特別研究報告

題目

ネットワーク仮想化にもとづくサービス機能の再配置がユーザーの通信品質に与える効果の評価

指導教員
村田 正幸 教授

報告者
金田 純一

平成 29 年 2 月 14 日

大阪大学 基礎工学部 情報科学科

平成 28 年度 特別研究報告

ネットワーク仮想化にもとづくサービス機能の再配置がユーザーの通信品質に与える効果の評価

金田 純一

内容梗概

近年、IoT (Internet of Things) の進展を背景に数多くの新しいアプリケーションやサービスが登場し情報ネットワークは急激に変化している。新しいアプリケーションやサービスの例として、カメラやセンサーなどを搭載したエンド端末において取得された情報を、別拠点のデータセンターで処理し、結果をエンド端末へ提示するサービスが考えられている。しかし、エンド端末とデータセンターが地理的に離れることや、大量のデータがデータセンターに集中することで、遅延が増大する。このような問題に対応すべく、ネットワークそのものに柔軟性を持たせる一つの方法としてネットワーク機能仮想化 (NFV: Network Functions Virtualization) が期待されている。さらには、ネットワーク機能だけではなくアプリケーション機能を仮想化してモバイルエッジに配置し、地理的な遅延の解消および負荷の分散によるアプリケーションやサービスに対する応答性向上を期待する、モバイルエッジコンピューティング (MEC: Mobile Edge Computing) の導入が進められている。

モバイルエッジコンピューティングによって応答性向上が期待される一方で、仮想化環境でのソフトウェア動作による処理速度の低下が懸念される。そこで、本報告では、実機を用いたモバイルエッジコンピューティング環境を構築し、アプリケーションやサービスを柔軟に提供した際に生じるユーザーの通信品質に与える効果を明らかにする。その結果、ネットワーク仮想化によるソフトウェア動作から生じる処理速度低下は、地理的な要因によって生じる遅延に比べ十分小さく、サービス機能を、遠隔地からユーザーに近い拠点に再配置することにより、通信遅延時間が最大でおよそ 30%低減され、ユーザーの通信品質が改善されることが明らかとなった。

主な用語

ネットワーク機能仮想化 (NFV: Network Functions Virtualization)、モバイルエッジコンピューティング (MEC: Mobile Edge Computing)、IoT (Iternet of Things)、OpenMANO

目 次

図 目 次

表 目 次

1　はじめに

近年、IoT (Internet of Things) の進展を背景に数多くの新しいアプリケーションやサービスが登場し情報ネットワークは急激に変化している。新しいアプリケーションやサービスの例として、カメラやセンサーなどを搭載したエンド端末において取得された実世界の情報を、別拠点のデータセンターへ転送し、画像認識や音声認識などの高負荷な処理を行い、結果をエンド端末へ提示するサービスが考えられている。しかし、情報を発信する端末と情報を処理するデータセンターが地理的に離れることとなり遅延が発生する。さらに、多数のエンド端末から発生する膨大なデータが集中し、データセンターの処理量が増大し、結果として遅延が増大する [1,2]。

このような問題に対応すべくネットワークの設計・運用、ネットワークサービスそのものに柔軟性を持たせる一つの方法としてネットワーク機能仮想化 (NFV: Network Functions Virtualization) が期待されている。NFV は、クラウドコンピューティングとともに発展したサーバー仮想化技術によって、ルーターやファイアウォール、ロードバランサなど、従来、専用機器で運用されてきたネットワーク機能の仮想化を行い、ハードウェア資源となる汎用サーバー上の仮想マシン (VM: Virtual Machine) で運用する考え方である [3]。NFV では、仮想化されたネットワーク機能を VNF (Virtual Network Function) と呼ぶ。NFV では、ネットワーク機能をソフトウェア処理し、ハードウェア資源と分離することで、それらを柔軟かつ動的に展開し動作させることが可能となり、ネットワークの設計・運用、サービスそのものに柔軟性が生まれる。

さらに、ネットワークの研究分野では、エンド端末が必要とする一連の情報処理および情報連携の一部を、よりエンド端末に近いネットワーク外縁部に配置したサーバーで行う、モバイルエッジコンピューティング (MEC: Mobile Edge Computing) の導入が進められている。その中で、展開するアプリケーションやサービスの機能拡張や規模の拡大を柔軟に行うため、ネットワーク外縁部に配置するエッジサーバーの仮想化に NFV を応用することが考えられている [4–6]。しかし、モバイルエッジコンピューティングが、地理的な遅延の解消および負荷の分散によるアプリケーションやサービスに対する応答性向上を期待する一方で、NFV において VNF が展開する仮想マシンは、ハイパーバイザーやエミュレーターなどの仮想化ソフトウェアの上に成り立っており、専用機器で動作させた場合に比べて、ソフトウェア動作による処理速度の低下が懸念される。また、NFV によって設計された仮想ネットワークでは、VNF が稼働する仮想マシンに割り当てる計算機資源 (CPU コア数やメモリ量など) やストレージ、ネットワーク資源 (リンク数、帯域幅など) に加え、仮想マシン自身が動作するサーバーが動的に変更されうる。そのため、それによる利用するアプリケーションやサービスに対する応答性への影響が懸念される。

そこで本報告では、実機を用いて、モバイルエッジコンピューティング環境で生じる通信遅延を測定する。モバイルエッジコンピューティング環境を構築するにあたり、NFV のオープンソース実装である OpenMANO [7] を用いる。OpenMANO を用いて構築されるネットワーク仮想化環境において、ネットワーク機能ではなく、アプリケーションサービスを仮想化システム上で動作させ、モバイルエッジコンピューティング環境を構築する。また、買い物代行サービスなどのモバイルエッジコンピューティングのアプリケーションを想定し、ユーザーと遠隔地にあるロボットが映像取得や音声発信を行い連携するシステムを作成した。本システムにおいて、映像加工を行う拠点を変更しつつ通信遅延時間を測定している。

本報告の構成として、まず 2 章でネットワーク機能仮想化 (NFV) およびモバイルエッジコンピューティング (MEC) について述べる。次に 3 章で、OpenMANO を用いたネットワーク機能仮想化環境の構築について説明し、4 章で、作成したサービスと利用したアプリケーションの説明を行い、実験環境とその結果について述べ、評価を行う。最後に、5 章で本報告のまとめと今後の課題について述べる。

2　関連研究

本章では、本報告に関連し、近年関連研究が盛んに行われているネットワーク機能仮想化 (NFV) およびモバイルエッジコンピューティング (MEC) について標準化動向をもとに説明し、検討されているモバイルエッジコンピューティングのユースケースを紹介する。

2.1　ネットワーク機能仮想化 (NFV)

近年のプロセッサー技術の進展により、ネットワーク機器をソフトウェア制御した場合でも十分な性能が期待できることから、資源の増減が任意に行え、専用機器の管理が不要な、汎用サーバー上でネットワーク機能を仮想化し運用することが現実味を帯びてきている。NFV では、サーバー仮想化技術を用いて、ルーターやファイアウォール、ロードバランサなど、従来、専用機器で運用されていたネットワーク機能を、ハードウェア資源となる汎用サーバー上の仮想マシンで運用し、ネットワークの設計・運用、サービスそのものに柔軟性を持たせることができる [3]。NFV の標準化は、ETSI (European Telecommunications Standards Institute) の配下に設立された ISG (Industry Specication Group) によって進められている。NFV ISG によって NFV の標準化に関するホワイトペーパーが複数発行されており、それらのうち文献 [8, 9] に記載のある NFV のフレームワークについて、図 1 を用いて主要な構成要素を中心に説明する。

VNF (Virtual Network Function)

VNF は、従来専用の機器で運用されてきたゲートウェイや DHCP サーバー、ファイアウォールなどのネットワーク機能が仮想化、つまりソフトウェアで実現されたものである。従来の専用機器と機能的な差はない。単体の仮想マシン上での実行に加え、VNF の内部コンポーネントを別々に複数の仮想マシンで実行することも考えられている。また、VNF は自身を管理する EMS (Element Management System) と合わせて運用される。

NFVI (NFV Infrastructure)

NFVI は VNF を展開させるために必要な環境そのもので、ハードウェアとソフトウェア、両方から成る。ハードウェア資源は、計算資源とストレージ、ネットワーク資源に分けられ、ハイパーバイザーなどの仮想化レイヤーを介して、それぞれ情報処理、データの保管、VNF 同士の接続を担う。計算資源には汎用サーバーの使用が、ストレージには NAS (Network Attached Storage) とサーバー自身の持つストレージの 2 種類が想定されている。これらの

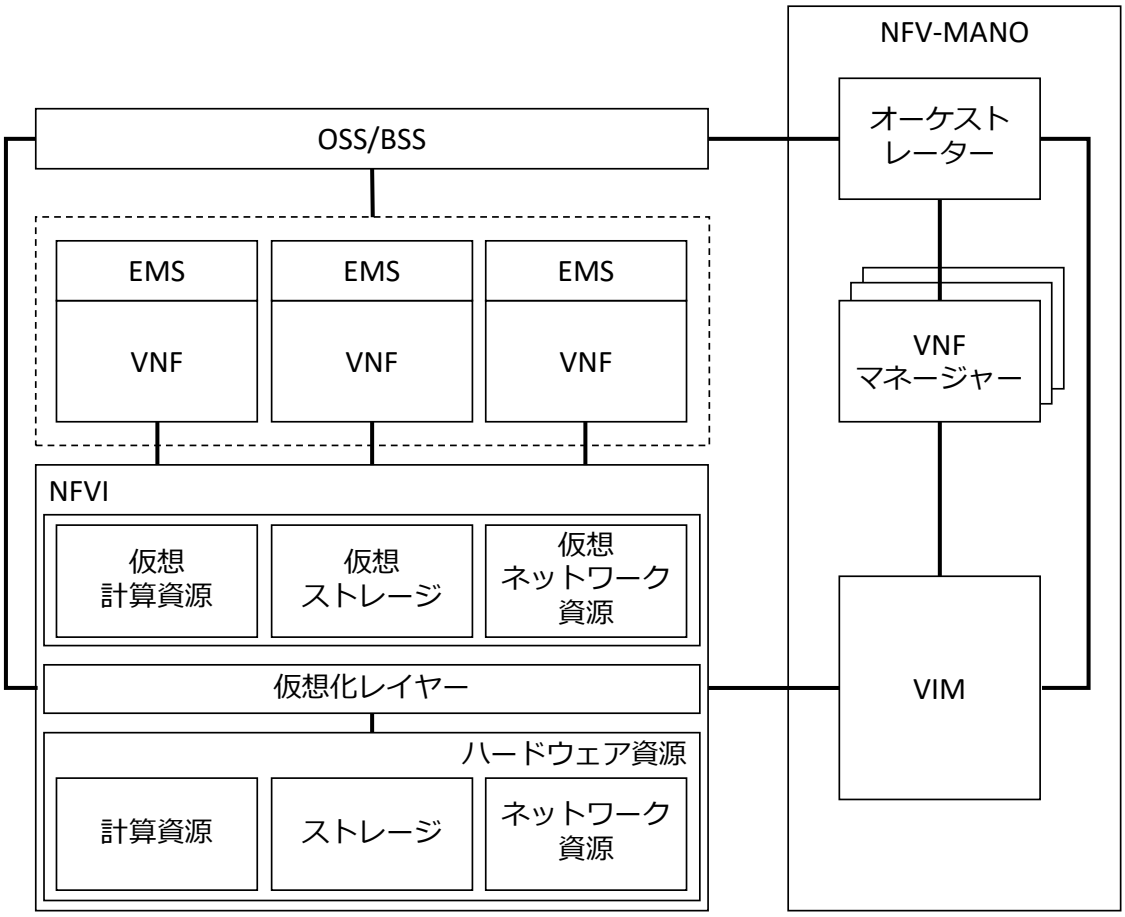

図 1: NFV のフレームワーク

ハードウェア資源は、数カ所に別れて異なる場所で稼働することが想定されており、ネットワーク資源は拠点間を接続するものと、拠点内のハードウェア同士を接続するものに分けられる。また、ハードウェア資源は仮想化レイヤーによって仮想化され、VNF を実行する仮想マシンとして提供される。つまり、ハードウェア資源と VNF は、仮想化レイヤーによって分離されているため、VNF を物理的に異なるハードウェア上で展開することが可能となる。より良い性能を得るため、仮想マシンがホストとなるサーバーのネットワークインターフェースなどのハードウェア資源に、直接接続できる特別な仕組みが存在するが、NFV は標準的なハードウェアの仮想化を推奨している。さらに仮想化レイヤーは、仮想マシン同士の通信のため VLAN を用いて仮想ネットワークを提供する。

NFV-MANO (NFV Management and Orchestration)

MANO は NFVI および VNF の管理・調整を行う、NFV の最も重要な構成要素である。NFVI の管理・調整は、MANO の構成要素の 1 つである VIM (Virtualised Infrastructure Manager(s)) が担当する。VIM は、NFVI として使われる仮想化ソフトウェア、計算資源、ストレージ、ネットワーク資源の構成情報を記録し、仮想マシンへの各資源の割り当てや、エネルギーの効率化、解放された資源の再利用を行う。それらの管理業務に加え、NFVI 管理とその可視化、NFVI の視点でパフォーマンスに関する問題の根本原因解析 (RCA: Root cause analysis)、障害およびキャパシティープランニング、モニタリング、最適化などの情報収集を行う。また、MANO の他の構成要素として、VNF の生成から消滅まで VNF を管理する VNF マネージャー、VIM と VNF マネージャーを統括し、NFVI 上で実行されるネットワークサービスを管理、調整するオーケストレーターが存在する。さらに、MANO が既存のネットワークで広く用いられている管理システムである OSS/BSS (Operation Support Systems / Business Support Systems) と相互作用し、NFV が外部にある既存管理システムと統合することも想定されている。

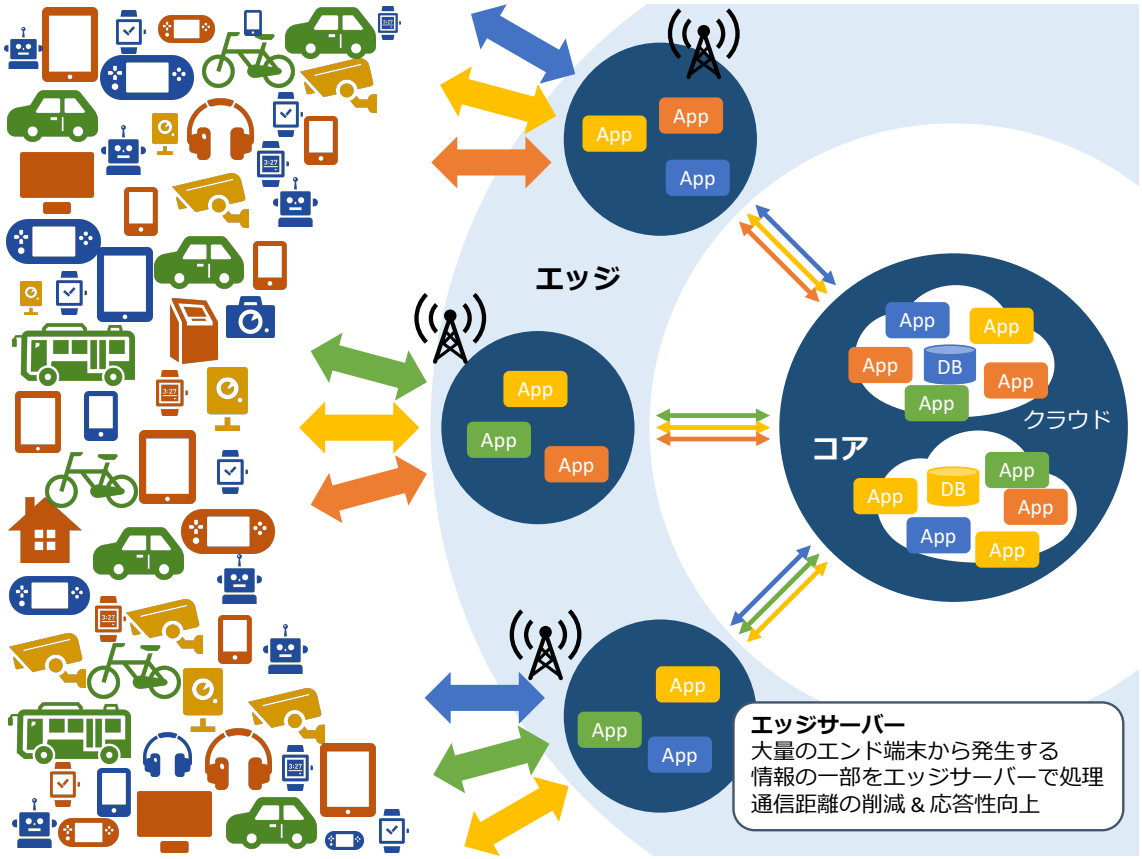

図 2: モバイルエッジコンピューティングの概念図

2.2 モバイルエッジコンピューティング (MEC)

現代では、多くの端末がネットワークに接続され、ユーザーはクラウドサービスとして提供されている多くのアプリケーションを利用している。これらのアプリケーションの背後では、サービスプロバイダーがデータセンターに多数のサーバー端末を集中配置し、そこでプログラムを動作させている。しかし、IoT のマーケットの需要は高まる一方であり、さらに数多くの多種多様な端末がネットワークに接続され、サービスの提供が行われると推測される。そのため、ネットワークに対して、より高い拡張性が要求されている。モバイルエッジコンピューティングは、これまでデータセンターで集中的に行ってきた処理の一部や、エンド端末で行ってきた処理を、エンド端末に近い場所に存在するエッジサーバーで行うことにより、遅延の低減、応答性の向上を目的としている。さらに、広告や宣伝などに応用可能な、エッジサーバーの地域性にもとづいた付加価値の創出や、処理拠点の分散によるビッグデータの処理能力向上などが期待される。モバイルエッジコンピューティングの概念図を図

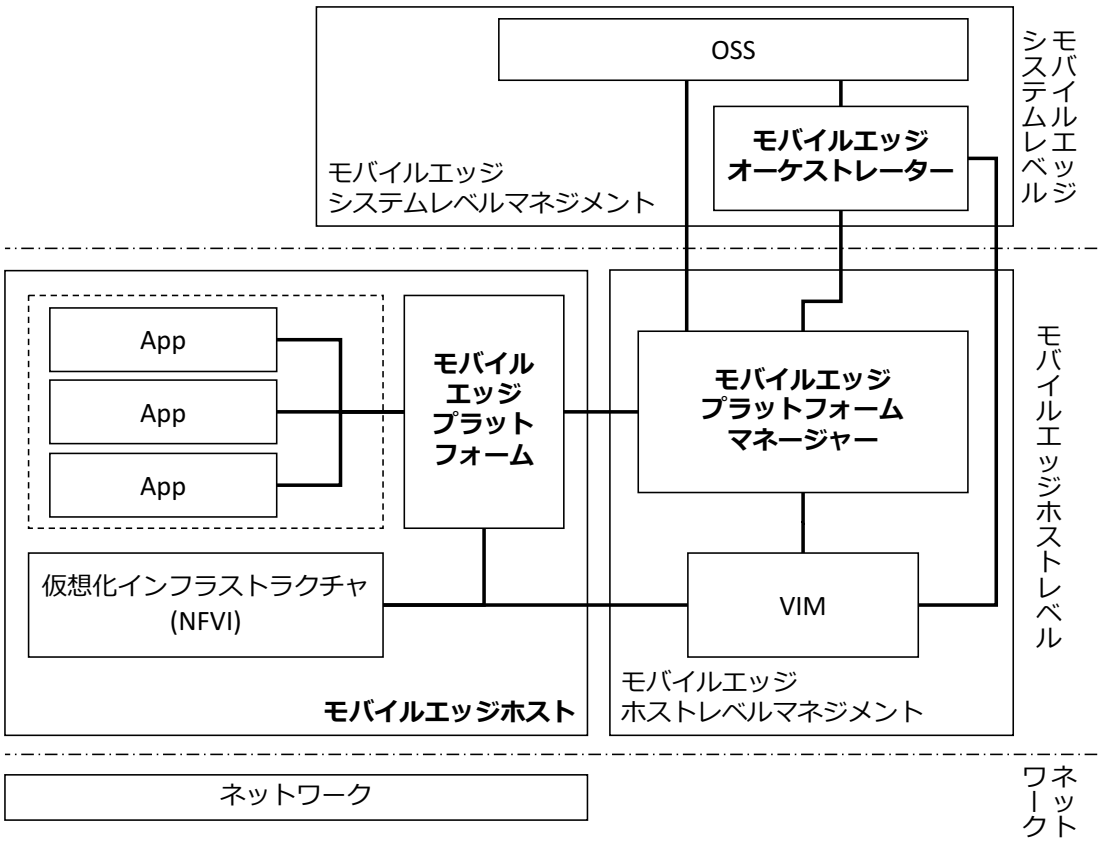

図 3: モバイルエッジコンピューティングのフレームワーク

2 に示す [4]。また、モバイルエッジコンピューティングの標準化は、ETSI の配下に設立された MEC ISG によって進められている。文献 [6] によれば、展開するアプリケーションやサービスの機能拡張や規模の拡大を柔軟に行うため、ネットワーク外縁部に配置するエッジサーバーの仮想化に、NFV を応用することが検討されている。以下では、文献 [10] に記載のあるモバイルエッジコンピューティングのフレームワークについて、図 3 を用いて主要な構成要素を中心に説明する。

モバイルエッジホスト

モバイルエッジホストは、仮想化インフラストラクチャとモバイルエッジプラットフォームによって構成され、アプリケーションが実行される環境を構築する。仮想化インフラストラクチャは、NFV の NFVI などが該当し、計算資源、ストレージ、ネットワーク資源をアプリケーションに提供する。また、アプリケーションと、DNS サーバーやローカルネッ

トワーク、外部ネットワークを接続するデータプレーン機能を提供する。アプリケーションは、仮想マシン上で実行され、稼働率や資源の割り当てなどに関してモバイルエッジプラットフォームと相互にやり取りを行う。さらに、アプリケーションは、自身に関する様々な制御情報や資源割り当て要求を行う。これらはモバイルエッジシステムマネジメントによって検証され、拒否された場合は規定値が採用される。

モバイルエッジプラットフォーム

モバイルエッジプラットフォームは、モバイルエッジホスト内において、アプリケーションが実行される環境を提供する。また、モバイルエッジプラットフォームマネージャーから、データプレーンのトラフィック制御情報と DNS レコードを受け取り、それぞれデータプレーンを構成する機器と、DNS サーバーへ指示を行う。さらに、ストレージと時刻情報の継続的な提供を行う。

モバイルエッジプラットフォームマネージャー

モバイルエッジプラットフォームマネージャーは、アプリケーションの作成から消滅まで管理を行い、関連情報をモバイルエッジオーケストレーターに通知する。さらに、最も基本的な管理機能をモバイルエッジプラットフォームに提供する。また、データプレーンのトラフィック制御情報と DNS レコードの管理を行い、これらの競合が起こらないようにする。

モバイルエッジオーケストレーター

モバイルエッジオーケストレーターは、モバイルエッジシステムマネジメントにおいて中心的な役割を果たす。展開されたモバイルエッジホスト、利用可能な資源とサービス、各機器の接続関係など、全体の構成情報をもとに、モバイルエッジシステム全体の保守を行う。また、信頼性の検証など、アプリケーションパッケージのテストを行い、アプリケーションに関する様々な制御情報や資源割り当て要求を検証し、オペレーターのポリシーに応じて調整を行う。これらを記録し、アプリケーションの実行環境を仮想化インフラストラクチャ上に準備する。さらに、アプリケーションをインストールする適切なモバイルエッジホストを、通信遅延や利用可能な資源などの観点から選定し、アプリケーションの作成、消滅、再配置を決定する。

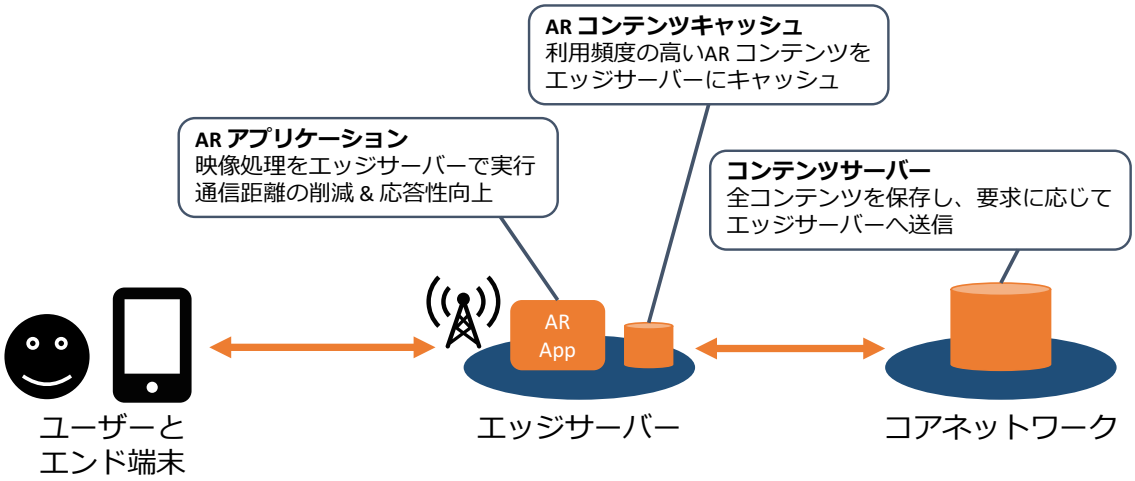

図 4: モバイルエッジコンピューティングによる AR コンテンツの配信

2.3 モバイルエッジコンピューティングのユースケース

文献 [4] は、モバイルエッジコンピューティングのユースケースとして様々な事例を挙げている。例えば、エッジサーバーの地域性を活用し、GPS を利用せず、ネットワークの計測によってエンド端末が位置情報を取得できるようにする。広告やスマートシティ、来客者数の解析などのサービスに役立つことが期待される。さらに、図 4 に示すような、AR (Augmented Reality) コンテンツの配信サービスも考えられている。AR は、実世界の映像に情報を付加しユーザーに表示するため、高いリアルタイム性が求められる。そこで、エッジサーバーに利用頻度の高い AR コンテンツをキャッシュし、エンド端末に配信することで、往復遅延時間 (RTT: Round-Trip Time) の低減、高い帯域幅の確保が期待できる。AR は、すでに利用可能なサービスが多数存在し、将来的には、屋内外問わず様々な場所で活用できると考えられている。また、監視カメラの解析サービスも挙げられる。街の至る所に設置された監視カメラを無線ネットワークで結び、その映像を基地局で展開するエッジサーバーで解析する。エッジサーバーは解析結果から重要なイベントが発生した場合のみ、そのイベント情報、メタデータ、ビデオの一部をバックエンドのサーバー内ストレージに保存する。こうすると、データ量が多く広い帯域を必要とする映像の伝送を、ネットワークの外縁部でとどめ、中央部のコアネットワークではデータ量の少ない情報のみを扱うことができる。加えて、文献 [11] では、モバイルエッジコンピューティングによって自動車の安全運転支援や観光ナビゲーションサービスを実現することについて述べており、さらに、将来の高度な気象予測への適用可能性を示唆している。

3　OpenMANO を用いたネットワーク機能仮想化環境の構築

本報告では、ネットワーク仮想化環境の構築にあたり、NFV のオープンソース実装である OpenMANO [7] を利用した。ただし、OpenMANO を用いて構築されるネットワーク仮想化環境においてネットワーク機能を動作させるのではなく、アプリケーションサービスを仮想化システム上で動作させ、モバイルエッジコンピューティング環境を構築する。OpenMANO は、スペインの大手通信事業者 Telefonica の内部組織 NFV Reference Lab によって作成された、NFV-MANO の実装に重点を置いたオープンソースプロジェクトである。現在 NFV および NFV-MANO の実装は、いくつかのプロジェクトで進められている [12]。選考にあたり、まず、図 1 に示した NFV フレームワークの実現を重視する OPNFV (Open Platform for NFV) [13] と OpenMANO の使用を検討した。OPNFV は Linux Foundation と共同で NFV のフレームワーク全体を既存のオープンソースコンポーネントの統合によって実装することを目指しているが、NFVI の実装が完了した段階であり、NFV-MANO の実装に至っていない。そのため、本報告では OpenMANO を用いることとした。なお、OpenMANO の開発元である Telefonica NFV Reference Lab は、現在、NFV-MANO として想定されている機能の一部を提供するソフトウェアの開発元と連携し、ETSI の傘下において、OSM (Open Source MANO) [14] の開発を開始しているが、現段階では動作実績が不十分であるため OpenMANO を用いている。本章では、OpenMANO を用いたネットワーク機能仮想化環境の構築について説明する。

3.1　OpenMANO アーキテクチャ

図 5 に、OpenMANO のアーキテクチャを示す [15]。図中右部に示しているのは、OpenMANO の 3 つの主要モジュール openvim、openmano、openmano-gui である。また、コンピュートノード、OpenFlow スイッチ、OpenFlow コントローラー、イメージストレージなど、主要モジュール以外は外部コンポーネントと呼ばれる。OpenMANO では、OpenFlow コントローラーとして FloodLight を用いる。OpenMANO の動作には以下の 2 つのノードが必要である。

- コントローラーノード

  コントローラーノードでは、図 5 中で囲まれている OpenMANO のモジュールと、OpenFlow コントローラーが実行される。OpenMANO を用いたネットワーク機能仮想化環境を構成する各機器は、制御用の接続としてコントローラーノードと物理的に接続される。この接続をコントロールプレーンと呼び、VNF が扱うデータが流れるデータプレーンと分離する。

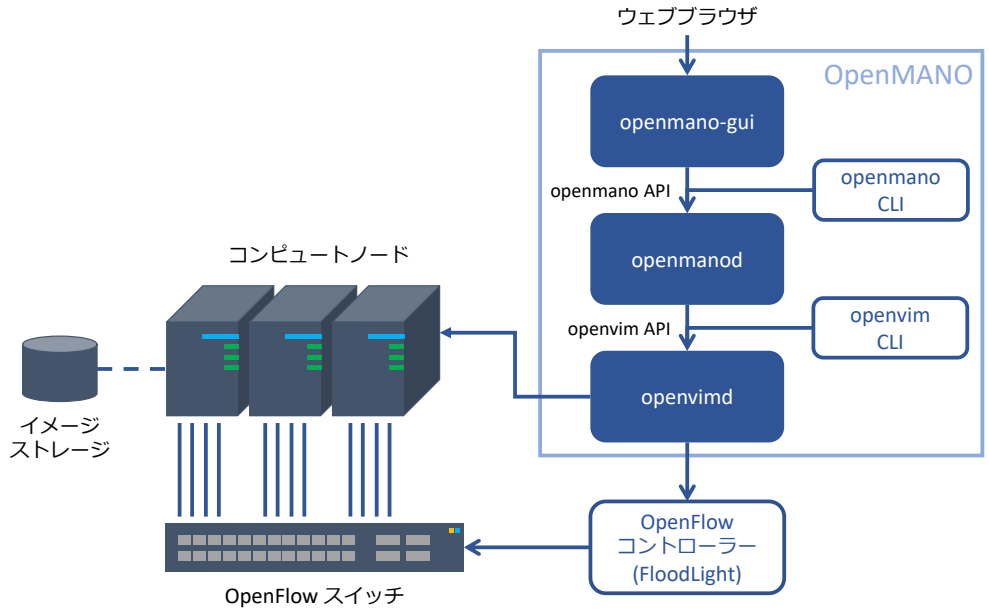

図 5: OpenMANO のアーキテクチャ

- コンピュートノード

  コンピュートノードは、図 1 に示した NFV フレームワークにおける NFVI に相当し、実際に VNF を実行する仮想マシンが展開される物理サーバーである。コンピュートノードは複数台使用することができ、各コンピュートノードはコントロールプレーンに加え、データプレーンにより OpenFlow スイッチを介して相互に接続される。これにより、各コンピュートノード上で展開する仮想マシン間の接続、通信を可能とし、VNF 間でデータのやりとりが行える。

これらの OpenMANO を用いたネットワーク機能仮想化環境の構築に利用する機器のハードウェアおよびソフトウェアの要件は 3.2 節で述べる。以降では、OpenMANO の 3 つの主要モジュールについて、図 1 に示した NFV フレームワークとの関係性も含めて、図 6 を用いて説明する [9, 16]。

openvim

openvim モジュールは、図 1 に示した NFV フレームワークの VIM に相当する。OpenMANO を用いたネットワーク機能仮想化環境では、ハードウェア資源として用いられる物理サーバーは、1 台のコンピュートノードとして扱われる。openvim は、コンピュートノード

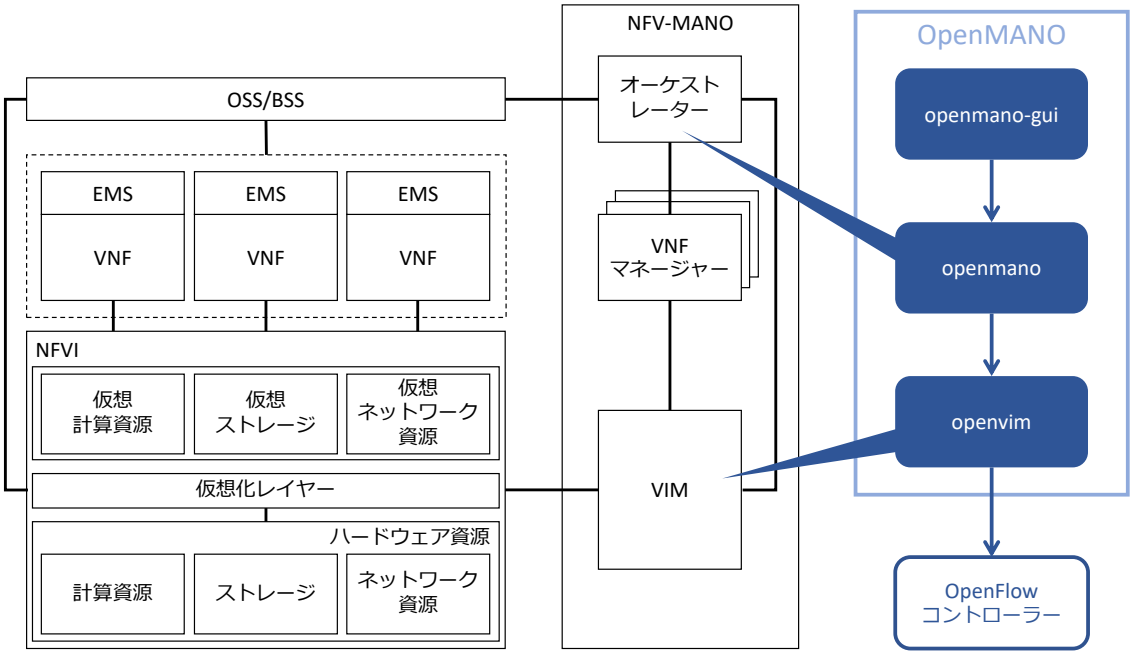

図 6: OpenMANO の主要モジュールと NFV フレームワークの関係

と OpenFlow スイッチ、OpenFlow コントローラーなどの外部コンポーネントが接続され、これらを管理する。コンピュートノードは複数台登録できるが、OpenFlow スイッチは 1 台のみの登録となる。仮想マシンを展開する際の資源の割り当てや、解放された資源の再利用を行う。各コンピュートノードおよび各仮想マシンの動作状態の情報は収集しているが、各コンピュートノードのハードウェア資源および各仮想マシンに割り当てられた仮想資源の利用率や負荷状況のモニタリングは行なっていない。クラウド環境構築用ソフトウェアである OpenStack との類似点が多く、イメージ、フレーバー、インスタンス、ネットワークの追加、削除、管理も行える。コマンドラインクライアントから操作を行う。

openmano

openmano モジュールは、図 1 に示した NFV フレームワークのオーケストレーターに該当する。OpenMANO 上では、各 VNF を結んだ仮想ネットワークは Scenario と呼ばれ、openmano モジュールは、VNF と Scenario のテンプレートの追加、編集、削除が行える。これらの、VNF と Scenario の管理に加え、openvim に接続し、それらのインスタンスの展開、削除を指示する。ただし openvim が、登録されたハードウェア資源とその空き状況の管理は行う一方で、資源の利用率や負荷状況のモニタリングを行わないため、インスタンス

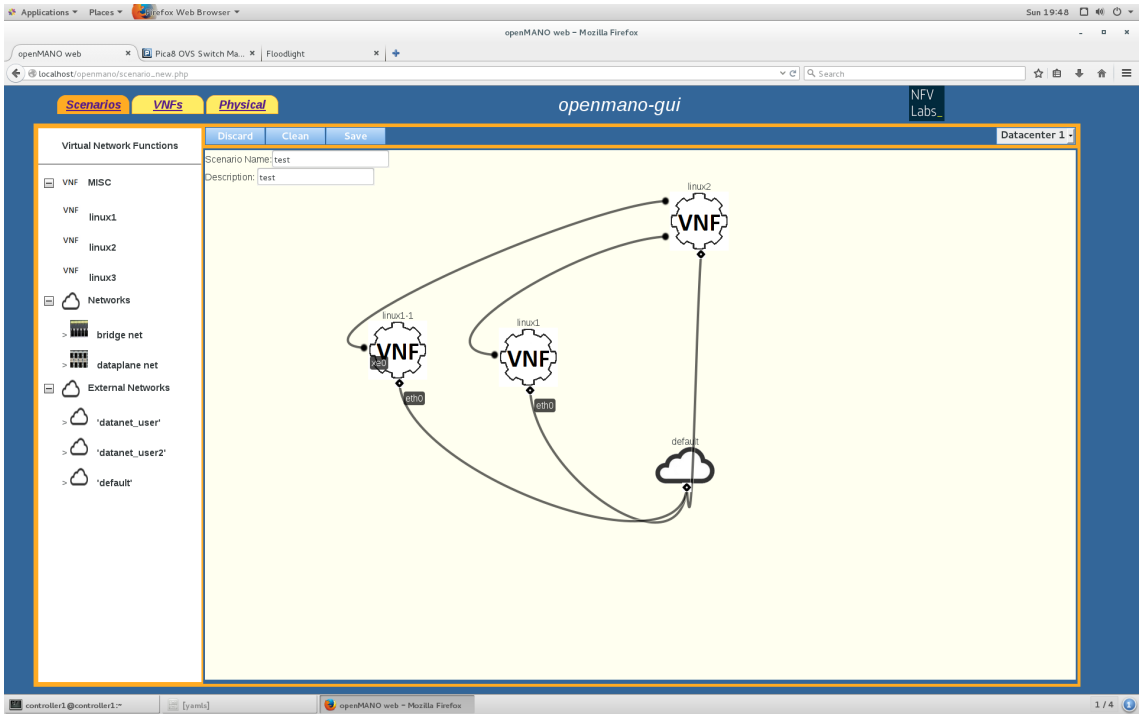

図 7: OpenMANO を用いた仮想ネットワークの作成方法

展開時には、各コンピュートノードにバランスよく仮想マシンが配置されるが、その後の環境変化によって仮想マシンの他のコンピュートノード上への移行や、仮想資源の増減などの管理、調整は行われない。コマンドラインクライアントから操作を行う他、次で説明する openmano-gui からも操作できる。

openmano-gui

openmano-gui モジュールを利用することで、ネットワーク管理者は、登録された VNF と Scenario の管理、Scenario の作成、インスタンスの展開、インスタンスの状態確認などの操作がウェブブラウザ上の GUI 環境で行える。図 7 に、OpenMANO を用いた仮想ネットワークの作成方法として、実際に構築した環境上で動作する openmano-gui にウェブブラウザからアクセスし、Scenario の作成を行なっている様子を示す。画面左部に並ぶ VNF をドラッグ＆ドロップで中央部に配置し、各 VNF の仮想ネットワークインターフェースを表す記号同士を、リンクを表す線で結ぶことができる。

3.2　ハードウェアおよびソフトウェアの要件

OpenMANO を用いたネットワーク機能仮想化環境の構築には、同じ LAN 内にあるコントローラーノードとコンピュートノード、OpenFlow スイッチが必要である。

コントローラーノード

コントローラーノードは、専用のハードウェアを用意せず仮想マシンを利用しても良いため、ハードウェアの要件は特に存在しない。ソフトウェア要件として、OS が 64 ビットの Linux である必要がある。開発側で、Ubuntu Server 14.04 LTS/Desktop 14.04.2 LTS、CentOS 7 上で動作することが確認されている。

コンピュートノード

コンピュートノードは、Intel の Ivy Bridge アーキテクチャXeon E5 プロセッサーを搭載したハードウェアが推奨される。開発元による動作確認はされていないが、Intel Core i3、i5 、 i7 シリーズでも動作すると考えられている。ネットワークインターフェースは、Intel X520 などの、SR-IOV (Single Root I/O Virtualization) 対応 10 ギガビットイーサネットインターフェースであることが強く推奨される。また、コンピュートノードは、仮想化ソフトウェアとして、KVM (Kernel-based Virtual Machine) と QEMU、libvirt がインストールされた RedHat 7、CentOS 7、Ubuntu Server 14.04 などの 64 ビット Linux 環境が必要である。SR-IOV および、これらの仮想化ソフトウェアについては、3.3 節で詳しく説明する。なお、コンピュートノードが複数台展開する場合は、仮想マシンのディスクイメージを共有しておく必要があるため、NFS (Network File System) を用いて共有のイメージフォルダを作成する。この際は、外部の NAS に仮想マシンのディスクイメージを格納し、各コンピュートノードは、クライアントとしてそれらを利用する方法が信頼性の面から推奨される。

OpenFlow スイッチ

Openflow スイッチは、コンピュートノードと接続するための 10 ギガビットイーサネットインターフェースを持ったものが、1 台必要である。

3.3 サーバー仮想化環境

コンピュートノードは、図1に示したNFVフレームワークにおけるNFVIに相当し、ハードウェア資源を仮想化する仮想化ソフトウェアが必要である。コンピュートノードの仮想化に利用した仮想化ソフトウェアは、KVMとQEMU、libvirtである。これらの仮想化ソフトウェアは、NFVIにおける仮想化レイヤーと考えることができる。KVMは、Linuxカーネル仮想化基盤と呼ばれ、Linuxカーネル自体をハイパーバイザーとして利用する。Intel VT (Virtualization Technology) やAMD-V (AMD-Virtualization) などに代表される仮想支援機構が搭載されたハードウェア上で動作するLinuxにおいて仮想化環境を実現する。しかし、KVMは周辺機器のエミュレートを行わないため、これらの機能をQEMUで補い仮想マシンへ提供する。QEMUは単体で仮想化環境の構築が可能だが、より高速な動作を可能とするKVMと合わせて利用される。libvirtは、仮想マシンを特定のハイパーバイザーに依存せず管理可能にするAPIを提供するライブラリである [17]。KVMやXenなどの幅広いハイパーバイザーをサポートしており、デファクトスタンダードの地位を築きつつある。

また、2.1節で述べたように、ISGが定めたNFVの要件では、利用するハードウェアの物理的特性による制約を避け、仮想化による柔軟性および拡張性を確保するため、標準的なハードウェアの仮想化が推奨されている。従来、このような仮想化による柔軟性および拡張性を確保し、1つのネットワークインターフェースを複数の仮想マシンで共有するためには、仮想マシンとネットワークインターフェースのやり取りをハイパーバイザーが調停し、1つのネットワークインターフェースに対する通信の一本化、複数の仮想マシンに対する通信の分割を行なう必要があった。しかし、この方法では、ソフトウェア処理がボトルネックとなり得る上に、仮想マシンによるネットワークインターフェースへのアクセスが集中すると、その処理に物理サーバーの多くの計算資源が割かれ、物理サーバー全体のパフォーマンス低下に繋がるという問題点があった。その一方で、仮想マシンが高いパフォーマンスでハードウェア資源を利用する仕組みとして、PCIパススルーが用いられてきた。しかし、PCIパススルーでは、1つの仮想マシンが1つのハードウェア資源を占有するという短所がある。OpenMANOは、パフォーマンスと柔軟性、拡張性の両方を改善し、ホストとなる物理サーバーのハードウェアを利用する方法であるSR-IOVを、コンピュートノードの推奨条件として挙げている。SR-IOVは、NFVIとして使用する汎用サーバーの物理ネットワークインターフェースの技術規格の1つであり [18]、PCIデバイス側で仮想化をサポートする技術規格である。SR-IOV対応のネットワークインターフェースでは、ハイパーバイザーによるソフトウェア処理を介さず、複数の仮想マシンとのやり取りをハードウェア処理するため、1つのネットワークインターフェースを複数の仮想マシンが高速で共用できる [19]。

4　サービス機能の再配置がユーザーの通信品質に与える効果の評価

本章では、OpenMANO を用いて構築したネットワーク仮想化環境上の、サービスアプリケーションについて説明し、サービス機能の再配置がユーザーの通信品質に与える効果の評価を行うための実験環境ならびに、その結果、評価を述べる。

4.1　サービスアプリケーション

近年登場した新しいアプリケーションやサービスとして、Pepper を用いた買い物代行サービスを実現することを考えた。買い物代行サービスでは、Pepper が実店舗に赴き、ユーザーは自宅にいながら買い物を楽しめる。さらに Pepper が取得した実世界映像に、2.3 節で紹介した AR 技術を用いて、商品情報を重ね合わせ、ユーザーに提示する。この処理を、エッジサーバーで行い、商品情報はクラウドから取得する。また、センシング技術を活用して、商品の触感やにおいなどをユーザーへ提示し、ユーザーの意図理解によるロボット制御も行う。ただし本報告では、Pepper からの映像のライブストリーミング配信と、ユーザーの音声の Pepper による発話のみを想定する。買い物代行サービスを実現するアプリケーションの処理内容を図 8 に示し、本報告における処理内容を図 9 に示す。

なお、Pepper とは、ユーザーの感情認識が可能な人型ロボットである。SDK (Software Development Kit) が配布されており、搭載するカメラやセンサー、動作モジュールを活用した Pepper 上で動作するアプリケーションの開発および、外部で動作するアプリケーションの開発、Pepper との連携が、API (Application Programming Interface) を利用し高い自由度で行うことができる。本報告では、ネットワーク機能仮想化環境でアプリケーションやサービスを柔軟に提供した際に生じる遅延を計測するとともに、将来的にはアプリケーションレベルの品質への影響を調査することを目的としている。Pepper を用いることで、搭載するカメラやセンサー、動作モジュールを活用したアプリケーションの構築が可能となり、実機を用いたサービスが実現され、これにより、ユーザー体験にもとづくアプリケーションレベルの品質の測定が可能となる。

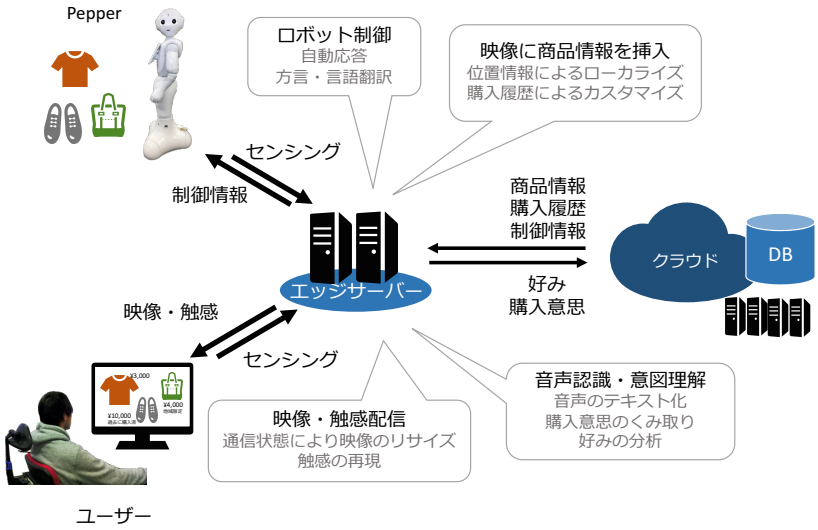

図 8: 想定するサービスアプリケーションの処理内容

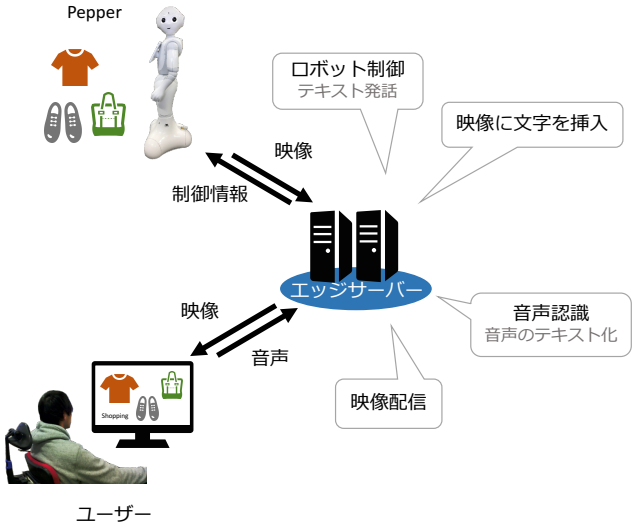

図 9: 想定するサービスアプリケーションの本報告における処理内容

4.2 実験環境

本節では、実験を行うために構築した環境について、まず上記サービスの実現に用いたアプリケーションについて説明し、実験用のネットワーク構成について述べる。最後に、実験を行うにあたり設定した、想定するサービス提供携帯と、それに伴うサービス機能の配置を変えた 5 つのシナリオについて説明する。

4.2.1 サービスの実現に用いたアプリケーション

Pepper は、搭載するカメラやセンサー、動作モジュールを活用した Pepper 上で動作するアプリケーションの開発および、Pepper 外部で動作するアプリケーションの開発、Pepper との連携が、API を利用し高い自由度で行うことができる。そこで、本報告では、主に ffmpeg [20] と Julius [21] の 2 つの外部アプリケーションと Pepper を連携し、本報告の想定するサービスを実現する。Pepper からユーザー方向のライブストリーミングと、ユーザーから Pepper 方向のユーザーの音声の Pepper による発話は、それぞれ ffmpeg、Julius が外部のサーバー上で処理を行う。ただし、Julius は API を用いて Pepper と連携する一方で、ffmpeg は、Pepper の OS である NAOqi (Gentoo Linux ベース) 上で動作する Pepper 内部の ffmpeg と直接連携する。また、想定するサービスをユーザーが利用するために、ユーザー PC において、動画を再生するメディアプレーヤーと Julius に音声を送信するアプリケーションが必要である。これらには、それぞれ ffplay と adintool を利用する。

以下では、ここまで述べてきたアプリケーションに加え、ストリーミングサーバーアプリケーション ffserver と、Julius と Pepper を連携させる上で自作したプログラム Julius2Pepper について、それらを利用しデータが処理される過程を示す図 10 を用いて説明する。

ffmpeg: 音声や映像の記録、変換およびストリーミングを行うクロスプラットフォームのフリーソフトウェア。Pepper の OS、NAOqi に標準搭載されているが、初期状態ではネットワーク経由のストリーミング機能が無効化されているため、有効化したものと置き換えて利用する。また、回転、リサイズ、映像への文字の挿入、コントラストなどの調整といった加工も可能であり、実際にサーバー上の ffmpeg (1) は映像に文字列の挿入を行う。ffmpeg (2) は ffserver へデータを中継する。ffmpeg から送信するデータはすべて、データリンク層プロトコルとして UDP を用いる。

ffserver: ffmpeg に付属する HTTP ベースのストリーミングサーバーである。MPEG、MJPEG、Flash など様々なストリーミング形式に対応している。映像の配信には、データリンク層プロトコルとして TCP を用いる。

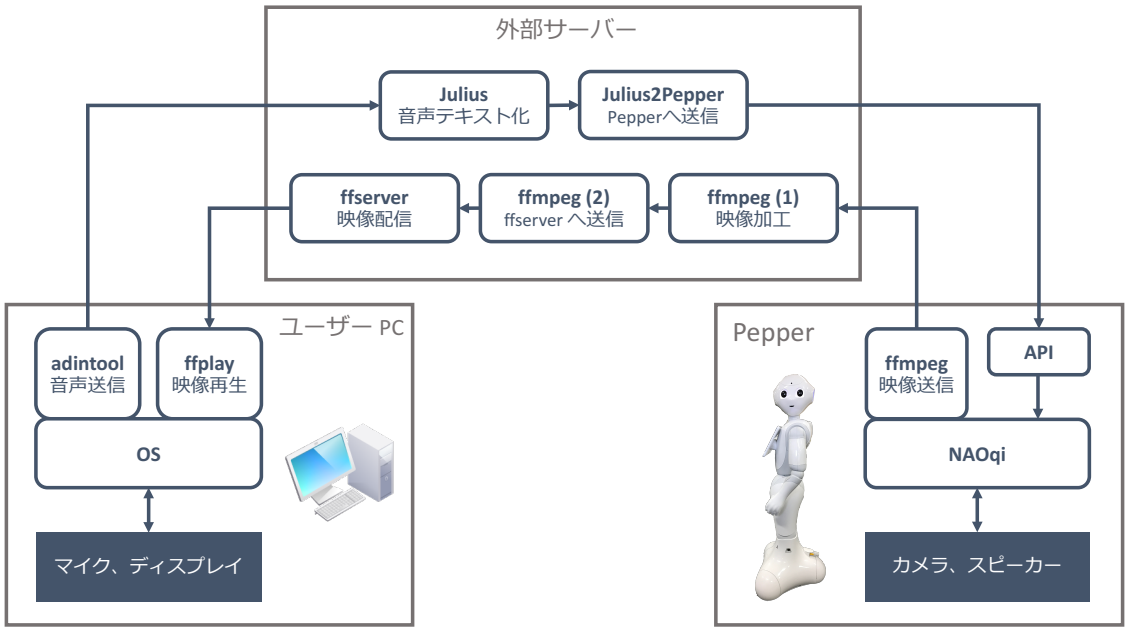

図 10: 想定するサービスアプリケーションにおけるデータの処理過程

ffplay: ライブストリーミングされる映像の再生に用いるメディアプレーヤーは、ネットワーク経由のストリーミング配信映像の再生が可能な、オープンソースのメディアプレイヤーを想定している。本報告の実験環境では、ffmpeg に付属する ffplay を用いるが、VLC Media Player の動作確認もしている。また、これらのメディアプレーヤーは、配信される映像を運ぶパケットが損失した場合においても映像の品質を確保するため、ストリーミング映像をバッファリングする機能を有する。しかしバッファリングによる映像の遅延を最小限に抑えるため、ffplay のバッファ容量を最小に設定する。

Julius: 音声認識システムの開発・研究のためのオープンソースの高性能な汎用大語彙連続音声認識エンジン。京都大学と名古屋工業大学の研究チームによって開発されており、公式サイト [21] によれば「数万語彙の連続音声認識を一般の PC やスマートフォン上でほぼ実時間で実行できる軽量さとコンパクトさを持つ」とされている。ネットワーク経由の音声入力に対応しており、さらにモジュールモードで起動することでソケット通信による出力が可能である。Julius の動作には、音響モデル、単語辞書、言語モデルが必要であるが、Julius 公式サイトで提供されている日本語ディクテーションキットに含まれる最小限のものを利用する。

adintool: Julius に含まれる音声波形データの記録・分割・送信・受信ツール。ユーザー PC

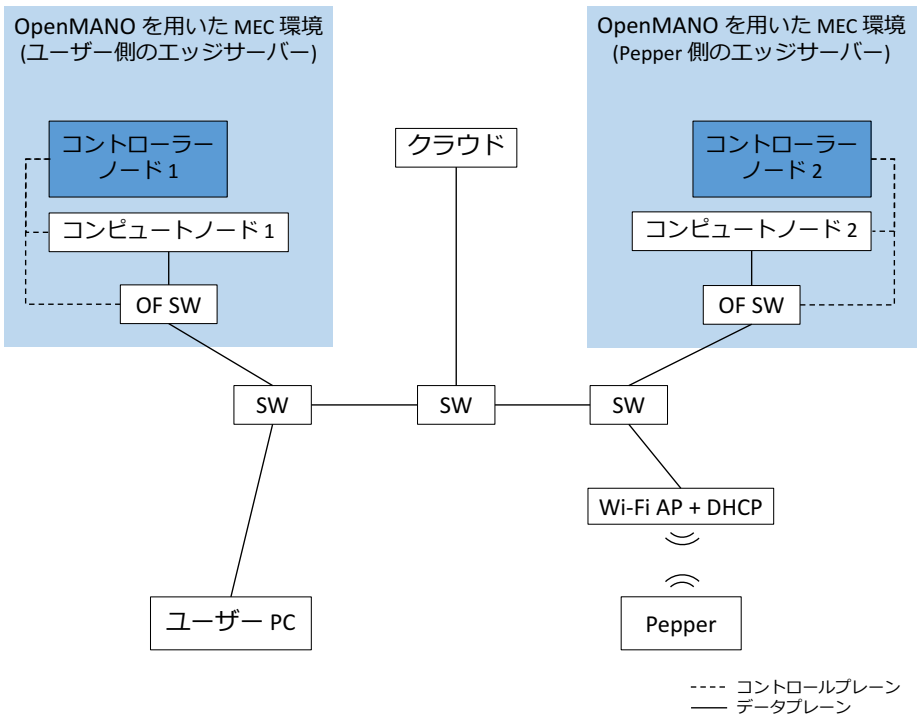

図 11: 実験用ネットワーク構成

で取得した音声をサーバー上で起動している Julius に送信するために用いる。

Julius2Pepper: 自作した Python プログラム。モジュールモードで起動した Julius の出力はテキスト化された音声認識の結果を含む XML 形式であり、そこから Pepper に発話させるテキストのみを取り出し、Pepper に送信する必要がある。そこでこのプログラムでは、モジュールモードで起動している Julius にソケット通信で接続したのち、XML 形式の出力を受信、解析し、所望のテキストを抽出する。さらに、Pepper の API を利用してこのテキストを Pepper に送信し発話させる。

4.2.2 ネットワーク構成

ユーザー PC および、Pepper、3 台の計算機をスイッチで接続し、実験室内にモバイルエッジコンピューティング環境を構築した。実験を行うネットワークの構成を図 11 に示す。図中の OP SW は OpenFlow スイッチを、SW は標準的なネットワークスイッチを、Wi-Fi AP + DHCP は DHCP サーバー機能を持った Wi-Fi アクセスポイントであることを表す。3 台の計算機は、コンピュートノード 1 およびコンピュートノード 2 と、クラウドを介して

表 1: 実験で用いる物理サーバーと仮想マシンの諸元

	物理サーバー	仮想マシン
CPU コア数	28	2
メインメモリ	64 GB	4 GB
ストレージ	720 GB	20 GB
ネットワークインターフェースの帯域幅	10 Gbps	10 Gbps
OS	CentOS 7	CentOS 7

接続されるノードとして利用する。図に示す「OpenMANO を用いた MEC 環境」はエッジサーバーを想定している。OpenMANO を用いてモバイルエッジコンピューティング環境が構築されており、4.2.1 節で説明した各アプリケーションはコンピュートノード上の個別の仮想マシンで実行される。買い物代行サービスでは、Pepper とユーザーが地理的に離れるため、それぞれに近い処理拠点として、2 つのエッジサーバーを用意する。また、クラウドはモバイルエッジコンピューティングにおいてネットワーク中心部にあるデータセンターを想定したサーバーである。このサーバーは仮想化環境は構築されておらず、アプリケーションはサーバー上で直接実行される。ただし、このネットワーク構成では、エンド端末であるユーザー PC および Pepper と、クラウドが地理的に離れていることが再現できていない。そのため、実験結果の評価を行う際は、地理的な要因によって発生する遅延を考慮する必要がある。コンピュートノード上のすべての仮想マシンで動作するアプリケーションと、クラウドを想定したサーバー上で実行されるアプリケーションは、図中で実線で描かれたデータプレーンを利用して通信を行う。Pepper は、図中で Wi-Fi AP で示す Wi-Fi アクセスポイントを介して、無線でデータプレーンに接続している。なお、コンピュートノードとクラウドを想定したサーバーとして利用したのは、同じモデルの同性能の物理サーバーである。しかし、コンピュートノード上で展開する仮想マシンは、物理サーバーのハードウェア資源を複数の仮想マシンで共有するため、必然的に物理マシンよりもハードウェア性能が抑えられる。表 1 に、実験に用いた物理サーバーと展開する仮想マシンの諸元を示す。なお、3.1 節で述べた通り、コントローラーノードは OpenMANO のモジュールが展開し、コントロールプレーン上で外部コンポーネントとやり取りする。これらは、データプレーン上の通信には一切関与しない。

4.2.3 シナリオ

また、本報告の実験では、仮想化環境の有無や、TCP 通信の距離など、想定するサービス提供携帯と、それに伴うサービス機能の配置を変えた 5 つのシナリオを用意した。以下で、5 つのシナリオの説明を行う。また、各シナリオにおける、サービスアプリケーションが実行される実験用ネットワーク上の位置、およびそれにより生じる通信経路のパターンを図 12 から図 16 に示す。なお、本報告の実験環境では、Pepper の ffmpeg と ffserver までは UDP 通信を、ffserver と ffplay 間は TCP 通信を利用しており、一般的に通信速度に差があると言われる UDP 通信と TCP 通信の通信距離の差によるライブストリーミング映像の遅延時間の比較を行った。

MEC シナリオ: MEC シナリオとして、エッジサーバーの利用を想定してサービスを提供する 3 つのシナリオを用意した。

シナリオ NFV1U: OpenMANO を用いたモバイルエッジコンピューティング環境が構築された 2 つのコンピュートノードをエッジサーバーとし、ユーザー側で展開する拠点に映像加工と配信処理を行うアプリケーションを配置しサービスを提供する (図 12)。実験のシナリオを示した図 12 における「CN1」がこのサーバーである。仮想マシン上でアプリケーションを実行する。また、このサーバーにサービスアプリケーションを配置することで、TCP 通信の通信距離を小さくしている。

シナリオ NFV1P: OpenMANO を用いたモバイルエッジコンピューティング環境が構築された 2 つのコンピュートノードをエッジサーバーとし、Pepper 側で展開する拠点に映像加工と配信処理を行うアプリケーションを配置しサービスを提供する (図 13)。実験のシナリオを示した図 13 における「CN2」がこのサーバーである。仮想マシン上でアプリケーションを実行する。また、このサーバーにサービスアプリケーションを配置することで、TCP 通信の通信距離を大きくしている。

シナリオ NFV2: OpenMANO を用いたモバイルエッジコンピューティング環境が構築された 2 つのコンピュートノードをエッジサーバーとし、ユーザー側で展開する拠点に配信処理を行うアプリケーションを、Pepper 側で展開する拠点に映像加工を行うアプリケーションを、それぞれ分散して配置しサービスを提供する (図 14)。実験のシナリオを示した図 14 における「CN1」および「CN2」がこれらのサーバーである。仮想マシン上でアプリケーションを実行する。負荷の分散を図っており、TCP 通信の通信距離はシナリオ NFV1U と同じ条件である。

非 MEC シナリオ: 非 MEC シナリオとして、クラウドの利用を想定してサービスを提供する 1 つのシナリオを用意した。

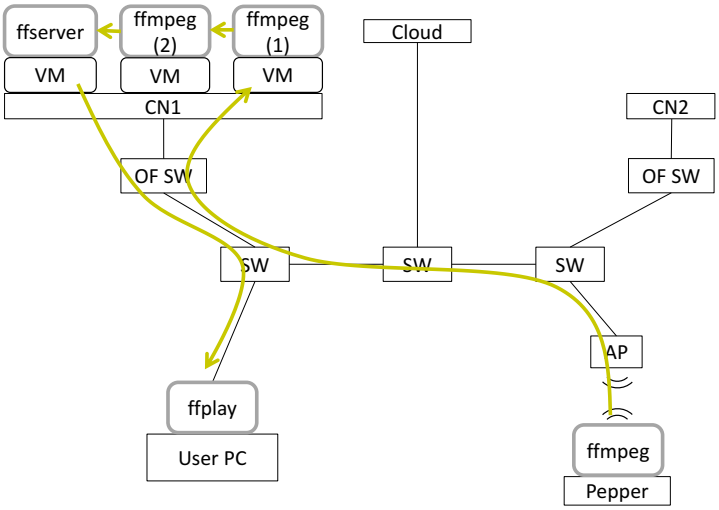

図 12: 実験用ネットワーク上のアプリケーション配置と通信経路: シナリオ NFV1U

シナリオ Cloud: モバイルエッジコンピューティングにおいてネットワーク中心部にあるデータセンターを想定したサーバーに、映像加工と配信処理を行うアプリケーションを配置しサービスを提供する (図 15)。図中の「Cloud」は、図 11 中のクラウドに該当する。アプリケーションはこのサーバー上で仮想化を行うことなく実行される。

比較シナリオ:

シナリオ Direct: 映像加工と配信処理を行うアプリケーションを配置せず、Pepper の ffmpeg からユーザー PC のメディアプレーヤー ffplay に直接映像を送信する (図 16)。エンド端末の処理時間を計測することを目的とする。

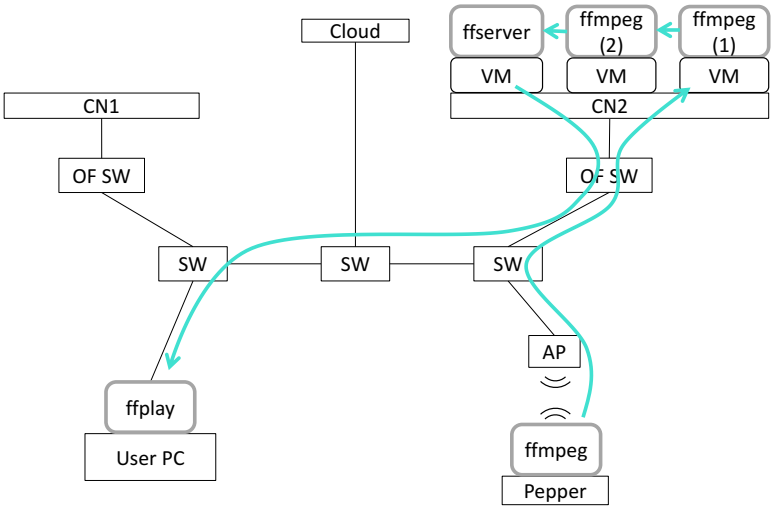

図 13: 実験用ネットワーク上のアプリケーション配置と通信経路: シナリオ NFV1P

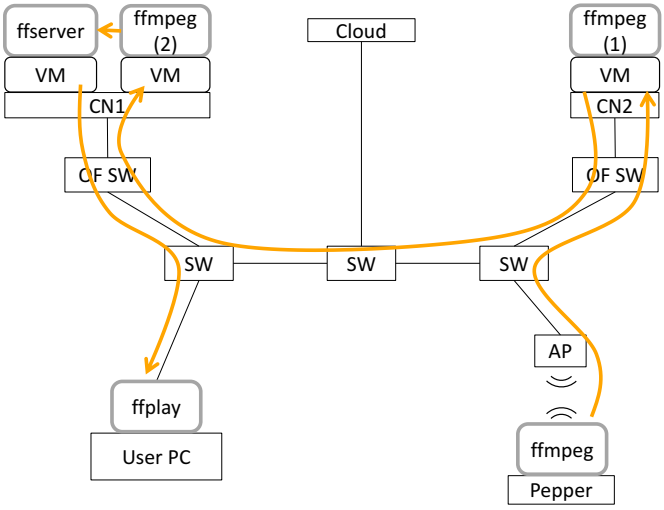

図 14: 実験用ネットワーク上のアプリケーション配置と通信経路: シナリオ NFV2

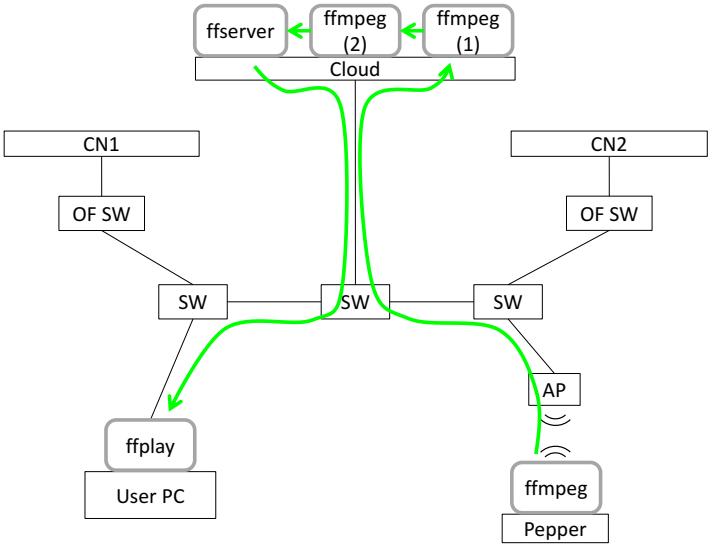

図 15: 実験用ネットワーク上のアプリケーション配置と通信経路: シナリオ Cloud

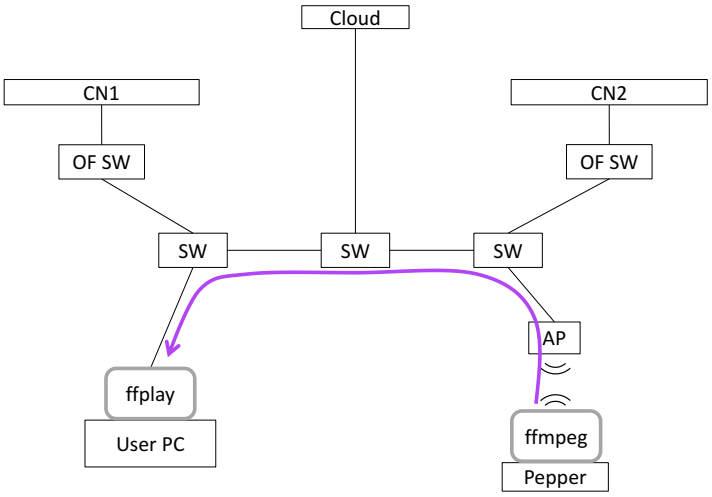

図 16: 実験用ネットワーク上の通信経路: シナリオ Direct

表 2: ライブストリーミング映像の遅延時間

シナリオ	遅延時間 [ms]		
	1 回目	2 回目	平均
NFV1U	471.574	463.396	467.485
NFV1P	476.178	476.574	476.376
NFV2	465.683	465.307	465.495
Cloud	460.564	459.238	459.901
Direct	429.950	423.436	426.693

4.3 実験結果と評価

ネットワーク仮想化にもとづくサービス機能の再配置がユーザーの通信品質に与える効果を評価するにあたり、ライブストリーミング映像のエンド端末間における遅延に着目し、4.2 節で記載した実験環境において、ライブストリーミング映像の遅延時間を計測する実験を行なった。加えて、各サーバー (クラウドとコンピュートノード) の処理時間を計測する実験を行なった。本節では、実験の方法および実験結果と、ネットワーク仮想化にもとづくサービス機能の再配置がユーザーの通信品質に与える効果を評価する。

4.3.1 ライブストリーミング映像の遅延時間

本報告では、OpenMANO を用いたモバイルエッジコンピューティング環境を実機を用いて構築し、モバイルエッジコンピューティングの考え方のもとでサービス機能を再配置した際に生じる、ユーザーの通信品質に与える効果を調べるため、4.2 節で記載した実験環境において、ライブストリーミング映像の遅延時間を求めた。その方法として、まず、Pepper の前でミリ秒単位で表示されるデジタル時計を表示し、その映像をライブストリーミングし、ユーザー PC に表示する。ここで Pepper に見せるデジタル時計は時刻同期の観点からユーザー PC の画面に表示する。次に、このデジタル時計と Pepper からのライブストリーミング映像をデスクトップ上に並べ、スクリーンショットを撮影する。これを、1 秒おきに 100 回繰り返す。最後に、100 枚のスクリーンショットすべてについて、撮影された 2 つの時刻の差を求め、平均遅延時間を算出する。図 17 に測定方法を示す。また、図 18 に示すスクリーンショットでは、左側の赤枠内の時刻が実世界時刻、右側の赤枠内の時刻がストリーミング配信された映像内の時刻である。ストリーミング配信された映像が示す時刻は、実世界時刻より遅れていることがわかる。この方法で、ライブストリーミング映像の遅延時間を、5 つ

のシナリオでそれぞれ 2 回測定した結果とその平均を表 2 に示す。結果から、映像加工と配信処理を行わないシナリオ Direct における遅延時間を、エンド端末の処理時間とみなすことができる。また、仮想マシン上で処理を行うシナリオ NFV1U において、仮想化が行われていないサーバーで処理を行うシナリオ Cloud よりも、平均でおよそ 33 [ms] 遅延が大きくなっていることが読み取れ、これは映像の処理と配信にかかる時間だと考えられる。仮想化による遅延増大は、シナリオ NFV1U、Cloud の比較から、平均でおよそ 16 [ms] であることがわかる。また、シナリオ NFV1U、NFV1P の比較により、TCP 通信の通信距離を削減することで、遅延が平均で約 9 [ms] 低減されたことがわかる。シナリオ NFV1U、NFV2 の結果の平均はほぼ変わりがなく、処理の分散による遅延低減は観測されなかった。

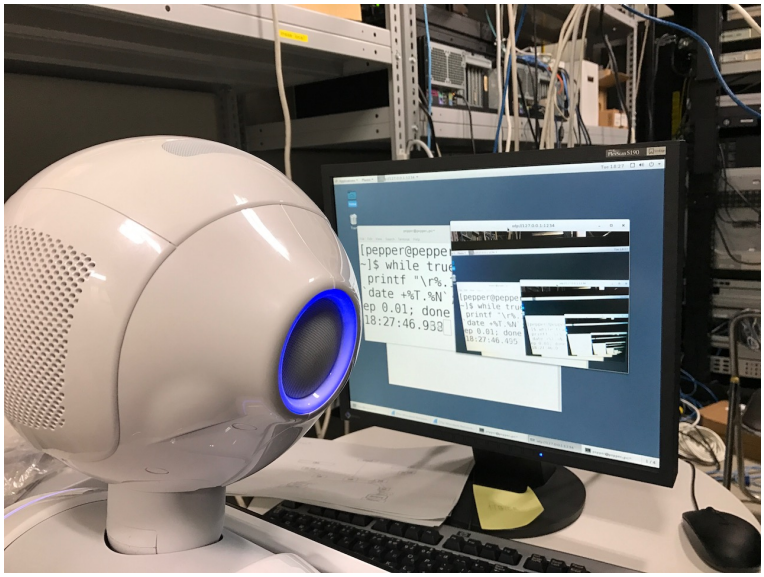

図 17: ライブストリーミング映像のエンド間の遅延計測の方法

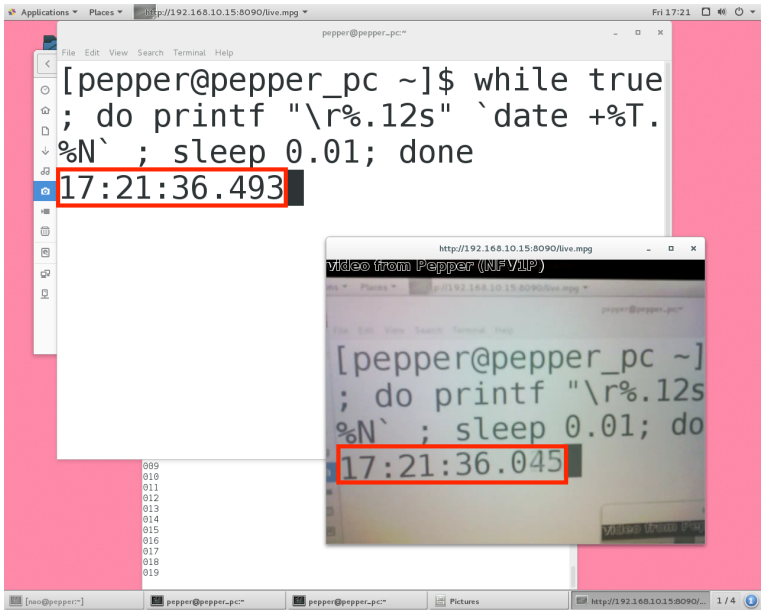

図 18: ライブストリーミング映像のエンド間の遅延計測で撮影したスクリーンショット

4.3.2 サーバー処理時間

アプリケーションの処理にかかるサーバー処理時間を調べ、ライブストリーミング映像の遅延時間に占める割合を明らかにするとともに、仮想化環境の有無によるサーバー処理時間の差を確認する必要がある。そのため、シナリオ NFV1U、NFV1P、NFV2、Cloud において、各アプリケーションのサーバー処理時間を計測する実験を行った。サーバ処理時間を直接計測することは困難であるため、各シナリオの通信経路において、サーバーの出入口となる地点でパケットキャプチャを行い、出入りするパケットの時差を求めた。ただし、サーバーへ到達したパケットは、処理を経て別のパケットとしてサーバーから送信されるため、同じデータを運ぶパケットを双方向で特定することは難しい。そのため、ライブストリーミングの開始・停止を繰り返し、開始時に流れるパケット p_first と最後に流れるパケット p_last について、サーバーの出入口となる地点でパケットの通過時刻の差を調べた。

まず、図 19 に示すシナリオ Cloud において、アプリケーションのサーバー処理時間を調べた。サーバーの出入口となる地点 C のネットワークインターフェースでパケットキャプチャを行なった。ライブストリーミングの開始・停止を 10 回行い、合計 20 個のパケットの平均時差を求めた結果を表 3 に示す。表中の空欄は、パケットの時差が負の値になり、不正な値が計測されたと判断したため、結果から除外している。結果から、地点 C を出入りしたパケットの時差は平均 7.320 [ms] であり、仮想化が行われていないサーバーの処理時間とみなすことができる。

次に、図 20 に示すシナリオ NFV1P において、アプリケーションのサーバー処理時間を調べた。コンピュートノードは 3.3 節で述べた SR-IOV を利用しており、ホストとなる物理サーバーのネットワークインターフェースでキャプチャしても、仮想マシン宛てのパケットはキャプチャされない。また、仮想マシン上で個別にパケットキャプチャを行うことは可能だが、同じ物理サーバー上で動作する複数の仮想マシンの処理時間の合計が、仮想化の有無の比較において意味を持つため、コンピュートノードの出入り口となる地点 D の OpenFlow スイッチでポートミラーリングを行い、他の PC で受信してパケットキャプチャを行なった。ライブストリーミングの開始・停止を 10 回行い、合計 20 個のパケットの平均時差を求めた結果を表 4 に示す。結果から、地点 D を出入りしたパケットの時差は平均 12.611 [ms] であり、仮想化されたサーバーの処理時間の 1 つとみなすことができる。

続いて、図 21 に示すシナリオ NFV1P と同様の方法で、シナリオ NFV1U において、アプリケーションのサーバー処理時間を調べた。コンピュートノードの出入り口となる地点 E の OpenFlow スイッチでポートミラーリングを行い、他の PC で受信してパケットキャプチャを行なった。ライブストリーミングの開始・停止を 10 回行い、合計 20 個のパケットの平均時差を求めた結果を表 5 に示す。結果から、地点 E を出入りしたパケットの時差は平

均 11.727 [ms] であり、この値も仮想化されたサーバーの処理時間の 1 つとみなすことができる。

最後に、図 22 に示すシナリオ NFV2 において、シナリオ NFV1P と NFV1U と同様の方法で、アプリケーションのサーバー処理時間を調べた。各コンピュートノードの出入り口となる地点 F と地点 G の OpenFlow スイッチでポートミラーリングを行い、他の PC で受信してパケットキャプチャを行なった。ライブストリーミングの開始・停止を 10 回行い、合計 20 個のパケットの平均時差を求めた結果を表 6 に示す。結果から、仮想マシンを分散して 2 つのコンピュートノードで処理したシナリオ NFV2 においては地点 F、地点 G でそれぞれ平均 6.658 [ms]、5.651[ms] のパケットの時差があることがわかり、合算すると 12.309[ms] になる。これは、同量の処理を同一サーバ上の仮想マシンで行ったシナリオ NFV1P、NFV1U と同等の値である。

以上から、仮想化によるサーバー処理時間の増大が、およそ 5 [ms] であることがわかる。4.3.1 節の結果から、仮想化によるライブストリーミング映像の遅延時間増大が 16 [ms] であることがわかっており、残りの 11 [ms] は仮想化によるプロトコルオーバーヘッドの増大だと考えられる。また、負荷の分散によるサーバー処理時間の低減は観測されなかった。

表 3: パケットキャプチャ地点の通過時刻の差: シナリオ Cloud、サーバー処理時間

ストリーミング回数	通過時刻の差 [ms]	
	p_first	p_last
1 回目	8.855	5.534
2 回目	8.944	6.681
3 回目	7.927	6.609
4 回目	9.007	5.682
5 回目	7.926	5.281
6 回目	8.611	6.095
7 回目	9.347	5.503
8 回目	8.461	5.683
9 回目	8.830	
10 回目	8.239	5.858

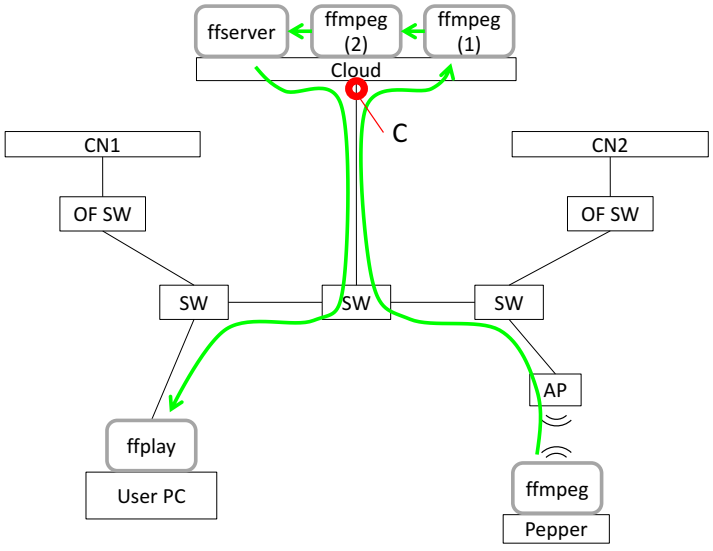

図 19: パケットキャプチャ地点: シナリオ Cloud、サーバー処理時間

表 4: パケットキャプチャ地点の通過時刻の差: シナリオ NFV1P、サーバー処理時間

ストリーミング回数	通過時刻の差 [ms]	
	p_first	p_last
1 回目	15.161	10.775
2 回目	12.909	9.651
3 回目	13.708	10.987
4 回目	15.948	10.254
5 回目	13.700	14.594
6 回目	15.628	10.648
7 回目	14.423	11.239
8 回目	14.113	10.643
9 回目	13.478	11.155
10 回目	13.543	9.666

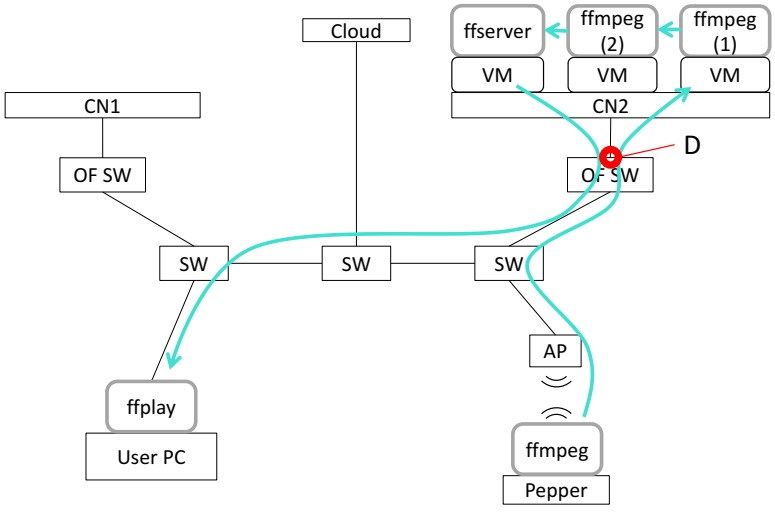

図 20: パケットキャプチャ地点: シナリオ NFV1P、サーバー処理時間

表 5: パケットキャプチャ地点の通過時刻の差: シナリオ NFV1U、サーバー処理時間

ストリーミング回数	通過時刻の差 [ms]	
	p_first	p_last
1 回目	15.606	10.015
2 回目	12.280	10.677
3 回目	12.568	10.379
4 回目	13.360	11.273
5 回目	11.970	10.015
6 回目	13.905	12.065
7 回目	13.058	8.966
8 回目	11.490	10.912
9 回目	12.972	9.432
10 回目	14.258	9.348

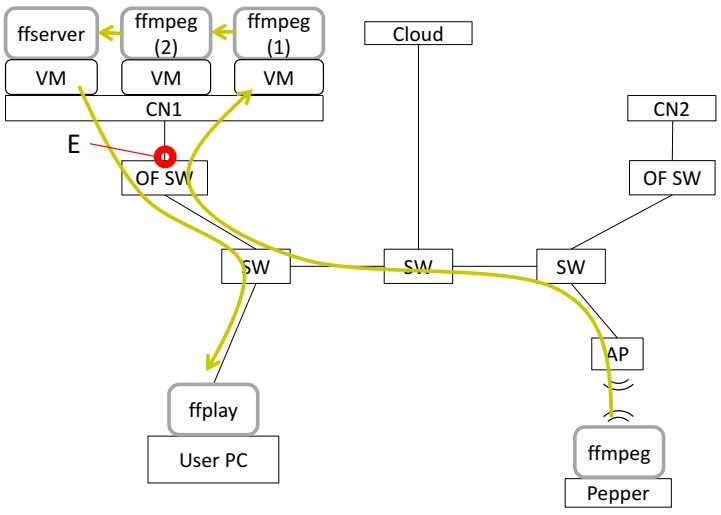

図 21: パケットキャプチャ地点: シナリオ NFV1U、サーバー処理時間

表 6: パケットキャプチャ地点の通過時刻の差: シナリオ NFV2、サーバー処理時間

ストリーミング回数	通過時刻の差 [ms]			
	地点 F		地点 G	
	p_first	p_last	p_first	p_last
1 回目	9.202	5.300	5.899	5.737
2 回目	8.883	4.845	7.364	4.235
3 回目	7.297	6.125	5.762	7.166
4 回目	8.131	5.252	5.978	4.533
5 回目	7.431	4.053	6.034	5.123
6 回目	9.956	5.394	5.386	5.207
7 回目	6.733	4.920	5.384	5.697
8 回目	6.330	5.542	4.681	5.415
9 回目	7.752	4.540	6.371	4.989
10 回目	8.759	6.706	6.715	5.341

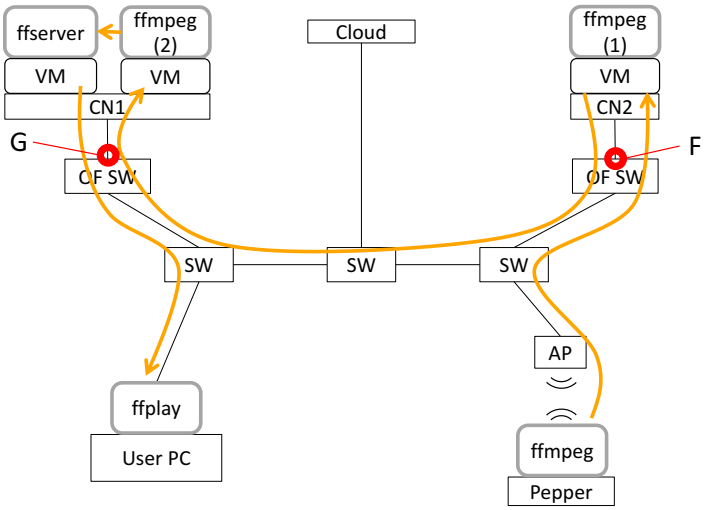

図 22: パケットキャプチャ地点: シナリオ NFV2、サーバー処理時間

4.3.3 評価

本報告では、ネットワーク仮想化にもとづくサービス機能の再配置がユーザーの通信品質に与える効果を評価するにあたり、ライブストリーミング映像の遅延時間に着目し、実験を行ってきた。実験結果から、次のことが言える。

- シナリオ Direct で計測されたライブストリーミング映像の遅延時間の大部分である 420 [ms] 程度の遅延が、通信以外のエンド端末で発生している。映像の再生に用いたメディアプレーヤー ffplay はソースコードが利用可能であるため、処理時間を計測したところ、おおよそ 10 [ms] の処理遅延時間であることがわかった。したがって、420 [ms] の大部分は Pepper 内部で発生しているものと考えられる。

- サーバー処理時間の計測実験において、仮想化が行われていないサーバーでかかる処理時間に比べ、仮想化されたサーバーでかかる処理時間の方が大きいことが確認された。さらに、ライブストリーミング映像の遅延時間の計測結果より、仮想化が行われていないサーバーでアプリケーションを実行する場合に比べ、仮想マシン上で実行する場合の方が、平均遅延時間が 16 [ms] 大きくなったことが読み取れるため、仮想化によるソフトウェア動作に起因する遅延の増大が見られた。

- ライブストリーミング映像の遅延時間の計測結果より、TCP 通信の距離を小さくするようにサービス機能を配置することで遅延が低減される。実験室規模では遅延が 9 [ms] 低減された。

- サーバー処理時間の計測実験において、仮想マシンを分散して 2 つのコンピュートノードで処理したシナリオ NFV2 の場合、同量の処理を同一サーバー上の仮想マシンで行ったシナリオ NFV1P、NFV1U の場合と同等のサーバー処理時間がかかっている。また、シナリオ NFV1U、NFV2 のライブストリーミング映像の遅延時間もほぼ同等であった。そのため、本報告の実験環境において、仮想マシンの分散による処理時間の低減は見られなかった。これは、同一サーバー上の仮想マシンですべての処理を行った場合でも、物理サーバーの計算資源や利用するネットワーク資源の負荷が、処理速度の低下をもたらすほど高まっていなかったと考えられる。そのため、それらの負荷が極限まで高まり、仮想マシンのライブマイグレーションなどによって処理の分散を行なった際に、どのような変化が起きるか調査することが、今後の課題である。

さらに、4.2.2 節で述べたように、本実験のネットワーク構成では、エンド端末であるユーザー PC および Pepper と、クラウドが地理的に離れていることが再現できていない。文献 [22] によれば、一般的にエンド端末からクラウドサービスを提供するデータセンターま

表 7: クラウドサービス宛ての ICMP パケット送信結果

	Amazon	Dropbox
受信パケット数	997	999
パケット損失率 [%]	0.3	0.1
最小往復遅延時間 [ms]	168.964	110.584
平均往復遅延時間 [ms]	207.041	123.616
最大往復遅延時間 [ms]	334.475	390.165
標準偏差	39.191	28.083

での距離によって生じる平均往復遅延は、国内のデータセンターで 100 [ms] 以下、米国のデータセンターで約 100 [ms]、欧州のデータセンターで約 200 [ms] であるとされ、いずれの場合でも百ミリ秒単位で遅延が発生する。実際に、学内のネットワークから、クラウドサービスを提供する Amazon と Dropbox 宛てに、ホスト名を amazon.com と dropbox.com で指定し、ICMP パケットを 1000 回送信したところ、それぞれ平均 207.041 [ms]、123.616 [ms] の往復遅延時間が計測された (表 7)。この結果を、実験のシナリオ Cloud における、ライブストリーミング映像の遅延時間測定の結果と合わせると、クラウドでライブストリーミング映像を処理した場合、映像に最大 660 [ms] 程度の遅延が生じることになる。仮想化が要因となって生じる遅延時間の増大分が 16 [ms] であることを考えると、仮想化が要因となって生じる遅延時間は、地理的な要因によって生じる遅延時間に比べ十分小さいと言える。

以上により、TCP 通信の距離を小さくするようにサービス機能を配置することで、実験室規模でエンド間の遅延を 9 [ms] 低減できることを示した。また、ネットワーク仮想化によるソフトウェア動作から生じる処理速度低下は、地理的な要因によって生じる遅延に比べ十分小さく、クラウドからエッジサーバーにサービス機能を再配置することで、エンド間の遅延を 30%低減できることを示した。したがって、UDP 通信と比較し、通信速度の小さい TCP 通信を利用する通信距離を小さくするようにサービス機能を再配置することや、遠隔地からユーザーに近い拠点にサービス機能を再配置することにより、ユーザーの通信品質を改善できることを、実機を用いて示した。

5　おわりに

モバイルエッジコンピューティングの標準化が進み、導入や展開が現実味を帯びてきている状況において、モバイルエッジコンピューティング環境における通信遅延の発生要因や発生量を理解し、サービス機能の再配置がもたらすユーザーの体験品質に対する効果を調査することは、今後のネットワーク設計、サービス設計の観点から重要である。そこで、本報告では、その第一段階としてユーザーの通信品質に与える効果を調査すべく、サービスのエンド端末間における遅延時間に着目し計測を行った。その結果、仮想化が行われていないサーバーでサービスアプリケーションを実行する場合に比べ、仮想マシン上で実行する場合の方が、ライブストリーミング映像の遅延時間が大きくなることがわかった。さらに TCP 通信の距離を小さくするようにサービス機能を配置すると、遅延時間が抑えられることがわかったが、負荷の分散による遅延低減は観測されなかった。加えて、各サーバー (クラウドとコンピュートノード) の処理時間を計測する実験を行なった結果、仮想化されたサーバーの処理時間の方が、仮想化が行われていないサーバーの処理時間に比べ、大きいことが確認できた。しかし、仮想マシンを分散した場合は、分散したサーバーの処理時間を合計すると、同一のサーバーで集中処理した場合の処理時間と同等であり、負荷の分散による処理時間低減は観測されなかった。

今後は、Pepper とユーザーの双方向の通信における遅延時間の計測や、仮想化環境上でハードウェア資源の負荷が高まった際に、仮想マシンのライブマイグレーションなどによるサービス機能の動的な再配置がもたらす、通信品質への影響を調査する予定である。また、ネットワーク仮想化にもとづくサービス機能の再配置が、ユーザーの通信品質のみならず、ユーザーの体験品質に対する効果を明らかにすることを目指す。

謝辞

本報告を終えるにあたり、熱心にご指導、ご教授いただきました大阪大学大学院情報科学研究科の村田正幸教授に深謝いたします。日頃より適切なご助言を賜り、研究生活において様々な機会を与えてくださいましたことに、感謝の念が絶えません。ならびに、大阪大学大学院情報科学研究科の荒川伸一准教授には、多くの時間を私の指導に割いていただき、的確なご助言、自主性を重んじたご指導を通じて、研究の方向性を示していただきました。心より感謝申し上げます。また、平素から広くご指導いただきました大阪大学大学院情報科学研究科の大下裕一助教、大阪大学大学院経済学研究科の小南大智助教に厚く御礼申し上げます。最後に、日頃より様々な面で支えてくださいました、井上昴輝氏、宮川裕考氏をはじめとする研究室のみなさま、友人、家族のご厚意に感謝の意を表して謝辞といたします。

参考文献

[1] N. Fernando, S. W. Loke, and W. Rahayu, “Mobile cloud computing: A survey,” *Future generation computer systems*, vol. 29, no. 1, pp. 84–106, Jan. 2013.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” in *Proceedings of ACM SIGCOMM 2012*. ACM, Aug. 2012, pp. 13–16.

[3] R. Mijumbi, J. Serrat, J.-l. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, “Network Function Virtualization: State-of-the-Art and Research Challenges,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, Sep. 2015.

[4] “Mobile-Edge Computing Introductory Technical White Paper,” ETSI, Sep. 2014.

[5] S. Ahmadi, “Future of Wireless Networks: SDN/NFV, MEC and Network Slicing,” IEEE COMSOC SVC Talk, Jun. 2016.

[6] “Mobile-Edge Computing (MEC); Technical Requirements,” ETSI GS MEC 002 V1.1.1, Mar. 2016.

[7] Telefonica. (2016, Oct.) OpenMANO. [Online]. Available: http://www.tid.es/long-term-innovation/network-innovation/telefonica-nfv-reference-lab/openmano

[8] “Network Functions Virtualisation - Update White Paper,” ETSI, Oct. 2013.

[9] “Network Functions Virtualisation (NFV); Architectural Framework,” ETSI GS NFV 002 V1.1.1, Oct. 2013.

[10] “Mobile-Edge Computing (MEC); Framework and Reference Architecture,” ETSI GS MEC 003 V1.1.1, Mar. 2016.

[11] “IoT とエッジコンピューティング,” NTT 未来ねっと研究所, Mar. 2016.

[12] R. Mijumbi, J. Serrat, J.-l. Gorricho, S. Latré, M. Charalambides, and D. Lopez, “Management and orchestration challenges in network functions virtualization,” *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, Jan. 2016.

[13] OPNFV. (2016, Oct.) Home — Open Platform for NFV (OPNFV). [Online]. Available: https://www.opnfv.org/

[14] OSM. (2016, Nov.) OSM. [Online]. Available: https://osm.etsi.org/

[15] NFV Labs (Telefonica NFV reference lab). (2016, Oct.) Getting Started - nfvlabs_openmano Wiki - GitHub. [Online]. Available: https://github.com/nfvlabs/openmano/wiki/Getting-started

[16] ——. (2016, Oct.) Home - nfvlabs_openmano Wiki - GitHub. [Online]. Available: https://github.com/nfvlabs/openmano/wiki

[17] Y. Goto, "Kernel-based Virtual Machine Technology," *Fujitsu Scientific and Technical Journal*, vol. 47, pp. 362–368, Jul. 2011.

[18] "Network Functions Virtualisation (NFV); Infrastructure; Compute Domain," ETSI GS NFV 003 V1.1.1, Dec. 2014.

[19] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471–1480, Nov. 2012.

[20] FFmpeg. (2016, Dec.) FFmpeg. [Online]. Available: https://ffmpeg.org/

[21] Julius development team. (2017, Jan.) 大語彙連続音声認識エンジン Julius. [Online]. Available: http://julius.osdn.jp/

[22] 田中 裕之, 高橋 紀之, 川村 龍太郎, "IoT 時代を拓くエッジコンピューティングの研究開発," *NTT* 技術ジャーナル, pp. 59–63, Aug. 2015.