能の連結をおこなった場合にも，最大で 9 倍の性能を達成した．また，VNF の可搬性について評価し，NFVI のハードウェアとオペレーティングシステムに非依存で動し，様々なプラットフォームに移植可能であることを示した．

第6章　結論

本章では，本研究の結論として本論文をまとめるとともに，本研究が社会に与えたインパクトについて述べ，将来に向けた展望について述べる．

6.1　本研究のまとめ

Network Function Virtualization(NFV) は，本来専用のハードウェアで実装されて来たネットワーク装置を，汎用的なハードウェアと仮想化されたネットワーク機能 (VNF) に分離する技術である．この技術は，VNF をソフトウェアにすることで，通信事業者の設備や運用にかかる費用の削減を目的としている．具体的には，共通化されたハードウェアのライフサイクルを伸ばすだけでなく，複雑化するネットワークの構築や構成変更をソフトウェアによって自動化し，多様化するネットワークのトラフィックに対応した運用を可能にする．

一方で，自動運転や遠隔医療をはじめとした高度で多様な通信品質を要求するさまざまなアプリケーションを，共通のプラットフォーム上で実現する概念であるネットワークスライシングが提案されている．ネットワークスライシングは，3GPP などの通信事業者が中心となった標準化団体などでその実装方法が議論されている．しかし，その具体的な実現方法が未定義なだけでなく，そのスコープが通信事業者の設備に限定されており，パブリッククラウドをはじめとしたアプリケーションの開発および動作環境との連携が考慮されていない．

そこで，本研究では，ネットワークスライシングが提唱する本来の目的であるアプリケーションの要求に応じたネットワークスライスの動的な構成変更を実現する手法として，ネットワークサービスの Platform as a Service(PaaS) を提唱した．PaaS は，本来クラウドコンピューティングにおいて，アプリケーションの要求に応じて動的にその構成要素を供給するプラットフォームである．この概念をネットワークにも適用することによって，アプリケーションの開発者は，自身が必要とするネットワーク機能をコードとして定義して管理・運用することが可能になる．また，その定義を宣言的に記述することによって，アプリケーションの状況に応じて，動的に構成要素の供給量や配置を最適化できるようになる．この

ようなネットワーク機能の管理を実現するには，ネットワーク機能がその動作環境の仕様に非依存で動作する「可搬性」が重要な要件であり，NFV の領域において VNF の可搬性とパケット処理性能との両立が必要であることを示した．

従来の NFV がこれらの要件を満たしているかを検証するため，産学共同でアプリケーションの要求に応じたネットワーク機能の配置を最適化できる環境について議論する次世代 Network Service Platform(NSP) コンソーシアムを立ち上げ，市販されている NFV 関連製品を検証・評価した．また，そこから得られた知見を基にして，通信事業者の顧客収容ネットワークを仮想化するユースケースである vCPE サービスを，アジア最大級の IT 技術の展示会である Interop Tokyo における出展者を収容する商用ネットワークとして実際に構築し，会期中を通して運用できることを実証した．

しかしながら，これらのコンソーシアムにおける検証・評価と，実際の商用ネットワークにおける構築・運用から，NFVI に仮想マシン (VM) を前提とした VNF の実装は，可搬性とパケット処理性能の両立に課題があることがわかった．市販されている VNF は，VM 上で動作するオペレーティングシステムとバンドルされており，NFVI の一部を包含した状態で出荷されている．VM は仮想化された周辺デバイス，特に NFV の視点では仮想 NIC の実装方法や，オペレーティングシステムにおけるデバイスドライバの実装方法の組み合わせからなる NFVI の実装方法は多様であり，VNF がそれら全てを網羅するのは困難である．また，VM が潜在的に抱えているネットワーク I/O のオーバーヘッドが大きく，これらを取り除くためにはハードウェアや VM の構成に特化したパケット処理の最適化が必要である．さらに，この最適化は VNF をネットワーク上で連結してサービスを構成する Service Function Chaining(SFC) において，ネットワーク構成やトラフィック処理の柔軟性を阻害する．したがって，VM を前提とした NFV は，可搬性とパケット処理性能の両立が困難である．

これらの課題を解決するため，本研究ではコンテナ型 NFV アーキテクチャを提案した．提案するアーキテクチャは，VNF をコンテナとして取り扱うことによって，オペレーティングシステムのカーネルから分離したプロセスの集合体として定義し，VNF と NFVI のインターフェースを POSIX API として定義することによって，VNF は NFVI のハードウェアやオペレーティングシステムの実装に非依存に動作可能になる．また，VNF と NFVI のインターフェースにおいて，オペレーティングシステムのカーネルが持つネットワークスタックをバイパスするネットワーク I/O 技術を利用することによって，可搬性を犠牲にすることなく高いパケット処理性能を実現可能になる．本研究では，コンテナ型 NFV アーキテクチャを実証するため，プロセス型 VNF を設計・実装・評価した．プロセス型 VNF は，VNF と NFVI のインターフェースに Netmap を利用し，POSIX API によってネットワー

ク I/O を実現しながら，従来の VM を利用した VNF と比較して，単体の VNF では約 2 から 5.5 倍のパケット処理性能を達成し，VNF の連結では最大で約 9 倍のパケット処理性能を達成した．したがって，コンテナ型 NFV アーキテクチャは，可搬性とパケット処理性能を両立可能なことを示した．

コンテナ型 NFV アーキテクチャは，ネットワークサービスの PaaS 化に要件を満たし，ネットワークスライシングにおけるネットワーク機能の分散配置と最適化の実現に貢献する．コンテナ型 NFV アーキテクチャは，VNF をコンテナとして取り扱うことによって，Kubernetes をはじめとするコンテナオーケストレータを適用可能にし，オンプレミス環境とパブリッククラウドを意識することなく，コンテナの最適配置，ライフサイクル管理，スケールアウト，自動回復などを実現するハイブリッドクラウドの概念を NFV に導入する．ハイブリッドクラウドの概念は，従来の NFV で大きなコストとなっていたハードウェアや VM の構成情報を含む NFVI と VNF の仕様の綿密な確認に基づく検証・構築・運用を不要にし，ソフトウェアによってこれらを自動化することが可能になる．

本論文の研究は，ネットワークスライシングの概念の実現に向けた課題を明らかにし，その課題の解決のための具体的な方法を提示した．自動運転や遠隔医療をはじめとした高度で多様な通信品質を要求するさまざまなアプリケーションが共通のプラットフォーム上で実現できる道を切り拓いた．

6.2　社会に与えるインパクト

本研究は，本研究が論文として公表された後に発展してきたネットワークスライシングを見据えた NFV に関連する研究や，ネットワーク装置ベンダにおける製品の開発に一定の影響を与えている．例えば，OpenNetVM[58] や Flurries[59] は，NetVM で培ったデータプレーンをコンテナに適用した研究である．また，コンテナにおけるネットワーク管理において，ネットワーク I/O を BPF で抽象化する手法 [60] も提案されている．ネットワーク装置ベンダにおける VNF 実装では，Juniper Networks 社が VM の代わりに Docker コンテナを利用して VNF を格納した cSRX を提供している．また，Affirmed Networks 社や ATHONET 社は，Amazon Web Service(AWS) で動作可能な 4G のモバイルコアである Evolved Packet Core(EPC) 製品を提供しており，アプリケーションを構成する要素の一部として VNF を取り扱い可能になってきている．

提案手法が実現する VNF の可搬性と拡張性は，将来的に VNF のアーキテクチャにサーバレスや Function as a Service(FaaS) の概念を導入可能にする．VNF で保持するステートの共有や永続化データを，パブリッククラウドが提供する Pub/Sub

型のメッセージングシステムやデータベースにオフロードし，VNF をサーバレス化できる．また，パケット処理をする頻度が低いアプリケーションに関しては，特定のパケットが着信したことをトリガーとして VNF をリアクティブに起動する FaaS の概念を適用できる．その結果，ネットワーク機能から動作環境を意識しないサーバレスの概念に基づいた設計が可能になり，トラフィックの需要に応じて弾力性のある VNF を構築可能になる．これらが実現すれば，これまで需要増の予測に基づいて設計されて来た通信事業者のネットワークを，実際の需要に基づく設計に変化させることができ，ネットワークの拡張性と柔軟性に関する課題を解決できる．

6.3　今後の展望

本研究では，アプリケーションに最適化された通信品質を持つネットワークを提供するネットワークスライシングを実現する手法として，ネットワークサービスの PaaS 化を提唱した．コンテナ型 NFV アーキテクチャは，ネットワークサービスの PaaS 化の実現に向けて，VNF の可搬性とパケット処理性能が両立可能なことを実証したが，クラウドコンピューティングや SDN の技術領域において課題が残っている．例えば，通信事業者とクラウド事業者のインフラストラクチャを横断的に取り扱うコンテナのオーケストレーションシステム，任意の順序でコンテナ間のトラフィックを制御するサービスチェイニング，ネットワーク機能とそのトラフィック制御を含む宣言型のアプリケーション構成管理方法など，様々な課題が残されている．今後は，ネットワークスライシングの概念の実現に向け，これらの課題を解決していく．

謝辞

2001 年に研究室に所属してから長きにわたりご指導いただき，本研究にも多くのご助言と励ましをいただきました慶應義塾大学環境情報学部 村井純教授に心より感謝いたします．また，研究室に入るきっかけとなる講義をご担当いただいてから今日まで，一人の研究者として対等な目線でご指導をいただきました慶應義塾大学環境情報学部 中村修教授に深く感謝いたします．また，本論文の執筆において細部までご指導いただいた慶應義塾大学環境情報学部 楠本博之教授に感謝いたします．そして，本研究を実施するにあたり，ネットワークの仮想化について超一流の研究環境を整えていただいた東京大学情報基盤センターネットワーク研究部門 関谷勇司准教授に深く感謝いたします．主査および副査の先生方には，学部生の頃から大学の内外で多くのチャンスをいただきました．純粋に技術を突き詰めるだけでなく，社会を動かしていくために何をしなくてはいけないか，自分に貢献できるのは何か，常にその道しるべを示していただきました．だからこそ，諦めることなく博士論文を書き上げることができました．心より尊敬と感謝の気持ちをお伝えします．

本研究を進めるにあたり，システムの設計・開発や論文の構成に関して尽力いただいた中村遼博士，上野幸杜氏に感謝いたします．自分をはるかに超える技術力を携えた後輩に助けられたのは誇らしいことでした．また，論文の執筆で苦しんだ時期に，自らの経験に基づき丁寧にご指導いただいた慶應義塾大学 大学院政策・メディア研究科 鈴木茂哉特任教授，田崎創博士に感謝いたします．査読者の視点に基づいて論文指導をいただいた慶應義塾大学環境情報学部 植原啓介教授，慶應義塾大学環境情報学部 ロドニーバンミータ教授に感謝いたします．また，本研究を始めるきっかけとして，NFV と言う新たな研究領域へと誘い，コンソーシアムの立ち上げを後押しをしていただいた湧川隆次博士に感謝いたします．成果を出すために，どんなに苦しくてもついていくと決めたことは正しかったと実感しています．また，ソフトバンク株式会社に入社後，あらゆる点で学位取得の活動を支えていただいた小林丈記氏，西和人氏，松嶋聡氏に感謝いたします．

研究者として歩み始めるにあたり，この研究室で自分のキャリアを築いていきたいと思った STREAM KG に暖かく迎え入れていただいた先輩方に感謝いたし

ます．コンピュータサイエンスにおいて，少しでも効率よく資源を使い切るこだわりを体現していただいた慶應義塾大学大学院メディアデザイン研究科 杉浦一徳教授に感謝いたします．公私を問わず常に厳しい目で叱咤激励いただいた重近範行博士に感謝いたします．DVTS と言うインターネットが切り開く未来のアプリケーションがこの手で作れることを教えてくれた小川晃通博士に感謝します．コンピュータの使い方を根本から教育し直していただいた田原裕市郎氏に感謝いたします．コンピュータを使って人にサービスを提供する意義と責任を教えていただいた小川浩司氏に感謝いたします．プログラミングとコンピュータの仕組みを丁寧に教えていただいた入野仁氏，岡田耕司氏に感謝いたします．研究室に入った時から共に苦しい時間を戦い抜いた久松剛博士に感謝いたします．そして，何よりも STREAM KG に決めたきっかけを作っていただいた海崎良氏に感謝いたします．また，学部時代から多くのプロジェクトへ共に参加し，大きな舞台へと誘ってくれた工藤紀篤博士に感謝いたします．自分が手がけたものが，人の役に立ち，社会を動かす力として貢献することを実感できたのは，何よりも大きな励みになりました．特に最初にチャンスをいただいた SOI プロジェクトの皆様には感謝いたします．まず遠隔教育と言う領域を切り開いてこられた慶應義塾大学大学院メディアデザイン研究科 大川恵子教授に感謝いたします．また，新参者の自分に信頼を寄せていただいた村上陽子博士に感謝いたします．同じ学年として様々なアプリケーション開発に一緒にチャレンジした白畑真氏，東京農工大学 総合情報メディアセンター 三島和宏准教授に感謝いたします．

WIDE Project，JGN，Interop NOC の諸氏には，プロとしてネットワークを設計・構築・運用するための技術と心がけを学ばせていただきました．本研究の原点は，この活動の中で生まれたネットワークの仮想化です．共にこのテーマについて議論を進めた奈良先端科学技術大学院 情報科学研究科 門林雄基教授，小林和馬博士，宇田仁博士，長谷部克幸氏，三宅喬博士，宍倉弘祐氏，橋本賢一郎氏，渡邊貴之氏，齋藤修一氏，棚橋弘幸氏，石原知洋博士に感謝いたします．

慶應義塾大学村井研究室の諸氏には，長い研究室の生活を共に過ごしただけでなく，全ての活動が刺激的かつ濃密な時間でした．ここで過ごした時間全てが財産であり，今の自分を作り上げています．ここに改めて感謝をいたします．また，長期にわたり研究活動を陰ながら支えていただいた，村井研究室秘書 渡辺康恵氏，宇佐美由美子氏に感謝いたします．そして，本論文を執筆するにあたり，先生方とのスケジュール調整，学事申請に掛かる書類の整理，本論文の印刷に至るまでの手配，そして何よりも論文執筆を継続していく意欲を支えてくれた渋谷雪絵氏に感謝します．

最後に長期にわたる研究生活を忍耐強く支えてくれた，父 堀場勝英 母 堀場政子に感謝の意を捧げます．

参考文献

[1] Yaakov (J) Stein. Sdn and nfv what' s it all about? https://slideplayer.com/slide/4476649/, Oct. 2014.

[2] J. Halpern and C. Pignataro. Service function chaining (sfc) architecture, October 2015. RFC7665.

[3] Jeremias Blendin, Julius Rückert, Nicolai Leymann, Georg Schyguda, and David Hausheer. Position paper: Software-defined network service chaining. In *EWSDN Workshop*, 2014.

[4] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu. Research directions in network service chaining. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pp. 1–7, Nov 2013.

[5] ETSI NFV ISG. Network Functions Virtualisation - Update White Paper. http://portal.etsi.org/nfv/nfv_white_paper2.pdf.

[6] NFV White Paper. Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1. October 2012.

[7] Myung-Ki Shin, Ki-Hyuk Nam, and Hyoung-Jun Kim. Software-defined networking (sdn): A reference architecture and open apis. In *2012 International Conference on ICT Convergence (ICTC)*, pp. 360–361. IEEE, 2012.

[8] ETSI NFV ISG. Service Chaining for NW Function Selection in Carrier Networks. http://nfvwiki.etsi.org/images/NFVPER(14)000004r2_NFV_ISG_PoC_Proposal_Service_Chaining_for_NW_Function_Select.pdf.

[9] Patrick Kwadwo Agyapong, Mikio Iwamura, Dirk Staehle, Wolfgang Kiess, and Anass Benjebbour. Design considerations for a 5g network architecture. *IEEE Communications Magazine*, Vol. 52, No. 11, pp. 65–75, 2014.

[10] Akhil Gupta and Rakesh Kumar Jha. A survey of 5g network: Architecture and emerging technologies. *IEEE access*, Vol. 3, pp. 1206–1232, 2015.

[11] Mehmet Ersue. Etsi nfv management and orchestration-an overview. In *Proc. of 88th IETF meeting*, 2013.

[12] Evangelos Haleplidis, Damascene Joachimpillai, Jamal Hadi Salim, Diego Lopez, Jason Martin, Kostas Pentikousis, Spyros Denazis, and Odysseas Koufopavlou. Forces applicability to sdn-enhanced nfv. In *2014 Third European Workshop on Software Defined Networks*, pp. 43–48. IEEE, 2014.

[13] Interop Tokyo. http://www.interop.jp.

[14] ShowNet. http://www.interop.jp/2015/shownet.

[15] ETSI NFV ISG. Network functions virtualization use cases. http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf.

[16] Akihiro Nakao, Ping Du, Yoshiaki Kiriha, Fabrizio Granelli, Anteneh Atumo Gebremariam, Tarik Taleb, and Miloud Bagaa. End-to-end network slicing for 5g mobile networks. *Journal of Information Processing*, Vol. 25, pp. 153–163, 2017.

[17] M Series. Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond. *Recommendation ITU*, pp. 2083–0, 2015.

[18] Jose Ordonez-Lucena, Pablo Ameigeiras, Diego Lopez, Juan J. Ramos-Muñoz, Javier Lorca, and Jesús Folgueira. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *IEEE Communications Magazine*, Vol. 55, pp. 80–87, 2017.

[19] Xenofon Foukas, Georgios Patounas, Ahmed Elmokashfi, and Mahesh K Marina. Network slicing in 5g: Survey and challenges. *IEEE Communications Magazine*, Vol. 55, No. 5, pp. 94–100, 2017.

[20] 3GPP. Management and orchestration; Concepts, use cases and requirements. Technical Specification (TS) 28.530, 3rd Generation Partnership Project (3GPP), 12 2018. Version 15.1.0.

[21] 3GPP. System architecture for the 5G System (5GS). Technical Specification (TS) 23.501, 3rd Generation Partnership Project (3GPP), 6 2019. Version 16.1.0.

[22] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet computing*, Vol. 13, No. 5, pp. 14–22, 2009.

[23] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *2009 Fifth International Joint Conference on INC, IMS and IDC*, pp. 44–51. Ieee, 2009.

[24] ETSI NFV ISG. Network functions virtualization(nfv) architectural framework. http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf.

[25] Tobias Binz, Uwe Breitenbücher, Florian Haupt, Oliver Kopp, Frank Leymann, Alexander Nowak, and Sebastian Wagner. Opentosca–a runtime for tosca-based cloud applications. In *International Conference on Service-Oriented Computing*, pp. 692–695. Springer, 2013.

[26] Yuan-Mao Hung, Shih-Che Chien, and Yung-Yi Hsu Chunghwa. Orchestration of nfv virtual applications based on tosca data models. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 219–222. IEEE, 2017.

[27] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, Vol. 55, No. 3, 2012.

[28] Beyond Softnet. J. salim, r. olsson et al. In *Proc. 5th Ann. Linux Showcase and Conf., Oakland, California*, 2001.

[29] Wenji Wu, P. DeMar, and M. Crawford. A transport-friendly nic for multicore/multiprocessor systems. *Parallel and Distributed Systems, IEEE Transactions on*, Vol. 23, No. 4, pp. 607–615, April 2012.

[30] Tom Herbert. Rfs: Receive flow steering. http://lwn.net/Articles/381955/, accessed Oct.26 2014.

[31] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 15–28. ACM, 2009.

[32] Kevin Fall, Gianluca Iannaccone, Maziar Manesh, Sylvia Ratnasamy, Katerina Argyraki, Mihai Dobrescu, and Norbert Egi. Routebricks: enabling general purpose network infrastructure. *ACM SIGOPS Operating Systems Review*, Vol. 45, No. 1, pp. 112–125, 2011.

[33] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. Packetshader: a gpu-accelerated software router. *ACM SIGCOMM Computer Communication Review*, Vol. 41, No. 4, pp. 195–206, 2011.

[34] Intel Corp. Intel dpdk: Data plane development kit. http://dpdk.org/., accessed Oct.26 2014.

[35] Luigi Rizzo. Netmap: A novel framework for fast packet i/o. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pp. 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[36] Luigi Rizzo and Matteo Landi. Netmap: Memory mapped access to network devices. *SIGCOMM Comput. Commun. Rev.*, Vol. 41, No. 4, pp. 422–423, August 2011.

[37] Ilias Marinos, Robert NM Watson, and Mark Handley. Network stack specialization for performance. In *ACM SIGCOMM Computer Communication Review*, Vol. 44, pp. 175–186. ACM, 2014.

[38] PCI-SIG. SR-IOV Primer: An Introduction to SR-IOV Technology. http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html, accessed February 2nd 2015.

[39] Abel Gordon, Nadav Amit, Nadav Har'El, Muli Ben-Yehuda, Alex Landau, Assaf Schuster, and Dan Tsafrir. Eli: bare-metal performance for i/o virtualization. *ACM SIGPLAN Notices*, Vol. 47, No. 4, pp. 411–422, 2012.

[40] Nicira Networks. Open vSwitch: An open virtual switch. http://openvswitch.org/, accessed Jan.28 2015.

[41] Intel Open Source Technology Center. Intel dpdk vswitch. https://01.org/packet-processing/intel-ovdk., accessed Oct.26 2014.

[42] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. Netvm: High performance and flexible networking using virtualization on commodity platforms. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pp. 445–458, Berkeley, CA, USA, 2014. USENIX Association.

[43] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pp. 459–473, 2014.

[44] 中村遼, 堀場勝広, 関谷勇司. SDN を用いたクラウドサービスネットワークの実現 (インターネット運用・管理, 一般). 電子情報通信学会技術研究報告. IA, インターネットアーキテクチャ, Vol. 113, No. 200, pp. 5–10, 2013.

[45] 中村遼. Interop tokyo 2014 shownet における sdn/nfv. http://www.sdnjapan.org/document_2014/30_session5_Nakamura.pdf, Oct. 2014.

[46] ETSI NFV ISG. Network functions virtualization(nfv) virtualisation requirements. http://www.etsi.org/deliver/etsi_gs/NFV/001_099/004/01.01.01_60/gs_NFV004v010101p.pdf.

[47] Rusty Russell. Virtio: Towards a de-facto standard for virtual i/o devices. *SIGOPS Oper. Syst. Rev.*, Vol. 42, No. 5, pp. 95–103, July 2008.

[48] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Enabling fast failure recovery in openflow networks. In *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*, pp. 164–171. IEEE, 2011.

[49] Klaithem Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi, and Jameela Al-Jaroodi. A survey of load balancing in cloud computing: Challenges and algorithms. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pp. 137–142. IEEE, 2012.

[50] P. Phaal, S. Panchen, and N. McKee. Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks, September 2001. RFC3176.

[51] Salvatore Renato Ziri, Ahmad Tajuddin Samsudin, and Christophe Fontaine. Service chaining implementation in network function virtualization with software defined networking. In *Proceedings of the 5th International Conference on Communications and Broadband Networking*, pp. 70–75. ACM, 2017.

[52] Luigi Rizzo and Giuseppe Lettieri. Vale, a switched ethernet for virtual machines. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pp. 61–72, New York, NY, USA, 2012. ACM.

[53] Abel Gordon, Nadav Har'El, Alex Landau, Muli Ben-Yehuda, and A-bombhay Traeger. Towards exitless and efficient paravirtual i/o. In *Proceedings of the 5th Annual International Systems and Storage Conference*, p. 10. ACM, 2012.

[54] Vivek Kashyap, Arnd Bergman, Stefan Berger, Gerhard Stenzel, and Jens Osterkamp. Automating virtual machine network profiles. In *Linux Symposium*, p. 147. Citeseer, 2010.

[55] IEEE 802 LAN/MAN Standards Committee. Ieee 802.1qbg - edge virtual bridging. http://www.ieee802.org/1/pages/802.1bg.html., accessed Oct.27 2014.

[56] Ilias Marinos, Robert N. M. Watson, and Mark Handley. Network stack specialization for performance. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, HotNets-XII, pp. 9:1–9:7, New York, NY, USA, 2013. ACM.

[57] Antti Kantee. Environmental independence: Bsd kernel tcp/ip in userspace. *Proc. of AsiaBSDCon*, pp. 71–80, 2009.

[58] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, KK Ramakrishnan, and Timothy Wood. Opennetvm: A platform for high performance network service chains. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pp. 26–31. ACM, 2016.

[59] Wei Zhang, Jinho Hwang, Shriram Rajagopalan, KK Ramakrishnan, and Timothy Wood. Flurries: Countless fine-grained nfs for flexible per-flow customization. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pp. 3–17. ACM, 2016.

[60] Ben de Graaf. Segment routing in container networks. Master thesis, Universiteit Van Amsterdam System and Network Engineering, 2017.

[61] Open source processor emulator. http://www.qemu.org/., accessed June. 5th 2019.

[62] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks, August 2014. RFC7348.

[63] S. Bradner and J. McQuaid. Benchmarking methodology for network interconnect devices, March 1999. RFC2544.

[64] Red Hat Inc. Red hat community service for users of openstack on red hat based platforms. http://www.rdoproject.org/., accessed June. 5th 2019.

[65] The virtualization api. http://libvirt.org/., accessed June. 5th 2019.

[66] Linux Foundation. Open platform for nfv(opnfv). http://www.opnfv.org/., accessed June. 5th 2019.

付 録A　次世代NSPコンソーシアムにおけるNFVの評価

A.1　コンソーシアムの活動方針

次世代 NSP コンソーシアムでは，ネットワーク・インフラに求められる新たな要求に応えることのできるネットワークサービスのアーキテクチャを提案し，実証実験を通じてその実現と普及を目指す．本コンソーシアムで提案する新たなネットワークサービスアーキテクチャでは，Software Defined Networking (SDN) と Network Functions Virtualization (NFV) を重要な要素技術として捉え，これらの技術の利点を生かすことで，動的な変化に対応することを可能とする．

ネットワークトラフィックの変化や SDN，NFV といった新技術の登場により，ネットワークインフラは，より柔軟かつ動的な構成変更ができるアーキテクチャに移行することが求められる．そこで本コンソーシアムでは，これらネットワークの変革に対応することのできる，新たなネットワークサービスアーキテクチャを提案する．ネットワークサービスとは，ユーザの利便性のためにネットワークに付随して提供される機能を意味し，ユーザを攻撃や成りすましから守ったり，ユーザがコンテンツを容易に得ることを支援したり，ユーザがどこにいても同じサービスを受けられることを担保する技術を意味する．

これらの目標を達成したネットワークインフラを実現するためには，SDN と NFV が重要な要素技術となると考えられる．特に本コンソーシアムでは，要素技術の中で大きな割合を占めると思われる Network Function Virtualization (NFV) に着目し，その基礎技術の検証を通じて，技術基盤を確かなものにすることを目指す．

本コンソーシアムは，次世代のネットワークサービスプラットフォームに求められる要件を明確にし，そのアーキテクチャの設計と構築を目的とした活動をおこなう．しかし，単に研究においてアーキテクチャを構築するのみならず，構築したネットワークサービスプラットフォームが，実サービスに展開されネットワークインフラとして社会に貢献できるものとなるよう，常に導入と運用の観点を忘れずに活動を展開する必要がある．この観点を持たなければ，真に導入可能とな

る次世代ネットワークサービスアーキテクチャを構築することはできず，単に研究のためのアーキテクチャとなってしまうためである．

そこで本コンソーシアムでは，次の三点を常に念頭においた研究活動をおこなう．

1. ハードウェアとソフトウェアの分離による汎用ハードウェアの有効利用とマルチベンダ相互接続性の検証
2. 性能・品質の評価指標の策定とデータの公開
3. 実ネットワークにおける運用と検証を通じた，構築手法や計測手法の公開

これらを念頭に置くことにより，研究に特化された現実とかけ離れたアーキテクチャではなく，導入と運用が可能なアーキテクチャの構築を目指す．

本コンソーシアムは，大学や官民を問わない研究機関を中心としたアカデミック組織からのメンバー，実際の製品を生み出したりその製品を用いてサービスを実現するベンダやインテグレータ，さらにサービスを顧客に提供する通信事業者という三者が協力体制を構築することにより，研究として新たなネットワークアーキテクチャを生み出すだけではなく，実用のサービスに取り入れ，運用することのできるアーキテクチャの構築を目指す．上記の方針に賛同し，本コンソーシアムに参加した企業の一覧を下記に示す．

- **幹事会員 (50 音順)**
  - アラクサラネットワークス株式会社
  - ソフトバンク株式会社
  - デル株式会社
- **賛助会員 (50 音順)**
  - IP Infusion Inc.
  - EMC ジャパン株式会社
  - イクシアコミュニケーションズ株式会社
  - 伊藤忠テクノソリューションズ株式会社
  - 株式会社インターネットイニシアティブ
  - インテル株式会社
  - ヴイエムウェア株式会社
  - エアロフレックスジャパン株式会社

- NTT コミュニケーションズ株式会社
- 有限会社銀座堂
- KCCS モバイルエンジニアリング株式会社
- KDDI 株式会社
- シスコシステムズ合同会社
- ジュニパーネットワークス株式会社
- Digilent Inc.
- 東京エレクトロン デバイス株式会社
- 株式会社東陽テクニカ @Benchmark
- 日商エレクトロニクス株式会社
- 日本アルカテル・ルーセント株式会社
- 日本電気株式会社
- 日本ヒューレット・パッカード株式会社
- 日本ラドウェア株式会社
- ネットワンシステムズ株式会社
- Virtual Open Systems SAS
- パロアルトネットワークス合同会社
- 華為技術日本株式会社 (ファーウェイ･ジャパン)
- フォーティネットジャパン株式会社
- 富士通株式会社
- 株式会社ブロードバンドタワー
- 株式会社マクニカ
- メラノックステクノロジーズジャパン株式会社

A.2　NFV の検証

この活動方針に基づき，まず NFV の評価をおこなうハードウェア・ソフトウェアのプラットフォームを構築し，様々なハードウェア，ソフトウェアの動作検証をおこなった．検証項目，計測手法に関してはコンソーシアム参加組織にて議論して

決定した．さらに，NFV の実運用に向けて，複数のネットワークベンダの COTS (Commercial off the shelf) と VNF (Virtual Network Function) を組み合わせてサービスを構成するため，各ベンダの製品を用いて接続試験をおこない，相互接続性の検証をおこなった．

A.3　検証環境

本コンソーシアムでは，検討の結果，NFV の基盤となる NFVI 環境を，表 A.1 に示す 5 種類としてパターン化した．この NFVI パターンは，本コンソーシアムが定義する最も基本的な NFVI 構成であり，今後の検証もこのパターン定義を基本としておこなっていく．また，このパターン定義は現時点での最新のものであり，利用している技術の進化や，新たな技術の登場によっても更新されていく可能性がある．

NFVI-1 は，ハイパーバイザに VMware ESXi 5.5 を採用し，VMware 内蔵仮想スイッチと，準仮想化 NIC である VMXNET3 を利用した環境となる．

NFVI-2 から 4 については，ハイパーバイザに QEMU KVM[61] を利用し，Open vSwitch を仮想スイッチとして利用して，物理 NIC と仮想化 NIC を接続している．その際，OVS(Open vSwitch) のデータプレーンに Intel DPDK[34] を利用する場合を NFVI-2，Linux カーネルを利用する場合を NFVI-3，そして VXLAN[62] のハードウェアオフローディングを利用する場合を，NFVI-4 として定義した．

なお，NFVI-1 から 4 は，VNF として動作するゲスト OS が，特殊なネットワークドライバに対応する必要がなく，ほぼ全ての VNF が動作可能となる NFVI パターンである．

最後に，NFVI-5 では SR-IOV[38] を利用し，物理 NIC を仮想化された PCI デバイスに分割し，PCI パススルーによって，VNF に PCI デバイスを直結させている．この場合，VNF として動作するゲスト OS に，SR-IOV によって仮想化された PCI デバイスを認識するための，特殊なデバイスドライバが必要となる．そのため，一部の VNF のみが対応しており，また SR-IOV をサポートしている物理 NIC も限られている．

表 A.1: 検証 NFVI パターン一覧

	NFVI-1	NFVI-2	NFVI-3	NFVI-4	NFVI-5
サーバ	Cisco UCS 200 M2	DELL R620	DELL R620	DELL R620	DELL R620
CPU	Xeon E5670 2.97GHz	E5-2600 v2 2.6GHz	E5-2600 v2 2.6GHz	E5-2600 v2 2.6GHz	E5-2600 v2 2.6GHz
メモリ	DDR3 1333MHz 96GB	DDR3 1333MHz 32GB	DDR3 1333MHz 32GB	DDR3 1333MHz 32GB	DDR3 1333MHz 32GB
物理NIC	Intel X520 82599ES	Intel X520 82599ES	Intel X520 82599ES	Mellanox Connect-X3 Pro	Intel X520 82599ES
ホストOS	VMware ESXi 5.5	Ubuntu 14.04	Ubuntu 14.04	Ubuntu 14.04	Ubuntu 14.04
ハイパーバイザ	VMware ESXi 5.5	QEMU KVM 2.2.1	Qemu KVM 2.0	Qemu KVM 2.0	Qemu KVM 2.0
仮想NIC	VMXNET3	virtio-net + vhost-net	virtio-net + vhost-net	virtio-net + vhost-net	SR-IOV
仮想スイッチ	VMware付属vSwitch	OVS 2.3.1 + DPDK 1.8.0 Dataplane DPDK	OVS 2.0 Dataplane Kernel Module	OVS 2.0 VXLAN Offloading	N/A

A.4　計測手法とシナリオ

本節では，本コンソーシアムが定義し推奨する，NFV に適した計測手法の定義について述べる．この計測手法は，第一期検証での計測を通じた経験から得られたものであり，同時に，専用機器に対する既存の計測手法が，そのまま NFV システムへの計測に適用することが困難であることを意味する．

これは，NFV を用いたシステムは，当然その多くの部分がソフトウェアを用いて構成されているため，既存の OS のスケジューラを利用している限り，定常的に安定した性能を出すことが難しいことに起因する．現在の OS では，ネットワーク I/O の処理は割り込みベースもしくはポーリングベースであり，そのタイミングは厳格なものではない．そのため，同様の負荷をかけたとしても同様の性能が得られるとは限らず，専用ハードウェアをもちいた既存機器の計測手法をそのまま適用すると，極端に低い性能値を示す場合が多い．

そこで，本コンソーシアムでは，暫定的な解決手法として，次の 2 通りの計測手法を用い，検証をおこなうこととした．これは根本的な解決策ではないため，計測手法に関しては，さらなる検討が必要となる．

A.5　性能試験手法

第一期検証をおこなうにあたって，本コンソーシアムでは，2 種類の計測手法を定義した．これは，既存の専用ハードウェア製品における性能測定手法がそのまま適用できなかったためである．

性能試験では，「スループット」と「転送レート」の 2 種類の測定方法を定義した．それぞれの定義を下記に示す．前述の通り，現在の OS と仮想化の仕組みで

は，資源競合が発生した場合，パケットロスが発生しやすく，スループットが低下する傾向にある．

- スループット
  RFC2544[63] に準拠したスループット試験．対象機器に入力されたフレームをロスなく転送できる最大レートを探索する試験方法である．対象機器に対して入力されるフレームレートをロスなく転送できた場合は，フレームレートを上げ，ロスが発生した場合はフレームレートを下げて試験をおこなう．これを繰り返し，バイナリ探索のアルゴリズムによって，ロスのない最大レートを検索する手法である．

- 転送レート
  物理 NIC の最大転送フレームレート (ワイヤーレート) を対象機器に対して入力し，パケット転送処理ができたフレームレートを測定する試験．すなわち，10Gbps のインタフェースを有したハイパーバイザにて VNF が動作している場合には，10Gbps のレートにてトラフィックを入力し，どの程度のトラフィックが出力されたかによって性能を計測する手法である．

A.6　性能検証シナリオ

第一期検証では，ハードウェアテスター製品を用いて性能測定をおこなった．その際，テスターに与えた計測パラメータを表 A.2 に示す．このパラメータは，スループットと転送レート両方の計測手法において利用した．

表 A.2: L2-L3 性能試験の詳細

	スループット	転送レート
通信の方向	双方向	
測定時間	10 秒	
フロー数	1 フロー，100 フロー	
フレーム長	64 バイト，512 バイト，1518 バイト	
フレームレート	開始 10%，最小 1%，最大 100% 許容ロスレート 0%	100%
測定内容	平均転送フレーム数/秒，遅延時間 (最大，最小，平均)	

A.7　検証結果

本章では，第一期検証にておこなった検証の結果をまとめる．第一期検証では，主に下記の 3 点を評価項目とした検証をおこなった．

1. VNF の NFVI 環境への依存性
   VM として動作する VNF が，NFVI 構成にどの程度依存しているか，どれだけ多くの NFVI パターンにて動作できるかを評価する．

2. VNF の自動デプロイに向けた運用性
   オーケストレーターを介して動的かつ容易に VNF の追加と削除ができるかを評価する．具体的には，VNF のインストール方法，ライセンス認証やデプロイ方法，運用上のインターフェースが統一されているか等を評価する．

3. NFVI 構成とパケット転送性能
   同一の VNF を，異なる NFVI 構成で動作させた場合のパケット転送性能を評価する．また，NFVI 構成と VNF の設計 (アーキテクチャ) におけるパケット転送性能の違いを考察する．

A.8　VNF の NFVI 環境への依存性

VNF と NFVI 環境との依存関係とは，VNF が，動作を保証する NFVI 環境をどの程度限定しているかである．NFV の本質はハードウェアとソフトウェアの分離であり，仮想化を用いたハードウェアの抽象化である．しかし，実際にはそれぞれの VNF が動作するハードウェアやソフトウェア環境が限定される場合がある．第一期検証中に観測された，VNF の NFVI への依存関係について，下記にまとめる．

A.8.1　NFVI のソフトウェアバージョンへの依存性

VNF が特定の NFVI ソフトウェアバージョンを要求する場合が確認された．具体的にはハイパーバイザ，仮想化環境の操作 API ライブラリ (Libvirtd)，VNF デプロイソフトウェア (Virt-Manager) のバージョンである．

最も影響が大きい依存関係は，ハイパーバイザである．少数ではあるが，ハイパーバイザのバージョンによって，正常に動作しない VNF が存在した．これは，ハイパーバイザのある特定のバージョンが提供している機能を利用しているため

であり，このような VNF は非常に環境依存性が高く，他の VNF との共存に影響を与えると考えられる．

A.8.2　VNF のアーキテクチャと仮想スイッチ

ある VNF では，仮想 NIC に割り当てられた MAC アドレスと，VNF であるゲスト OS が管理する NIC の MAC アドレスが異なる場合があった．このような VNF 実装は，1 つの仮想 NIC が複数の機能を有する，もしくはトラフィックを透過させるような機能が必要となる VNF に見うけられる．

VMware の仮想スイッチは，VM の仮想 NIC に割り当てた MAC アドレスを記憶しており，通常は宛先 MAC アドレスが VM の仮想 NIC と異なるパケットを転送しない．しかし，VNF のアーキテクチャによってはこれらのパケットの転送を許可する必要がある．そのため，仮想スイッチ側のセキュリティレベルを変更する，もしくは VNF の仮想 NIC の MAC アドレスをハイパーバイザが割り当てた仮想 NIC の MAC アドレスと一致させるなどの処置が必要になる．しかし，これらの設定の粒度は仮想スイッチ単位となっており，ある VNF がこの機能を要求するからといって仮想スイッチ全体でセキュリティレベルを変更すると，NFV システム全体としてのセキュリティレベル低下や，他の VNF の想定しない動作を招く可能性がある．

A.8.3　SR-IOV の L2 Classifier

SR-IOV は，VF によって物理 NIC を複数の仮想 PCI デバイスに分割する．SR-IOV 機能を持つ物理 NIC は，その内部に L2 Classifier と呼ばれる L2 スイッチに似た機能が実装されている．PF に入力されたパケットは，MAC アドレスや VLAN タグに基づいて対応する VF にパケットを振り分けられる．

しかし現段階では，多くの Linux ディストリビューションにおいて，L2 Classifier に対する転送ルールを追記するためのインターフェースが存在しない．L2 Classifier 内の転送テーブルを変更するには，Linux kernel 4.0 以降に導入されている switchdev の機構が必要であり，多くの Linux ディストリビューションでは，まだ最新の kernel を採用していないためである．そのため，VNF のアーキテクチャによっては，パケットを意図通りに転送できない場合がある．

つまり，VNF がハードウェア高速化技術として SR-IOV を利用している場合には，VNF が有する MAC アドレスは，ハイパーバイザが仮想 NIC に対して割り当てた MAC アドレスのみしか利用することはできず，L2 透過モードで動作する

ような VNF は，その実装方法によっては動作しない場合が発生することになる．さらに，L2 Classifier の実装はデバイスドライバ，もしくはハードウェアに依存しており，IPv6 マルチキャストパケットの転送などが実装されていない場合がある．そのため，PF を promiscuous mode で動作させる，もしくは VNF で想定される利用方法に適した L2 Classifier が実装されたデバイスドライバへの置き換えをおこなう必要がある．これは VNF の NFVI 環境への依存度を高める結果となる．

A.9　VNF の自動デプロイに向けた運用性

VNF を，オーケストレーターを介して自動的にデプロイするには，デプロイ方法の抽象化が必要である．VNF は VM として抽象化されているが，VNF のテンプレート，デプロイ手法，ライセンス認証が統一されていないため，現状では自動化が困難となっている．

A.9.1　VNF のテンプレートとデプロイ方法

本検証において提供いただいた VNF が採用しているデプロイ手法は，主に以下の六種類に分類される．

- OVA/OVF ファイルによるデプロイ
  主に VMware をハイパーバイザとする場合に採用される手法．VNF のハードディスクイメージと，メモリ量や CPU 数といった VM のメタデータを記述したファイルを利用した自動デプロイ手法である．PCI パススルーなど特定のデバイス ID 情報を必要とするようなオプションを設定する場合を除き，均一な VNF デプロイが可能となる．ただし，詳細なオプション設定のために，設定ファイルを手動で編集するには不向きである．

- OpenStack[27] を利用したデプロイ
  RDO[64] が提供する OpenStack を利用したデプロイ手法．Neutron によって OVS や SR-IOV のネットワークを，Glance によってハードディスクイメージを管理し，Nova を用いて起動する．SR-IOV の PCI-ID 等を管理するため，NFVI のハードウェア情報に依存する部分については一部手動設定が必要となる．また，VNF も OpenStack の VM デプロイ手法に対応している必要がある．

- Virt-manager によるデプロイ
  Virt-manager の GUI を利用したデプロイ手法．しかし，生成される Libvirtd の XML ファイルが，Virt-manager もしくは Libvirtd のバージョンによって異なるため，CPU Pinning 等一部の最適化に関するオプションの記述について，生成された XML ファイルを手動で修正する必要がある．

- Virt-install によるデプロイ
  Virt-manager の CLI を用いたデプロイ手法．GUI の場合と同様，生成される Libvirtd の XML ファイルが，Virt-manager もしくは Libvirtd のバージョンによって異なる．そのため，CPU Pinning 等のオプションに関しては，生成された XML ファイルを手動で変更する必要がある．

- libvirtd[65] の XML によるデプロイ
  Virt-manager もしくは Virt-install 後に生成される XML ファイルを，手動で微調整してデプロイする手法．Virt-manager の場合と同様，XML のフォーマットが Libvirtd のバージョンによって変化し，後方互換性が無い場合があるため，必ずしも XML ファイル形式での配布が適しているとは限らない．

- VNF 提供ベンダが配布する独自スクリプトによるデプロイ
  仮想 CPU やメモリ量，仮想 NIC の構成等を引数で指定可能なスクリプトによって VNF を定義しデプロイする手法．バックエンドに Libvirtd が利用されている場合が多いが，スクリプトの引数で指定可能なパラメータ以外は，スクリプト内に値が埋め込まれているため汎用性が低く，対象の VNF 以外では利用できない．しかし，VNF 毎の自動デプロイという点では，合理的な手法とも言える．

以上の通り，VMware と OpenStack によるデプロイでは，ハイパーバイザとその制御 API が固定されるため，均一な VNF デプロイ環境が構築できる．一方で，CPU Pinning や PCI パススルー ID といったハードウェアに密接なオプションの設定が難しい．

libvirtd を中心としたツール群 (Virt-manager や Virt-install) では，libvirtd のバージョンに依存した XML の記述仕様が存在するため，均一かつ動的な VNF のデプロイは，現状では困難である．一方で，簡単に手動で XML ファイルを編集することができるため，細かいオプション指定がやりやすいという利点がある．

現段階での対応状況としては，VMware によるデプロイは多くの VNF で対応しているが，OpenStack に関しては少数の VNF でのみ，対応が確認された．KVM におけるデプロイでは，Virt-manager を用いたデプロイ手順を解説したマニュアルが最も多かったが，バックエンドとなる Libvirtd によって生成される XML が異

なるため，XML のテンプレート自体を配布する方式を採用している VNF は存在しなかった．代わりに，XML の記述例がマニュアルに添付されている VNF はいくつか存在した．

この問題は RDO や OPNFV Arno[66] による，NFV に最適化された OpenStack が浸透し，ターゲットとなる NFVI 環境がソフトウェアバージョンの点で絞られる事によって，統一的なデプロイ方法が浸透していくと考えられる．OpenStack による VNF デプロイに関しては，OPNFV Arno の検証と平行して引き続き検討していく必要がある．

A.9.2　ライセンス認証

本検証において，VNF におけるライセンス認証方式は，VNF ベンダによって様々な方法があることが確認できた．ライセンス認証は，主にライセンス発行とライセンス検証のプロセスに分類される．現状では，各 VNF 単位で，異なるライセンス発行方式と検証方式が採用されている．

ライセンス発行手法は，主に三種類に分類される．UUID を基にシリアル番号を発行する手法と，UUID と無関係に発行される手法，また，利用できる合計資源量にもとづいて生成される手法である．

UUID を元にライセンスが発行される手法では，VNF をデプロイ後，CLI によって UUID を確認する手法，Web インターフェースから確認する手法が存在した．

UUID 番号と無関係にライセンスが発行される場合は，UUID との関連づけを持たないライセンスファイルが用意される．この場合は，ライセンスファイルは Web から取得したり，販社経由で直接渡されたりする．

合計資源数で制限する場合は，ネットワーク越しにライセンスサーバに対して通信をおこない，自身の資源量を伝えることでライセンスを取得する手法である．主に，ライセンスサーバが指定する CPU 数やコア数，ネットワーク帯域の上限等によって，VNF の動作を制限する．

また，ライセンスの認証は，ライセンスファイルを VNF にインポートして認証をおこなう手法と，ライセンスサーバに対してネットワーク越しに認証をおこなう手法が存在した．ライセンスファイルをインポートする方法は，TFTP，FTP，HTTP，SSH など，VNF によって異なる手法を利用する．

第一期検証にて集まった VNF だけ見ても，複数種類のライセンス認証方式が存在する．つまり，VNF のライセンス認証手法は統一されておらず，VNF 自体のデプロイとは別途おこなう必要がある．これは，VNF の簡易なデプロイを阻害する要因となるため，早急な統一化や標準化が必要と思われる．

A.10　NFVI のネットワーク構成とパケット転送性能

NFVI におけるネットワーク構成と VNF のネットワークアーキテクチャによって，VNF のパケット転送性能は大きく変化する．本検証では，表 A.1 で定義した NFVI 環境の違いによって，どの程度のパケット転送性能に差があるかを計測した．

A.10.1　仮想スイッチ実装による転送性能の違い

まず，仮想スイッチ実装による，パケット転送性能の違いを検証した．仮想スイッチを利用して，物理 NIC を複数の仮想 NIC に分割する場合，NFVI が提供する主なコンポーネントは，物理 NIC，仮想スイッチ，仮想 NIC の 3 種類である．つまり，仮想スイッチを利用した際のネットワーク I/O 性能は，これら 3 つのコンポーネントの中で最もボトルネックとなるものに依存して，性能が決定される．本検証では，主に仮想スイッチ，仮想 NIC の性能が，利用する NFVI 構成によってどのように変化するかを検証した．

まず，OVS のデータプレーンに Linux カーネルモジュールを利用した場合と，DPDK を利用した場合とを測定した．図 A.2 に，Linux カーネルモジュールをデータプレーンに利用した場合を示す．横軸はパケットサイズ (バイト) とフロー数を示しており，縦軸はワイヤーレートに対して転送できたパケット数の割合を示している．

Linux カーネルモジュールをデータプレーンに採用した場合，スループットでは約 1.7 倍から 2.1 倍，フォワーディングレートでは約 1.15 倍から 3 倍の範囲で，フロー数を増加させた事による性能向上が確認された．

これは，物理 NIC から Linux カーネルに対するネットワーク I/O において，物理 NIC のマルチキューを複数の CPU で受信する Receive-Side Scaling (RSS) が有効に働いたためである．一方でスループットとフォワーディングレートの間には約 2 倍から 6 倍の転送レートの差が見られ，安定したネットワーク I/O が実現できていないことが分かった．

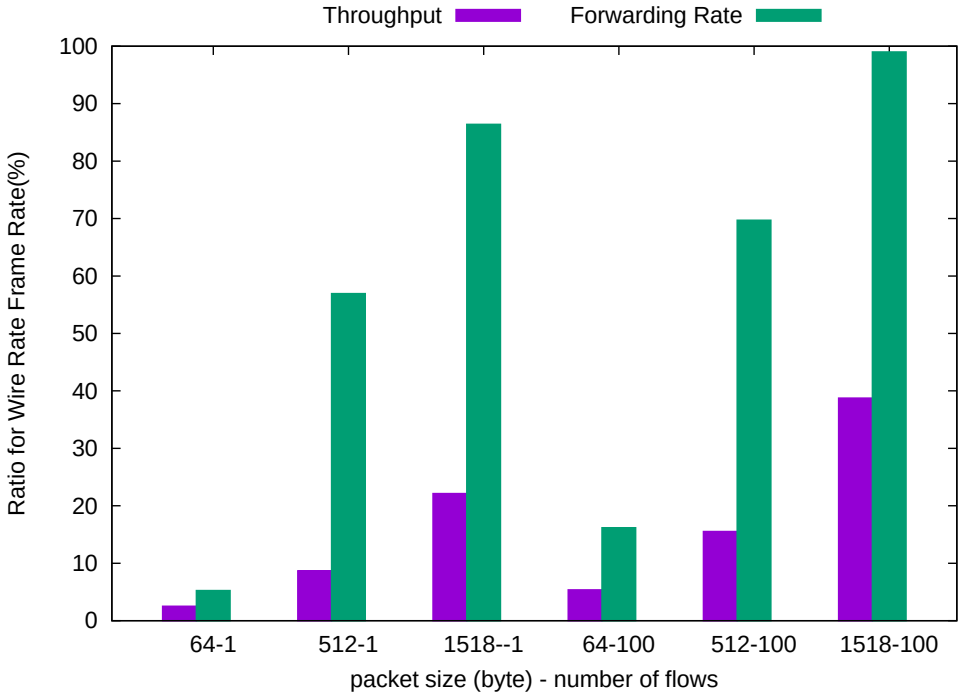


図 A.1: Linux Kernel Module をデータプレーンとした OVS のパケット転送性能

さらに，DPDK をデータプレーンにした場合を図 A.2 に示す．横軸はパケットサイズ (バイト) とフロー数を示しており，縦軸はワイヤーレートに対して転送できたパケット数の割合を示している．DPDK をデータプレーンに使用した場合，64 バイトのショートパケットの場合を除いて，ワイヤーレートでのパケット転送を実現している．また，フロー数の変化による性能の変化が非常に小さい．この理由は物理 NIC のポートに 1 つに対して，1 つの CPU を割り当てて，ポーリングによって I/O をおこなっているためである．そしてスループットとフォワーディングレートの間での性能差も非常に小さく，安定したネットワーク I/O が実現できている事が分かる．

つまり，10Gigabit Ethernet において，仮想スイッチのデータプレーンに DPDK を利用した場合，ほぼネットワーク I/O 部分のボトルネックは存在しないと言える．

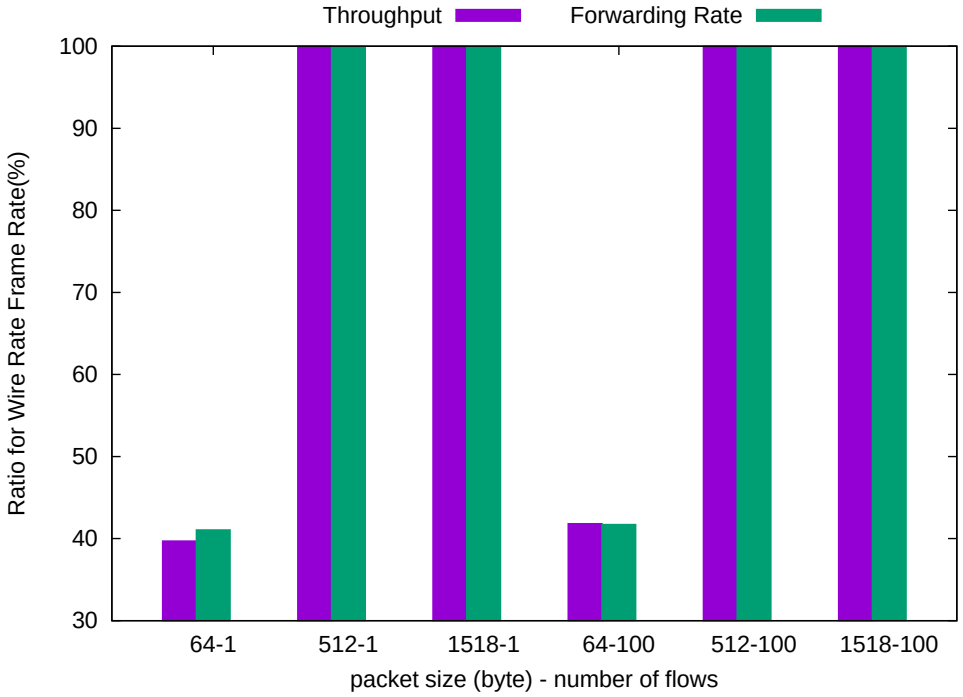


図 A.2: DPDK をデータプレーンとした OVS のパケット転送性能

A.10.2　仮想 NIC 実装による転送性能の違い

次に，仮想 NIC と仮想スイッチを組み合わせた場合のネットワーク I/O 性能を測定した．測定においては，VMware では VMware 内蔵仮想スイッチと VMXNET3 の組み合わせ，KVM では OVS と virtio の組み合わせを利用した．

仮想 NIC のパケット転送性能を左右する要素として，マルチキュー機能がある．マルチキュー機能を実現するには，ハイパーバイザ側の仮想 NIC が対応しているだけでなく，VNF のデバイスドライバと OS の割り込みハンドラーが対応している必要がある．そのため，VNF に対する CPU の割当数と，フロー数を増加させる効果は，その VNF がどこまでマルチキューに対応しているかによって異なる．

図 A.3 に，ある VNF において VMware の仮想スイッチと VMXNET3 を利用したパケット転送性能を示す．図示した VNF をはじめ，いくつかの VNF では，フロー数を増加させる事による性能の向上は確認できなかった．VMXNET3 自体は，デフォルトでマルチキュー機能に対応しているため，VNF となっているゲスト OS 側のデバイスドライバとカーネルが，マルチキューに未対応であったことが想定される．一方で，後述する OVS と virtio を利用した場合と比較して，スループッ

トとフォワーディングレートの性能差が小さく，安定したネットワーク I/O が実現できていることがわかる．また，スループットにおいて約 2 倍から 4 倍の性能差が確認された．

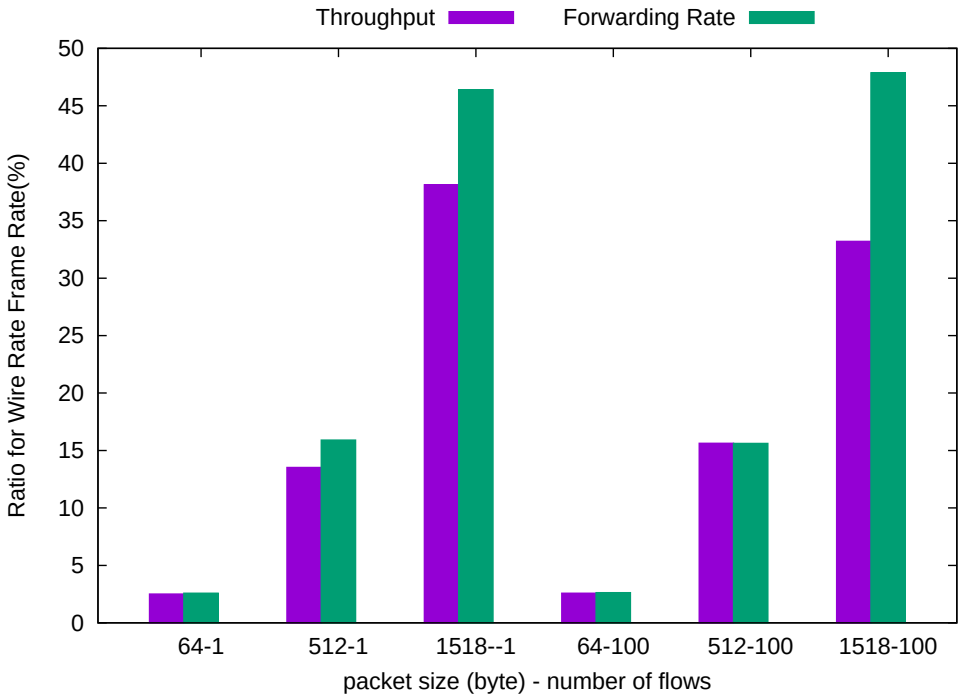


図 A.3: VMware 仮想スイッチと VMXNET3 を利用した VNF のパケット転送性能

図 A.4 に，図 A.3 の測定と同じ VNF を KVM 上で OVS と Virtio を利用したパケット転送性能を示す．VMXNET3 と同様にフロー数を向上させる事による性能の向上は確認できなかった．また，VMXNET3 と比較してパケット転送能力は低く，スループットとパケットフォワーディングレートの性能差も大きかった．

A.10.3　SR-IOV を利用した VNF の性能測定

SR-IOV をネットワーク I/O に採用する VNF の場合，主にカーネルドライバ型 VNF とユーザスペース型 VNF の 2 種類のアーキテクチャに分類される．

カーネルドライバ型の VNF は，SR-IOV によって仮想化された PCI デバイスを，パススルー技術によって VNF (ゲスト OS) から直接制御し，VNF (ゲスト OS) のカーネルにてパケット転送処理をおこなう．図 A.5 に，あるカーネルドライバ型 VNF のパケット転送性能を示す．カーネルドライバ型の VNF は，ゲスト OS

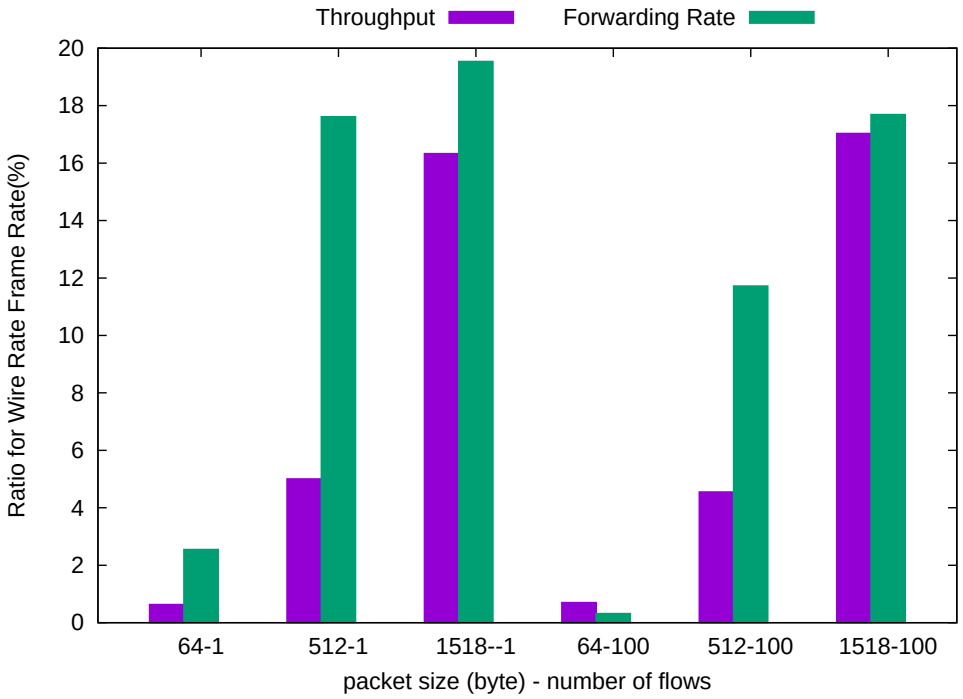


図 A.4: Linux Kernel Module をデータプレーンとした OVS と VirtIO を利用した VNF のパケット転送性能

からは TX と RX キューが 2 ペアある PCI デバイスとして見えるため，フロー数の増減による性能の変化が小さい．また，ベアメタル上にて動作させた Linux OS と比較しても大きな性能差が少ない事から，Linux OS 自身のネットワーク I/O 性能がボトルネックとなっていると考えられる．また，カーネドライバ型は，VNF 内の OS で処理される割り込み等が，qemu-kvm プロセス内，すなわちゲスト OS 内で完結するため，vcpu pining をおこなう事によって，他の VNF との資源競合を回避しやすいという利点がある．

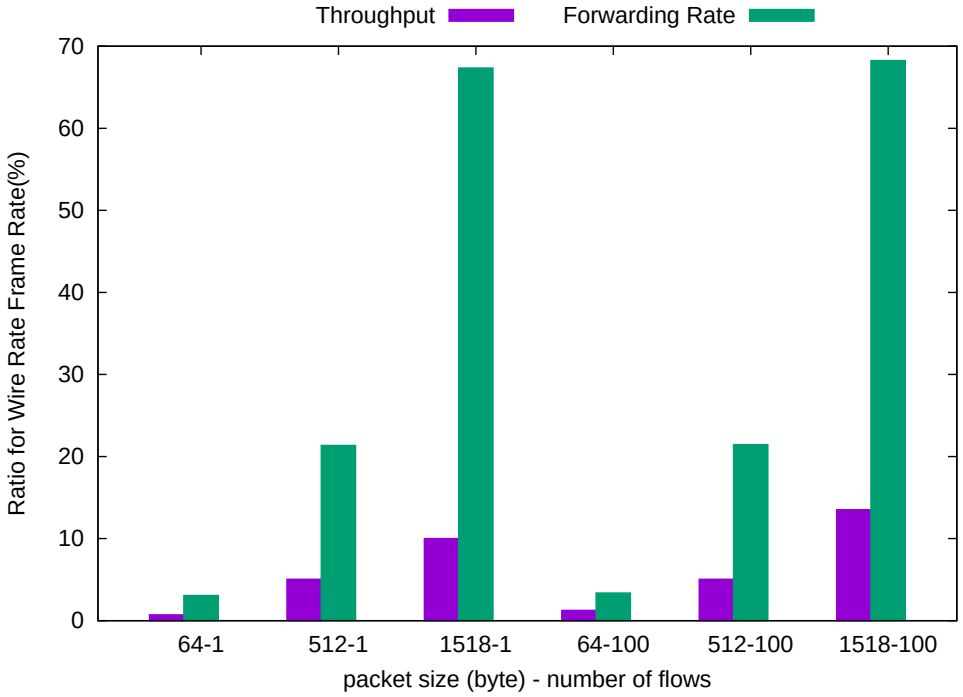


図 A.5: カーネルドライバ型 VNF のパケット転送性能

ユーザスペース型の VNF は，SR-IOV によって仮想化された仮想 PCI デバイスを，パススルー技術によって VM にアタッチし，VNF (ゲスト OS) 自身が Intel DPDK 等を利用して，ユーザスペースでパケット転送処理をおこなう．

ユーザスペース型は，主にパケット I/O に対して専用の CPU コア割当を要求する．傾向としては，PCI パススルーによって占有したポートに対して，1 コアを割り当てるよう推奨される．図 A.6 に，あるユーザスペース型 VNF のパケット転送性能を示す．フレームサイズ 64 バイトのショートパケットでは，7Gbps 程度の性能であるが，フレームサイズ 512 バイト以上ではワイヤーレートの転送性能を実現している．また，占有する仮想ポートに対して，CPU コアを 1 つ割り当てるため，フロー数を増加させる事による並列処理は発生しない．そのため，1 フローによる測定と 100 フローによる測定に性能差が存在しない．ただし，性能を確保するため，CPU コアの割当てに関していくつかの制限がある．今回の検証で確認された制限としては以下の 2 点である．

- 物理 CPU コア ID を明示的に指定する．HyperThreading に対応した CPU の場合，BIOS 上で明示的に HyperThreading 機能を無効にする，もしくは物

理 CPU コア ID を共有する論理 CPU コアの中から一つだけ CPU ID を指定して利用する．

- 複数の物理プロセッサが存在する場合，同じ VNF 内では一つの物理プロセッサ内のコアのみを利用する．昨今の CPU アーキテクチャは NUMA を採用しており，複数のプロセッサが共有するメモリへのアクセスコストが不均質である．そのため，ある VNF において利用する物理プロセッサを単一に制限し，メモリアクセスコストを一定にするよう留意して割当をおこなう．

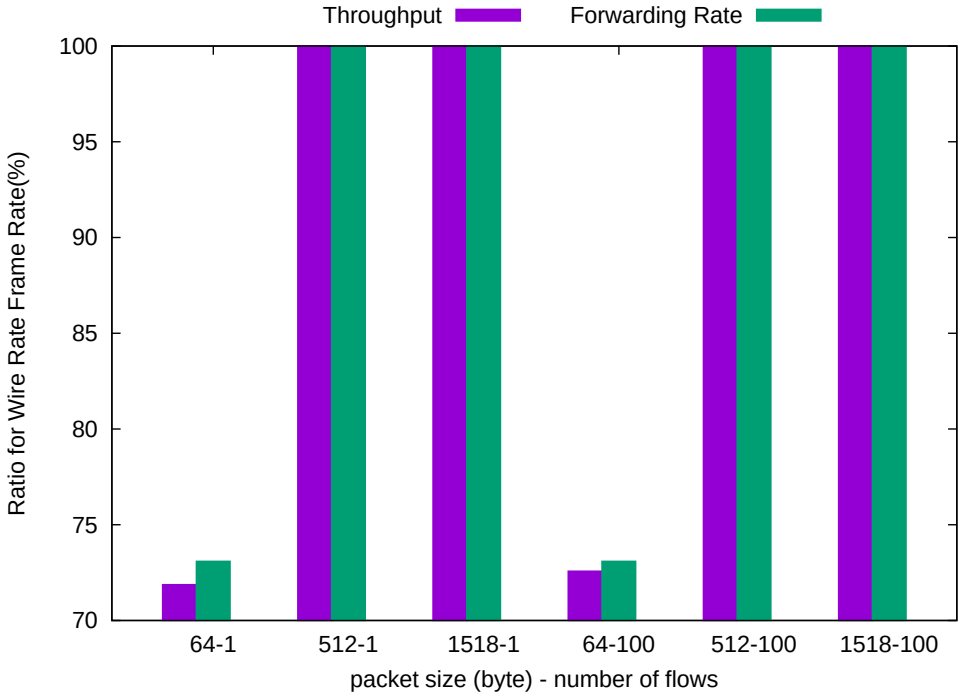


図 A.6: ユーザスペース型 VNF のパケット転送性能

A.10.4　VXLAN オフロードによる転送性能

Linux における VXLAN のオフローディング機能は，実質的には TCP Segmentation Offloading(TSO) 機能などのフラグメント処理をハードウェア上でおこなう．図 A.7 に，VXLAN オフローディングを利用したパケット転送性能計測環境を示す．Routing は，ベアメタルの Linux が パケットをフォワーディングする L3 ルータとして動作し，パケットの出力をおこなう際に VXLAN にカプセル化する

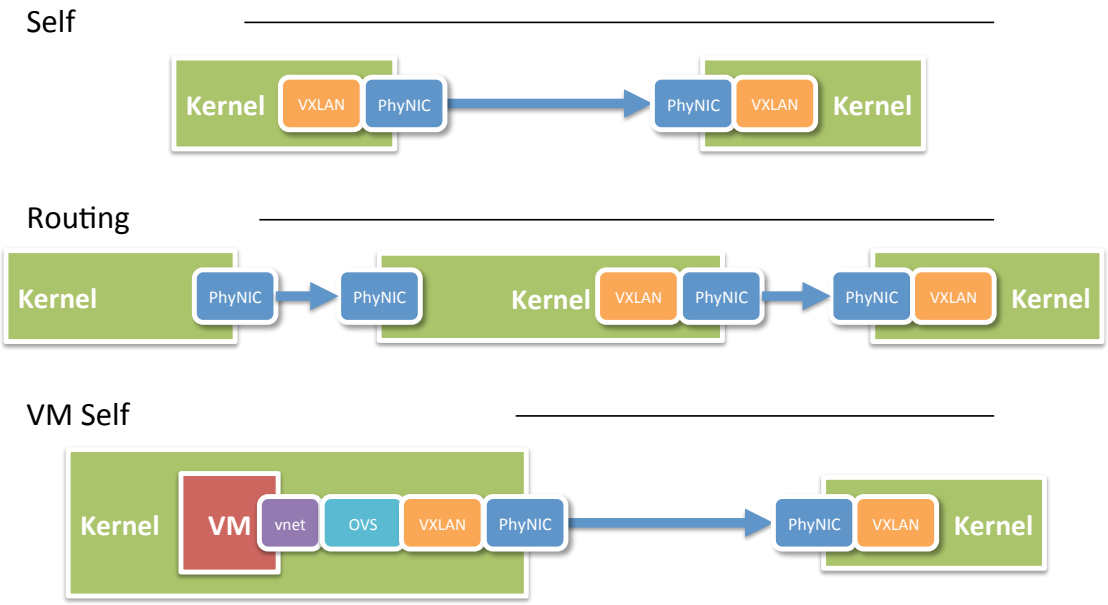


図 A.7: VXLAN の測定環境一覧

場合である．Self は，ベアメタルの Linux が，自ら TCP フローを生成し，VXLAN にカプセル化する場合である．VM-Self は，VM が TCP フローを生成し，Linux カーネルモジュールをデータプレーンとして，OVS を介して VXLAN にカプセル化する場合である．

図 A.8 に，Linux カーネルにて TSO を有効にした状態で，VXLAN オフローディングを無効にした場合と，有効にした場合のパケット転送性能を示す．Self と VM-Self においては，VXLAN オフローディングを有効にすることで，1.1 倍から 1.2 倍の性能向上が確認できた．一方，Routing においては，VXLAN オフローディングを有効にした場合と無効にした場合で，性能に変化は無かった．

したがって，VXLAN オフローディングが有効なホスト自体，もしくはそのホスト内の VM (VNF) がパケットを生成する際には，VXLAN オフローディングは有効に機能するが，パケット転送をおこなうルータやスイッチとして動作する場合，VXLAN オフローディングは，その効果を発揮できないことが確認できた．

つまり，パケット転送をおこなう，vRouter や vSwitch といった VNF では VXLAN オフローディングによる性能向上は見込めないが，一方で，ロードバランサ等の，一回 TCP セッションを VNF 自身が中継するような場合には，VXLAN オフローディングが有効に機能すると考えられる．

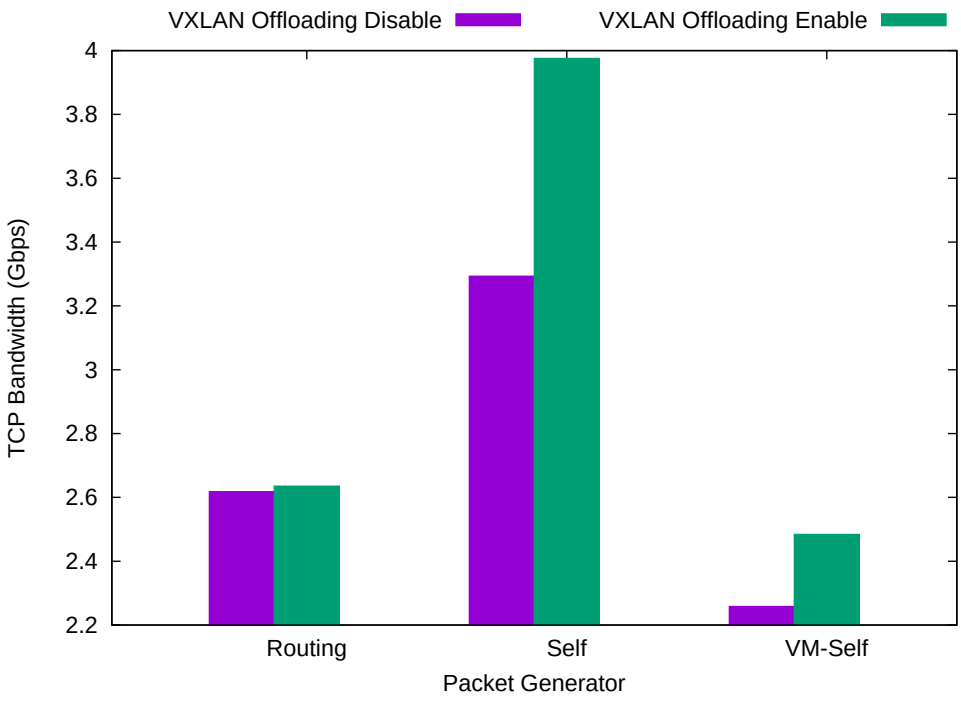


図 A.8: VXLAN オフローディングを有効にした際の TSO の効果