SDNを構成するネットワークサービス間の連携を実現する方式に関する研究

高橋　基　中島　潤
北海道情報大学

Studies on the method to realize the cooperation between network services and control plane of SDN

Motoi TAKAHASHI and Jun NAKAJIMA
Hokkaido Information University

平成26年11月

北海道情報大学紀要　第26巻　第1号別刷

〈論　文〉

SDN を構成するネットワークサービス間の連携を実現する方式に関する研究

高橋基*　中島潤†

Studies on the method to realize the cooperation between network services and control plane of SDN

Motoi Takahashi*　Jun Nakajima†

要旨

OpenFlow を筆頭に SDN(Software-Defined Networking)が注目されている．多くの場合，既存の SDN アーキテクチャでは，ファイアウォールやロードバランサなどのネットワークサービスが，SDN コントローラの API を利用するといった一方向の連携のみが想定されている．よって，ネットワーク上に発生した障害をネットワークサービスへリアルタイムに通知するなど，制御プレーンからネットワークサービスへ能動的に連携を行う際に問題が生じる．本研究では，双方向性を兼ね備え，ネットワークサービス間の連携を実現するフレームワークを提案し，その実装を行なった．

Abstract

SDN (Software-Defined Networking) has attracted attention, led by OpenFlow. In many cases, the existing SDN architectures assume only the one-way network cooperation such that load balancers and firewalls use the API of the SDN controller. Therefore, a problem arises when the control plane intends to work with the network services, as exemplified by the circumstance that the network services are notified of the failures that occurred on its network in real time.　In this study, we propose a framework which works interactively and realizes the cooperation between network services, and carried out its implementation.

キーワード

SDN(Software-Defined Networking)　Northbound API　OpenFlow

* 北海道情報大学大学院経営情報学研究科, Graduate School of Business Administration and Information Science, Hokkaido Information University

† 北海道情報大学経営情報学部准教授, Associate Professor, Faculty of Business Administration and Information Science, Hokkaido Information University

1. 序論

1-1 研究の背景

1-1-1 従来のネットワークアーキテクチャの課題と SDN

昨今，社会を支えるネットワークインフラの整備は進み，ネットワーク技術は著しい発展を遂げている．しかし，スマートフォンの普及やデータセンタ・クラウドコンピューティング技術の登場によってネットワークトラフィックは増加した．これに伴い，ネットワーク運用コストやスケーラビリティ，技術的な制約の問題を受け，従来のネットワークが抱える問題を解決し，柔軟性・迅速性を実現するための技術が求められていた．また，従来のネットワーク技術の仕組みでは，管理自動化が困難であり，ネットワーク機器を制御するプロトコルもベンダ固有であることが一般的であった．

これらの問題に対するソリューションの一つがSDN(Software-Defined Networking)であり，より柔軟でスケーラブルなネットワークの構築と運用管理の実現が期待できる新たなネットワーク技術として注目を集めている．SDNとは，ネットワークをソフトウェアで制御するために従来の垂直統合型システム[1]を分離し，ネットワークの柔軟性や迅速性，管理性の向上が期待できる新たなコンセプト，およびそれを実現する技術である[2]．SDNを実現する技術の一つであるOpenFlow は，従来の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を分離し，間のインタフェースのオープン化をした（図1.1）．インタフェースがオープンになったことで，コントロールプレーンを自由にプログラムし，任意のネットワーク制御ソフトウェアを組み込むことが可能となった．これによって従来のスイッチネットワークでは実現できなかった柔軟性・迅速性を備えたネットワーク制御の実現が期待されている．

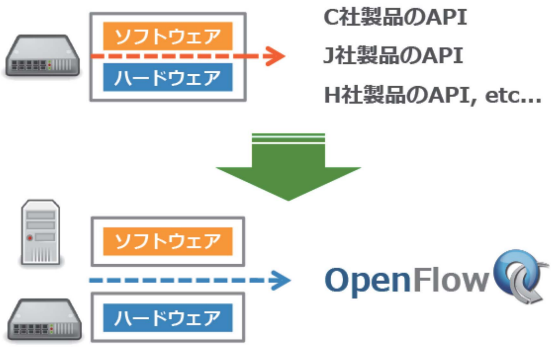

図 1.1
OpenFlowに対応したネットワーク機器

1-1-2 既存の SDN の課題

既存のSDNアーキテクチャは一般に３層に分けて考えられている（図1.2）[3] [4]．ONF(Open Networking Foundation)[5] によると，アプリケーションレイヤとコントロールレイヤ間のAPIをNorthbound API，コントロールレイヤとインフラストラクチャレイヤ間のAPIをSouthbound APIとしている．

Southbound APIは業界唯一の標準であるOpenFlowプロトコルによる標準化が進められているが，Northbound APIの標準化は進められていない．現状では各団体，各ネットワークベンダ企業の製品によって異なったAPIが利用されており，独自の様々なNorthbound APIが混在している．既存のNorthbound APIは，コントロールレイヤがアプリケーションレイヤに対してネットワークの制御情報を提供する目的で設計されている．しかし，ネットワーク侵入検知システムやネットワークモニタリングシステムのように，インフラストラクチャレイヤで発生したイベントをアプリケーションへ通知する必要があるサービスでは，SDNコントローラからアプリケーションに対してリアルタイムに通知を行うべきであると考えるが，既存のNorthbound APIの多くはREST(Representational State Transfer)[6] を用いたWeb APIを利用してい

るため，これをリアルタイムに実現することは容易ではない．このようなリアルタイム性を必要とするサービスを構築するためには，Northbound APIに双方向の通信を容易に実現する仕組みが必要であると考えた．

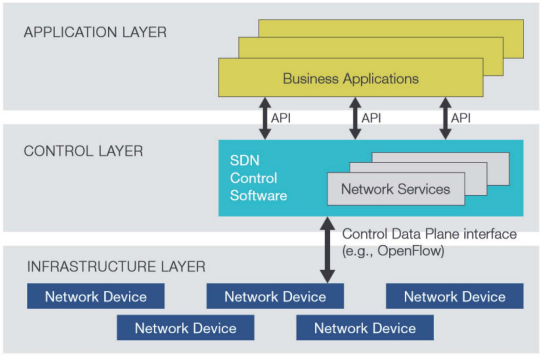

図 1.2
既存の SDN アーキテクチャ
https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf より引用

1-1-3 双方向通信の実現方法

既存のSDN フレームワークで双方向の連携を実現することを考える．

Web技術による双方向通信

既存のSDNフレームワークに用いられるNorthbound APIは．RESTによって設計されたWeb APIによって実装されている．Web APIを用いて双方向で連携を取ることを考えたとき，HTTPはコネクションレス型のプロトコルであり，クライアントからサーバに対してのみ能動的に通信を発生させられる．

Webをプラットフォームとするゲームや Webチャットシステム，あるいは常に最新の情報を必要とする株式情報を取り扱ったサービスでは，WebブラウザとWebサーバ間でよりリアルタイムに通信を行ないたいというニーズが存在した．Web上でリアルタイムに更新を行なう技術としてAjax技術が登場したが，クライアントのリクエストにサーバがレスポンスを返すというWebのアーキテクチャとしてのWebブラウザとWebサーバ間の関係は変わっておらず，Webサーバから能動的にリクエストを送ることは不可能であった．

そこで，擬似サーバプッシュを実現するComet[7] が登場した．CometではHTTPプロトコルを利用し，Webブラウザからのリクエストに対してWebサーバはレスポンス処理を保留し，任意のタイミングでレスポンスを返すことで，任意のタイミングでサーバから情報を送信する．この仕組みをロングポーリングと呼び，既存のHTTPプロトコル上で実現できるというメリットがあったが，HTTPコネクションを長時間占有するためにWebサーバのリソースを消費してしまう点と，タイムアウト毎に再接続が必要でありオーバヘッドが発生する点に問題があった．

これらの問題に対して新たに登場したのが，WebSocket[8] [9] である．WebSocketでは，HTTPプロトコルのUpgradeヘッダを使用し，WebブラウザとWebサーバ間の通信をHTTPとは異なる軽量のコネクション型プロトコルへ切り替える．AjaxやCometとは異なり，小さなオーバヘッドとHTTPコネクションを占有しないという特徴があり，現在はW3CがAPIの策定を進めている．

このことから，リアルタイムに双方向で情報交換を行う場合には，コネクション型のプロトコルを利用する必要があると考えた．

個別開発によるNorthbound API

アプリケーション，SDNコントローラごとに独自に定義されたNorthbound APIを使用し，連携対象となるアプリケーションごとに異なるAPIで連携を行なう方法がある．
しかし，マルチベンダ環境では，通信部分を個別に開発しなければならず，使用していたアプリケーションを更新する際には再度，通信部分を再開発が必要となる可能性がある．
また，特定のアプリケーションに依存することによってベンダロックインに陥る可能性があり，特定のアプリケーションに依存しない，

オープンな通信プロトコル, Northbound API を利用することが望ましいと考えている.

1-2　研究の目的

本研究では，既存のSDNでは容易に実現できなかった，双方向性を持ったネットワークサービス間連携を実現するSDNフレームワークを提案する．既存のSDNフレームワークでは，コントロールレイヤからアプリケーションレイヤへ能動的に通信を発生させることが容易ではない．提案手法では，アプリケーションレイヤとコントロールレイヤ間だけでなくアプリケーション同士の連携も視野に入れ，双方向の連携を容易に実現するSDNフレームワークを提案する．

提案手法の有効性を確認するために，検証を行うための試作システムとして仮想ネットワークタップ装置の構築を行なった．試作システムでは，双方向に連携が必要となる利用シナリオを想定し，アプリケーション同士が双方向に連携可能であることを確認する．

2.　SDN の動向と研究対象領域

2-1　既存の SDN

SDN の明確な定義は存在しておらず, 三者三様の捉え方がある．現在のSDNは，コントロールプレーンとデータプレーンをオープン化されたインタフェースによって分離し，ネットワーク構成管理，ネットワークリソース管理，ネットワークフロー管理などをソフトウェアによって実現可能とする技術・コンセプトの総称であると考えられる[2] [10] [11] [12] ．SDN はデータセンターを中心に成熟した技術であり，データセンターにおける需要を満たしつつあるが, 一般にLAN と呼ばれている大学や企業内ネットワークへのSDN への導入も進んでいる．2012 年4 月に開催されたONF 主催のOpen Networking Summit 2012 では，米Google 社の持つデータセンター間のインターナルバックボーンネットワークのトラフィックは，全てがOpenFlow を用いたSDN によって運用されていることが，テクニカルインフラストラクチャー部門のシニア・バイスプレジデントであるUrs Hoelzle 氏によって発表されている[13] ．米Google 社がSDNを採用した理由として，ネットワーク制御の集中管理による運用管理の効率化を挙げている．

また，国内においては，金沢大学付属病院のネットワークでOpenFlow を用いたSDN 導入事例もある[14] [15] ．

2-1-1 本研究で取り扱う SDN

既存のSDN 技術・製品は，ネットワークの集中管理・可視化，ネットワーク・トポロジーの簡略化, 1つのシステム環境を複数アプリケーションで共同して利用するマルチテナント環境の構築などを実現し，現在はデータセンターでの利用が活発である．従来のネットワーク技術で構築されたデータセンターでは, ネットワーク仮想化による管理の複雑化や, 4094 個が最大個数であるVLAN ID の制限によるスケーラビリティの不足，などといった課題が存在していたが，これらはOpenFlow によって解決できる．データセンターは最もSDN の恩恵が得られる環境であり，キャリアネットワーク，サービスプロバイダでのSDN 適用には，NFV技術[16] [17] が選択されているように[18] [19] [20] ，現状のSDN はデータセンターのニーズに応える技術であると考えられる[21] ．

本研究では，一般にLANと呼ばれる大学や企業内ネットワーク，病院などで利用されているネットワークでのSDN 適用を視野に入れている. したがって, 本研究で取り扱うSDN は，ネットワークプリンタやメールサーバ,

あるいは個人が所有するノートパソコンやスマートデバイスなどが混在する環境を想定している．

LAN と比較してデータセンターに無い要素として，無線LAN 環境やユーザ認証がある．無線LAN は不特定多数のユーザに利用され，想定外のユーザ端末が接続される可能性があり，組織内LAN においてユーザ認証は必須技術である．また，アプリケーションレイヤには，汎用サーバ上で動作するネットワークサービス以外にも，ユーザ端末上で動作するアプリケーションが存在し，Northbound 領域においては様々なアプリケーションとの連携可能性がある．

データセンターやキャリアネットワークのニーズは，仮想化による設備投資の最適化，リソース可視化による運用コストの最適化などで，それらはSouthbound 領域で解決されてきた．LAN に存在するエンドユーザのニーズに応えるためには，ユーザの持っている端末がどのようにしてSDN の恩恵を受けられるかが重要であり，エンドユーザが利用する上位のアプリケーションレイヤとコントロールレイヤ間のNorthbound 領域における連携が重要であると考えた．よって，本研究ではNorthbound 領域で発生するネットワークサービス間の連携を研究対象とする．

3. 新たな SDN フレームワークの設計

SDN を構成するネットワークサービス間の連携を想定し，SDN フレームワークで考慮すべき要件を明確にする．

3-1 既存の SDN フレームワーク

OpenDaylightは，Linux Foundationの運営によるSDN 製品・市場の活性化を目的としたオープンソースプロジェクトで，2013 年4月に発足された．Linux Foundation は OpenDaylight を「コミュニティが様々な企業の力を借り，新しいイノベーションを発展させ，SDN に関するよりオープンで透明性の高いアプローチの構築を目指すオープンソースフレームワーク」としている[22] ．実態として，複数の参加企業が自社のソースコードを持ち寄る形でSDNコントローラなどの複数の開発プロジェクトが存在しており，OpenDaylight プロジェクトとしてのファーストリリースは2013 年12 月上旬に予定されていたが数週間の遅延が発生し，2014年2月にリリースされた．OpenDaylight は，モジュール・プラグインアーキテクチャを特徴としており(図3.1)，Java 仮想マシンによって動作する．SDN コントローラに実装されているNorthbound API は，REST 技術を用いたWeb API によって実装されている．

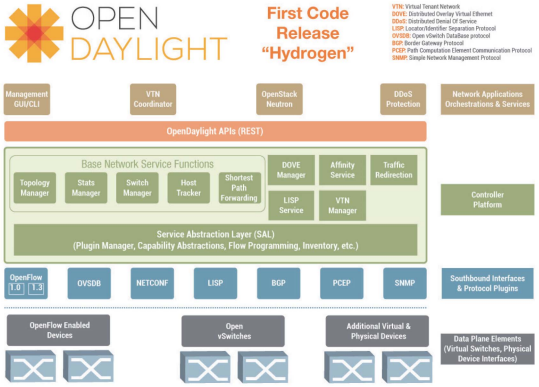

図 3.1

OpenDaylight コードネーム:
Hydrogen のアーキテクチャ

http://www.opendaylight.org/announcements/2013/09/opendaylight-project-releases-new-architecture-details-its-software-defined

より引用

3-2 ネットワークサービス間の連携を実現するための要件

本研究では，既存のSDN フレームワークでは容易に実現できなかった双方向通信を実現するSDN フレームワークを提案したい．そこ

で, 提案するSDN フレームワークで満たすべき要件をまとめる.

3-2-1 双方向通信の確保

1.1 節で述べたように, 既存のSDN フレームワークでは，アプリケーションレイヤとコントロールレイヤ間では双方向通信が考慮されていない．既存のNorthbound API は，SDN コントローラが提供するWeb API をアプリケーションが利用する形式である．よって現状では, SDN コントローラからアプリケーションへの通信タイミングは，アプリケーションからのリクエストに対するレスポンスとしてのみで，受動的な情報交換となる．例えば，ネットワークスイッチに障害が発生した際には，ネットワークスイッチを制御しているSDN コントローラからアプリケーションへ，その旨を伝えたいと考えたときに能動的に情報交換を行なう術がない．ネットワーク障害へ迅速に対応するためには，この情報がリアルタイムに伝えられることが要件として考えられる.

そこで，コントロールレイヤからもアプリケーションレイヤに対して能動的に通信を発生させるための双方向通信の仕組みが必要である．双方向に能動的な情報交換が可能になると，リアルタイム性が重視されるシステムの要件を満たすことができる.

3-2-2 通信プロトコルの氾濫抑制

現在，Northbound API に標準プロトコルは存在していない． ファイアウォールやロードバランサといったネットワークサービスには，運用ポリシに従って様々なソフトウェアまたはアプライアンスが利用される．多くの場合，同一の機能を提供するサービスであっても，ベンダやアプリケーションごとに異なったプロトコルが利用されている．標準プロトコルが存在しない場合，各ベンダ・組織による複数の独自プロトコルが氾濫することとなる.

また，従来はアプライアンス製品として販売されていたネットワーク機器は，Southbound API が非公開であり，製品ベンダによって独自開発されていた．そのためネットワークシステムの更新時には，システム更新の作業量や，それに掛かる膨大な費用コストを抑えるために，事実上ネットワーク機器ベンダを変更できないというベンダロックインが発生していた. しかし現在ではSDN 技術の発展によって，Southbound 領域にはオープンな標準技術としてOpenFlow が存在し，ベンダロックインから解放されたと言える.

SDN におけるNorthbound API の開発は比較的容易であり，Southbound API 領域に比べてベンダロックインが発生する可能性は低いと予想できる．しかし，オープンな標準規格が存在していないNorthbound API 領域でも，コントローラ製品におけるソフトウェアレベルのベンダロックインが発生する可能性がある．つまり，コントローラ製品に依存した特定のベンダによって定義されるNorthbound API を使用していた場合，Southbound 領域で回避できたベンダロックインが，Norhtbound 領域において再度発生すると考えられる.

通信プロトコルの氾濫を避け，ベンダロックインを回避するためには，オープンな標準プロトコルを採用するか，あるいはネットワークサービスごとに抽象化した通信プロトコルの設計が必要であると考えた.

3-3　新たな SDN フレームワークの提案

先にSDN フレームワークの要件として挙げた以下の２点

1. 双方向通信の確保
2. 通信プロトコルの氾濫抑制

を満たすSDN フレームワークを設計する. 要件を満たすための仕組みとして，コネクション型プロトコルの利用と，アプリケーションを抽象化することを提案する.

3-3-1 コネクションプロトコルの利用

既存のNorthbound API で利用されているHTTP プロトコルはコネクションレス型プロトコルである[23] ．コネクションレス型プロトコルであるHTTPプロトコルでは，クライアントからのリクエストに対してレスポンスを返すタイミングは限られており，サーバ側から任意のタイミングでクライアントへリクエストを送信することは容易ではない．よって，サーバからクライアントへ任意のタイミングで情報を発信するためには，双方向通信を実現する必要がある.

本研究では，双方向通信を実現するために，コネクション型プロトコルを利用して，常にアプリケーション間でコネクションを維持し，お互いに任意のタイミングで情報交換することを提案する.

3-3-2 アプリケーションの抽象化

新たなネットワークサービスの導入を考えたとき，多数のベンダや開発者がアプライアンス，又はソフトウェア製品を提供しており，これを利用するネットワーク管理者は組織のポリシーに従って，最も適したアプリケーションを選択する．つまり，同様の機能を提供するサービスであっても，複数のアプリケーションが存在している．しかし，これらを１つのネットワークサービスとして捉え，抽象化することができると考えた．また，同様のサービスを提供するアプリケーション同士であっても，異種のアプリケーション同士が情報交換するためには，同一のプロトコルを利用する必要がある.

そこで本研究では，ネットワークサービスごとにプロトコルを定義することで，アプリケーションを抽象化することを提案する．ネットワークサービスごとに抽象化したプロトコルを定義することで，新たなアプリケーションの導入やネットワークサービスの機能に変更が生じても，他のネットワークサービスのプロトコルに影響を及ぼすことなく，プロトコルの追加・拡張が可能であると考えた.

3-4　提案するSDN フレームワークの設計と実装

提案の有効性を検証するために，アプリケーション間の連携を必要とする試作システムを構築する．そこで，3.3 節で示した提案内容

1. コネクション型プロトコルの利用
2. アプリケーションの抽象化

を組み込んだSDN フレームワークの実装を行なった. 検証に向けたシステム構成として，情報交換を行うためのメッセージ配信サーバと，これを利用するクライアントアプリケーションを２つ配置する．連携のためのアプリケーション間の情報交換はメッセージ配信サーバを中継して行ない，連携を取る２つのアプリケーションには同様のクライアント実装を組み込んだ.

3-4-1 メッセージ配信サーバの実装

コネクション型プロトコルを利用してアプリケーション間の連携を行うための実装として，アプリケーション毎にクライアント実装，サーバ実装を組み込むP2P 方式による実装も可能であるが，ネットワーク上で連携を行うアプリケーションの情報を一箇所に集約するためにクライアント・サーバモデルで実装を行なった. サーバの役割として，SDN 上に

存在しているアプリケーションの情報を集約し，アプリケーション同士がお互いを発見しあう空間の提供と，アプリケーション間で情報交換するためのメッセージ配信を行う，連携のためのネットワーク中心ノードとなる．

また，メッセージ配信サーバには，3.2 節で示したSDN フレームワークの要件に加えて，メッセージ配信サーバでは以下の機能を実装した．

アプリケーションの発見機構

アプリケーションが他のアプリケーションと情報交換を行うときは，通信相手のIP アドレスなどの端末情報を事前に入手し，アプリケーションに設定しておく必要がある．ユニークにアプリケーションを識別するためのラベルと，そのユニークなアプリケーションが提供する，サービスを識別するためのラベルをクライアントに設定することで，この手続を簡略化する．ラベルを利用することで，連携を取りたいサービスを提供しているアプリケーションをメッセージ配信サーバ上で発見し，連携することができる．

グルーピングによる１対多の情報交換

クライアントをグループ化し，１対多の情報交換を実現する．グルーピングによる情報交換の流れは，IP マルチキャストグループ[24] と同等の機能であり，クライアントはグループに対して任意に参加・離脱が可能である．

例えば，同一のサービスを一括りにグループを作成し，アプリケーションプールとして利用したり，クライアントを仮想的なネットワークセグメントに括るように扱うことも可能であると考える．

メッセージ配信サーバ内でのグループの扱いは，グループ名によって管理され，クライアントはグループ名を指定して参加・離脱を行なう．グループ名はクライアントによって任意に作成され，グループに参加するクライアントが無くなると，グループはサーバ上から破棄される．

3-4-2 連携のためのクライアント実装

メッセージ配信サーバと通信し，他のアプリケーションと連携を取るためにメッセージを送信するクライアント部の実装を行う．メッセージ配信サーバ上に転送するメッセージとして，アプリケーションを抽象化したプロトコルを用いて情報交換する．今回は，メッセージを配信するためのプロトコルの上でアプリケーションの抽象化を行なった．しかし，これは必要条件ではなく，抽象化されたネットワークサービスのプロトコルはメッセージ配信サーバによってカプセル化されており，メッセージ配信プロトコルには依存しない．

アプリケーションをサービスとして抽象化

本研究では，SDN コントローラもネットワークサービスの１つと捉えているため，OpenFlow コントローラはSDN コントローラサービスを提供するアプリケーションの１つとして考える．4 章にて詳細を説明するが，検証環境では管理モニタとOpenFlow コントローラの，２つのアプリケーションが存在している（図．3.2）．それぞれ，障害通知を受け取るサービスと，仮想ネットワークタップ装置の機能を提供するサービスとして抽象化して考え，プロトコルを定義した．クライアント間で交換する情報は，連携を取るサービスによって異なる．

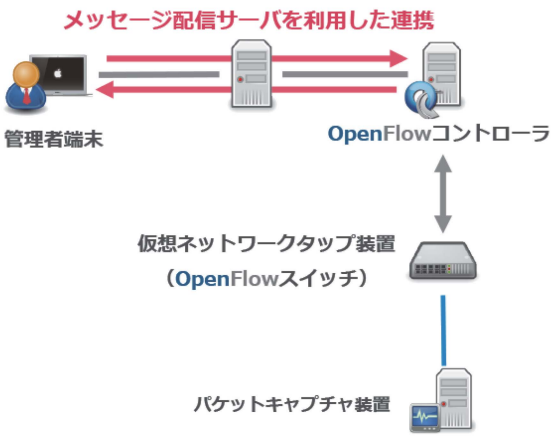

図 3.2
検証環境・仮想ネットワークタップ装置の構成

3-4-3 メッセージ配信サーバに実装した通信プロトコルの概要

メッセージ配信サーバで利用するプロトコルは，IRC[25] [26] [27] [28] をベースとしてプロトコルに変更と拡張を行ない，実装した．IRC は，サーバを介してクライアントとクライアントがテキストデータを交換し，インターネット上でチャットを行なうための通信プロトコルである．試作システムに用いるメッセージ配信サーバも同様にテキストデータによる情報交換によってアプリケーション間の連携をサポートする．テキスト要素間の区切り文字はスペース文字とした．

■ クライアントがメッセージ配信サーバへ送信するコマンドの一部の概要

●ログイン
コマンド名：LOGIN
パラメータ：<ID>
補足：ID には任意の半角英数文字列が指定可能．

●サービスの通知
コマンド名：TYPE
パラメータ：<サービス名，サービス名，...>
補足：ログインの直後に送信する．カンマ区切りで複数指定可．

●メッセージの送信
コマンド名：MSG
パラメータ：<宛先シーケンス番号> <ターゲット> <メッセージ>
補足：宛先シーケンス番号を指定しない場合は0 を指定する．

●メッセージの送信
コマンド名：MSG
パラメータ：<宛先シーケンス番号>
<ターゲット> <メッセージ>
補足：宛先シーケンス番号を指定しない場合は0．

●グループの参加
コマンド名：JOIN
パラメータ：<グループ名>
補足：グループが存在しない場合は自動的に作成される．

●グループの離脱
コマンド名：PART
パラメータ：<グループ名>
補足：グループ参加状態でサーバから切断した場合には自動的でグループから離脱処理される．

●サーバから切断
コマンド名：QUIT
パラメータ：なし
補足：QUIT コマンドを送信せずに切断した場合はタイムアウトされる．

●サービスを提供しているアプリケーションのリストを取得
コマンド名：LIST
パラメータ：<サービス名>

補足：該当するアプリケーションのID のリストが返信される．

●PING 応答
コマンド名：PONG
パラメータ：なし
補足：PING コマンドへの応答．

■ メッセージ配信サーバがクライアントへ送信するコマンドの一部の概要

●メッセージの受信
コマンド名：MSG
パラメータ：<宛先シーケンス番号> <シーケンス番号> <ターゲット> <メッセージ>
補足：宛先シーケンス番号は送信者が指定した番号で，シーケンス番号にはメッセージ配信サーバが自動生成した番号が入る．ターゲットにはおおよそクライアント識別子か，クライアントが参加しているグループ名が入る．

●クライアントの生存確認
コマンド名：PING
パラメータ：なし
補足：クライアントはPONG 応答を返す．

クライアントからクライアントへコマンドが送信される場合は，コマンド名の前に送信元であるクライアントのクライアント識別子が付加される．例えばクライアントA からクライアントB に対して「Hello, world!」とメッセージを送信する場合，クライアントA はメッセージ配信サーバに対して「MSG 0 B Hello, world!」と送信する．メッセージ配信サーバは，受信した内容からクライアント識別子として「:A」を先頭に付加し，クライアントAとクライアントB に対して「:A MSG 0 1234 B Hello, world!」を送信する．メッセージ送信側であるクライアントA にもメッセージを配信しているのには，MSG コマンドに付与されたシーケンス番号を送信元クライアントにも知らせる目的がある．

4. 試作システムによる提案の有効性の検証

4-1 検証の目的

提案の有効性を確認するために検証を行った．提案の有効性を確認するためには，提案するSDN フレームワークを利用してアプリケーション同士で双方向に連携できることを確認する必要がある．そのために実際にSDN フレームワークの実装を行ない，アプリケーション同士が連携を必要とする環境を構築して，アプリケーション間で連携が発生する利用シナリオを設定し，問題なくシステムが動作するかを確認する．また，実環境を用意することで，設計段階では想定できなかった問題を洗い出してゆく．

4-2 検証に向けた試作システムの開発

提案手法を用いた試作システムとして仮想ネットワークタップ装置を構築した（図. 3.2）．

4-2-1 仮想ネットワークタップ装置の開発

従来のネットワークタップではポートミラーリングによってトラフィックのタッピングを行うため，監視対象となるポートを流れる全てのトラフィックが複製されるものであった．試作システムで構築する仮想ネットワークタップ装置は，フィルタリングルールにしたがって特定のデータのみを対象にトラフィックをタッピングすることができる．このフィルタリングルールは，ユーザが任意に設定してゆく．

フィルタリングルールの追加・削除・変更などといった設定変更操作はリモートから行えるよう，専用のモニタリングソフトウェアを用意し，これを管理モニタとする．管理モニタはスタンドアロンで動作し，フィルタリングルールの設定変更操作の他，ネットワークタップ上で発生した障害をリアルタイムに監視することができる．ここで取り扱う障害は，LAN ケーブルの断線や，LAN ケーブルが抜けた場合など，OpenFlow コントローラの機能によって検出が可能な障害のみとする．

4-2-2 仮想ネットワークタップ装置の開発

仮想ネットワークタップ装置を利用し，特定のトラフィックのみをネットワークパケットキャプチャ装置へ転送するユースケースを想定する．このユースケースにおいては，管理モニタと仮想ネットワークタップ装置との間では双方向に連携を取る必要があり，以下にその利用シナリオを示す．以下の２つの利用シナリオによって，双方向に連携が実現できることを確認し，提案するSDN フレームワークの有効性を確認する．

管理モニタからOpenFlow コントローラへの連携

仮想ネットワークタップ装置の設定変更操作のために，管理モニタからOpenFlow コントローラに対して能動的に発生する連携シチュエーションを想定した利用シナリオである（図．4.1）．

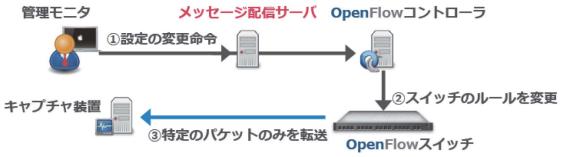

図 4.1
管理モニタから
OpenFlow コントローラへの連携

OpenFlow コントローラから管理モニタへの連携

仮想ネットワークタップ装置で発生した障害を管理モニタへ通知するために，OpenFlow コントローラから管理モニタに対して能動的に発生する連携シチュエーションを想定した利用シナリオである（図．4.2）．

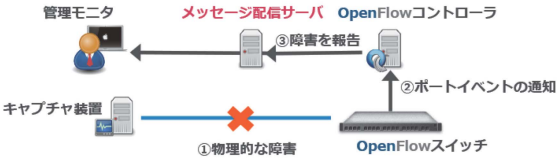

図 4.2
OpenFlow コントローラから
管理モニタへの連携

4-2-3 検証で用いた抽象化プロトコル

検証する利用シナリオに向けて，管理モニタとOpenFlow コントローラにアプリケーション抽象化プロトコルを実装した．実装したプロトコルの一部と概要は以下の通りである．

■障害情報を受け取るサービスのプロトコル

●アラートの受信
コマンド名：ALERT
パラメータ：なし
　　補足：警報を鳴らしたいときに使用する．
●障害情報の通知
コマンド名：INFO
パラメータ：<障害レベル> <障害情報>
　　補足：障害レベルには数値，障害情報には詳細を記載する．

■　ネットワークタップ装置のプロトコル

●フィルタリングルールの追加
コマンド名：FILTERADD

パラメータ:<TAP PORT> <IN PORT> [マッチングルール]

補足:TAP PORT タッピングしたパケットを転送するスイッチポート．IN PORT タッピングするために監視するスイッチポート．マッチングルール使用できる情報はIP アドレス，MAC アドレス，TCP/UDP ポート番号．Web トラフィックにマッチさせる場合，マッチングルールには例えば「PORT DST=80」と記載する．

●フィルタリングルールの削除
コマンド名：FILTERREM
パラメータ：<ルール番号>

補足：指定した番号のフィルタリングルールが削除される．フィルタリングルールのリスト番号は詰められる．

●フィルタリングルールの全削除
コマンド名：FILTERREMALL
パラメータ：なし

●設定済みフィルタリングルールのリストを取得
コマンド名：FILTERLIST
パラメータ：なし

4-2-4 検証環境と検証の流れ

実際に構築した検証環境の論理構成を図．4.3 に示す．まず，検証に利用したネットワークスイッチのVLAN 機能によって，2つの仮想スイッチ「スイッチA」と「スイッチB」を作成した．スイッチA は，L2 制御のみ行なう標準的な機能を持ったスイッチング・ハブであり，スイッチB はOpenFlow スイッチとして動作する．スイッチB は，スイッチAに接続されたOpenFlow コントローラによって制御され，検証環境の構成ではスイッチBが仮想ネットワークタップ装置本体として機能する．このとき，OpenFlow コントローラが仮想ネットワークタップ装置の制御機能を実装しており，OpenFlow スイッチはOpenFlow コントローラの指示によってタッピングを行なう．OpenFlow コントローラはスイッチA に接続されており，管理モニタからのフィルタリングルールの変更操作や，管理モニタに対しての障害の通知を行なう．検証環境において，OpenFlowコントローラに指示を出すのはスイッチA のに接続された管理モニタであり，この指示を出すための連携には，同様にスイッチA に接続されているメッセージ配信サーバを経由する．

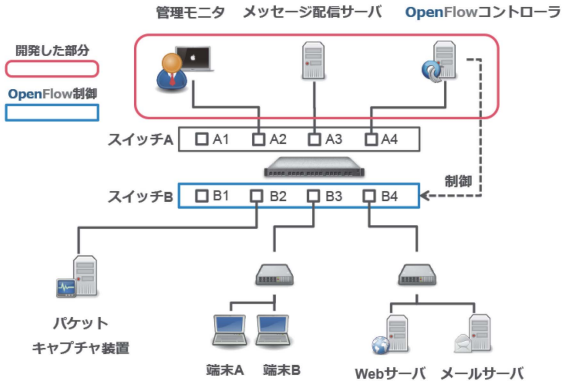

図 4.3
検証環境の論理構成

検証で想定した仮想ネットワークタップ装置のユースケースは，パケットキャプチャ装置を使ったネットワークトラフィックのキャプチャである．仮想ネットワークタップ装置はスイッチB であり，スイッチB のB3 ポートとB4 ポート間の通信トラフィックを，パケットキャプチャ装置が接続されているB2 ポートへタッピングする．このとき，タッピングするパケットはフィルタリングルールに従ったものであり，任意のパケットのみがパケットキャプチャ装置へタッピングされる．B3 ポートには複数のユーザ端末が接続されていることを想定しており，B4 ポートにはWeb サーバやメールサーバなどのサービスが稼働しているものとした．

4-3 検証に向けた試作システムの開発

検証環境（図．4.3）を元に説明する．
前提として，管理モニタとOpenFlow コントローラをメッセージ配信サーバにログイン状態にする．また，メッセージ配信サーバ上の管理モニタのクライアント識別子を「MONITOR」，OpenFlow コントローラのクライアント識別子を「OFC」とする．

4-3-1 管理モニタから OpenFlow コントローラへの連携

管理モニタからOpenFlow コントローラへの連携目的は，仮想ネットワークタップ装置に対するフィルタリングルールの設定変更である．そこで，仮想ネットワークタップ装置のユースケースとして，Web トラフィックのみをタッピングする設定を行なうことを想定して検証を行なった．

まず，未設定状態では，データのタッピングが行われないため，Web トラフィックのみタッピングされるよう，クライアント識別子「OFC」に対してフィルタリングルールの設定を行なった．B3 ポートに接続されているユーザ端末から，B4 ポートに接続されているサーバ宛てに，Web アクセスやICMP メッセージを送信して，Web トラフィックのみが B2 ポートにタッピングされることを確認し，管理モニタからOpenFlow コントローラへの連携が成功していることを確認した．

4-3-2 OpenFlow コントローラから管理モニタへの連携

OpenFlow コントローラから管理モニタへの連携目的は，障害情報の通知である．そこで，故意に物理障害を発生させて検証を行なった．

まず，仮想ネットワークタップ装置上で障害を発生させるために，パケットキャプチャ装置側のLAN ケーブルを物理的に抜き，スイッチB のB2 ポートをリンクダウンさせた．これによってスイッチB 上ではスイッチポートの状態変更イベント（以後，ポートイベントと呼ぶ）が発生し，OpenFlow コントローラへポートイベントがアップされる．
OpenFlow コントローラでは発生したポートイベントに従って，パケットキャプチャ装置の繋がっているB2 ポートがリンクダウンした旨を，クライアント識別子「MONITOR」に対してメッセージ送信した．管理モニタは，クライアント識別子「OFC」から受け取った障害情報を画面に表示し，OpenFlow コントローラから管理モニタへの連携を確認した．

4-4 開発環境

4-4-1 開発環境

仮想ネットワークタップ装置の機能は OpenFlow コントローラとして実装し，OpenFlow コントローラフレームワークであるTrema[29] を用いて開発した．Trema はC 言語とRuby 言語のどちらかで開発することができるが，開発スピードを考慮し，Ruby 言語を選択した．OpenFlow スイッチには Hewlett-Packard 社のHP 3500-24 Switch を利用し，実機環境で動作検証を行なった．開発で使用したOpenFlow プロトコルのバージョンは1.0 であり，これは使用したHP 3500-24 Switch のサポートするOpenFlow プロトコルのバージョンに依存したものである．また，管理モニタ上のクライアント実装と，メッセージ配信サーバの実装にもRuby 言語を用いた．実装言語を統一することで，OpenFlowコントローラ上に実装したクライアントの通信部分をユーザ端末側のクライアント実装にライブラリとして流用し，クライアントの通信部分の開発は一度で済ませることができた．

4-4-2 検証環境の物理構成

検証環境の物理構成を図．4.4 に示す．

OpenFlow スイッチとして使用したHP 3500-24 Switch には，vlan-1 とvlan-2 のVLAN 設定を行ない，デフォルトVLAN であるvlan-1 を通常のスイッチ制御，vlan-2 をOpenFlow 制御するように設定した．

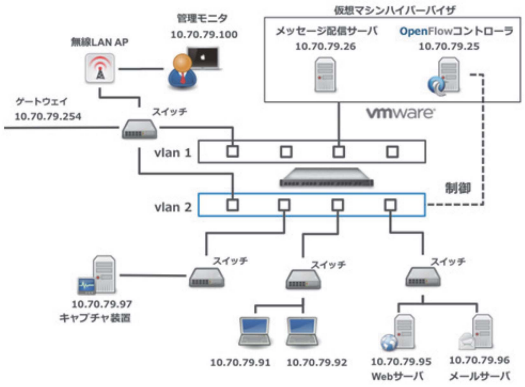

図 4.4
検証環境の物理構成

また，OpenFlow スイッチを制御するためにはOpenFlow コントローラが必要である．仮想ネットワークタップ装置の機能を実装したOpenFlow コントローラによってvlan-2 を制御するために，OpenFlow 制御する際のコントローラの指定にtcp:10.70.79.25:6633 を設定した．

また，メッセージ配信サーバ及びOpenFlow コントローラは，サーバ仮想化ソフトウェアであるVMware ESXi を用いて仮想マシンハイパーバイザ上に構築した．

検証・開発に用いたサーバのスペック，ソフトウェアのバージョンは以下の通りである．

■管理モニタ
型番：MacBook Air (Mid 2011)
CPU：1.7 GHz Intel Core i5
メモリ：4GB
OS：OS X v10.9.1
Ruby：v2.0.0-p247

■メッセージ配信サーバ
CPU：AMD PhenomII X6 1065T 2.9GHz*1
メモリ：512MB
OS：Debian v6.0.8
Ruby：v2.0.0-p353

■OpenFlow コントローラ
CPU：AMD PhenomII X6 1065T 2.9GHz*2
メモリ：512MB
OS：Ubuntu 12.04.3 LTS
Ruby：v2.0.0-p353
Trema：v0.4.6

■OpenFlow スイッチ
型番：HP 3500-24 Switch (J9470A)
ファームウェア：K.15.06.5008 (with OpenFlow support)
OpenFlow：v1.0 準拠

■仮想マシンハイパーバイザ
CPU：AMD PhenomII X6 1065T 2.9GHz
メモリ：4GB
OS：VMware ESXi 5.0.0 Update 1 build-702118

5. 結論

5-1 まとめ

本研究では，既存のSDN フレームワークでは実現が容易でなかった双方向での連携を実現するSDN フレームワークの提案を行ない，提案したSDN フレームワークを用いた試作システムとして仮想ネットワークタップ装置を開発した．さらに，試作システムを用いて提案の有効性を検証した．検証では，仮想ネットワークタップ装置のユースケースとして双方向に連携が必要な検証シナリオを２つ設

定し，試作システムは検証シナリオに耐えることができた．これにより，提案手法は双方向のネットワークサービス間の連携を実現する方式として有効であることを確認した．

既存のSDN フレームワークの多くは，Northbound API としてREST を用いたWeb API を利用していたため，リアルタイムな双方向通信を容易に実現することができなかった．提案したSDN フレームワークでは，コネクション型プロトコルを用いてクライアント・サーバモデルの通信プロトコルを設計し，双方向の連携を実現した．Northbound 領域においては，ネットワークサービスごとに通信プロトコルを定義することでアプリケーションの抽象化を行った．これにより，アプリケーションをネットワークサービスの１つとして捉えて連携することを可能とし，通信プロトコルの氾濫抑制が期待できる．また，連携相手を発見する仕組みを提供することで，通信相手を設定する手続きを簡略化した．

提案したSDN フレームワークでは，双方向通信を実現するために，クライアント・サーバモデルでコネクション型プロトコルを設計し，連携相手となるアプリケーションを発見するための機構を用意することで，通信相手の設定手続きを簡略化した．また，アプリケーションを抽象化し，ネットワークサービス間の連携を実現するために設計された通信プロトコルによって，Northbound 領域における通信プロトコルの氾濫抑制を期待できる．このことから，提案したSDN フレームワークを用いることで，既存のSDN フレームワークでは実現が容易ではなかった双方向通信の実現と，通信プロトコルの氾濫抑制が期待できる．

5-2　今後の課題

5-2-1 提案したプロトコルが持つ課題

提案したSDN フレームワークは，メッセージ配信サーバを中継サーバとして配置し，全トラフィックが中継サーバを経由するクライアント・サーバモデルであった．よって，トラフィック解析などで大量のトラフィックを転送しなければならないシステムを構築した場合に，中継サーバへの過剰な負荷，オーバヘッドの問題が発生すると予想される．そこで，大量のトラフィックが発生する場合には，クライアント・サーバモデルではなくP2P モデルを使った通信方式を利用する必要がある．

本研究で提案したSDN フレームワークは，チャットシステムであるIRC の通信プロトコル及びアーキテクチャを参考に設計を行ない，クライアント・サーバモデルで実装をした．IRC プロトコルにはDCC[30] [31] プロトコルと呼ばれるP2Pモデルでクライアント間が通信を行なう，IRC プロトコルとは別の通信プロトコルが存在する．提案したSDN フレームワークにおいても，このようなクライアント・サーバモデルとP2P モデルの双方のメリットが得られる仕組みが必要であり，Skypeに見られるようなハイブリッドP2P アーキテクチャ[32] を通信プロトコルとして利用することが望ましいと考えている．

5-2-2 追加の検証

提案したSDN フレームワークで設計した連携プロトコルでは，アプリケーションレイヤ内で，アプリケーションが他のアプリケーションと連携を取る目的で，アプリケーション対アプリケーションの連携も可能である．検証ではアプリケーションレイヤとコントロールレイヤ間の双方向の連携を確認したが，アプリケーション対アプリケーションの検証が行えていない．しかし，設計した連携プロトコル上では，SDN コントローラをネットワーク上に存在するアプリケーションの１つとして考えており，今回検証したアプリケーション対コントローラと，アプリケーション対

アプリケーションは，クライアントの実装上に相違はないことから，アプリケーション対アプリケーションの連携においても有効であると考えている．

また，3.3 節では，アプリケーションの抽象化を提案として挙げた．アプリケーションの抽象化を実現するプロトコルの有効性を検証するためには，同一のサービスを提供する複数個のアプリケーションを用意し，提供するサービスに必要な連携シチュエーションを想定してプロトコルを定義しなければならない．今回行った検証で構築したシステムでは，１つのサービスに対して１つのアプリケーションのみで構成されており，十分な検証が行えていないと考えている．今後は，検証規模を大きくし，同一のサービスを提供するアプリケーション群をブラックボックス化されたアプリケーションプールとして利用するなど，アプリケーションが混在する環境を想定した検証を行なう必要がある．

参考文献

[1] ITpro まとめ- 垂直統合型システム : ITpro, http://itpro.nikkeibp.co.jp/article/COLUMN/20130912/504269/（参照日: 2014.01.10）

[2] インターネット用語 1 分解説～ SDN (Software Defned Networking) とは～ -JPNIC, https://www.nic.ad.jp/ja/basics/terms/sdn.html（参照日: 2014.01.10）

[3] Software-Defined Networking (SDN) Definition - Open Networking Foundation, https://www.opennetworking.org/sdn-resources/sdn-definition（参照日: 2014.01.10）

[4] SDN Architecture Overview, Version 1.0, https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf（参照日: 2014.01.10）

[5] Home - Open Networking Foundation, https://www.opennetworking.org/（参照日: 2014.01.10）

[6] Roy Thomas Fielding (2000), Architectural Styles and the Design of Network-based Software Architectures, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm（参照日: 2014.01.10）

[7] 江島 健太郎 (2006), Lingr and Comet - 技術解説編, http://japan.cnet.com/blog/kenn/2006/09/22/entry_lingr_and_comet/（参照日: 2014.01.10）

[8] The WebSocket Protocol, http://tools.ietf.org/html/rfc6455（参照日: 2014.01.10）

[9] The WebSocket API, http://www.w3.org/TR/websockets（参照日: 2014.01.10）

[10] SDN とは: NEC の SDN ソリューション | NEC, http://jpn.nec.com/sdn/about sdn.html（参照日: 2014.01.10）

[11] Software-Defined Network (SDN) 概要 |Brocade Communications Systems, Inc., http://www.brocadejapan.com/solutions-technology/technology/software-defined-networking/overview（参照日: 2014.01.10）

[12] IBM Software Defined Networking - Japan, http://www-06.ibm.com/systems/jp/networking/solutions/sdn.html（参照日: 2014.01.10）

[13] Urs Hoelzle (2012), OpenFlow @ Google - Urs Hoelzle, Google, http://www.youtube.com/watch?v=VLHJUfgxEO4（参照日: 2014.01.10）

[14] 金沢大学附属病院が，新ネットワークに NEC の「プログラマブルフロー」を導入 (2012 年 6 月 11 日): プレスリリース| NEC, http://www.nec.co.jp/press/ja/1206/1101.html（参照日: 2014.01.10）

[15] 金沢大学附属病院様
ProgrammableFlow 導入事例,
http://jpn.nec.com/univerge/pow/pdf/kanazawa jirei detail 20131025.pdf
（参照日: 2014.01.10）

[16] Network Functions Virtualisation – Introductry White Paper,
http://portal.etsi.org/NFV/NFV_White_Paper.pdf
（参照日: 2014.01.10）

[17] Network Functions Virtualisation – Update White Paper,
http://portal.etsi.org/NFV/NFV_White_Paper2.pdf
（参照日: 2014.01.10）

[18] Marc Cohn (2013), NFV, An Insider's Perspective: Part 1 — Goals, History, and Promise,
http://www.sdncentral.com/education/nfv-insiders-perspective-part-1-goals-history-promise/2013/09/
（参照日: 2014.01.10）

[19] Marc Cohn (2013), There's a Network in NFV: The Business Case for SDN - | SDNCentral,
http://www.sdncentral.com/education/nfv-insiders-perspective-part-2-theres-network-nfv-business-case-sdn/2013/09/
（参照日: 2014.01.10）

[20] Marc Cohn (2013), NFV Insider's Perspective: An Operator Shift Underway - | SDNCentral,
http://www.sdncentral.com/education/nfv-insiders-perspective-operator-shift-underway/2013/10/
（参照日: 2014.01.10）

[21] Prayson Pate (2013), NFV and SDN: What's the Difference?,
http://www.sdncentral.com/technology/nfv-and-sdn-whats-the-difference/2013/03/
（参照日: 2014.01.10）

[22] 業界リーダー各社が共同で取り組む OpenDaylight プロジェクト主要技術の提供でソフトウェア・デファインド・ネットワーキング（SDN）構築を加速化,
http://www.linuxfoundation.jp/content/opendaylight launch releasae（参照日: 2014.01.10）

[23] 戸根勤[著], 日経オープン編集部[編集]（2002）, 『基礎からわかるネットワーク入門』(シリーズ＜基礎からわかる＞), 日経 BP 社

[24] Dave Kosiur [著], 苅田幸雄[邦訳]（1999）, マスタリング TCP/IP IP マルチキャスト編, オーム社

[25] RFC 2810 - Internet Relay Chat: Architecture,
https://tools.ietf.org/html/rfc2810
（参照日: 2014.01.10）

[26] RFC 2811 - Internet Relay Chat: Channel Management,
https://tools.ietf.org/html/rfc2811
（参照日: 2014.01.10）

[27] RFC 2812 - Internet Relay Chat: Client Protocol,
https://tools.ietf.org/html/rfc2812
（参照日: 2014.01.10）

[28] RFC 2813 - Internet Relay Chat: Server Protocol,
http://tools.ietf.org/html/rfc2813
（参照日: 2014.01.10）

[29] Trema,
http://trema.github.com/trema/
（参照日: 2014.01.10）

[30] The Client-To-Client Protocol (CTCP),
http://www.irchelp.org/irchelp/rfc/ctcpspec.html
（参照日: 2014.01.10）

[31] A description of the DCC protocol,
http://www.irchelp.org/irchelp/rfc/dccspec.html
（参照日: 2014.01.10）

[32] 森下民平, P2P アーキテクチャ,

http://www.cac.co.jp/softechs/pdf/st250107.pdf（参照日: 2014.01.10）