データネットワークを用いた高信頼SDN コントロールプレーンの設計と実装

HASHIMOTO, Naoki / 橋本, 直樹

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
10
(開始ページ / Start Page)
1
(終了ページ / End Page)
6
(発行年 / Year)
2015-03-24
(URL)
https://doi.org/10.15002/00011706

データネットワークを用いた高信頼 SDN コントロールプレーンの設計と実装

Design and Implementation of Reliable SDN Control Plane over Data Plane Network

橋本　直樹
Naoki Hashimoto
法政大学大学院情報科学研究科情報科学専攻
E-mail: naoki.hashimoto.4z@stu.hosei.ac.jp

Abstract

Software-Defined Networking (SDN) enables dynamic and adaptable network through directly programmability from services and applications. OpenFlow, which is reference architecture of SDN, decouples the network management plane from the data transfer plane to achieve the flexibility on managing the network traffic. In other words, OpenFlow network became to have two types of devices and two types of networks connecting them. The reliability of the management plane is essentially important to achieve the stable network using OpenFlow technology. In this paper, we propose a dynamic OpenFlow channel setup method over the data forwarding network in-band. It simply sets the forwarding rules for the packets of OpenFlow control channels to realize both of building overlay channels and handling the recovery on emergency. We implemented this method on the OpenFlow test-bed using software switch with OpenFlow 1.3.

1　はじめに

サービスの多様化およびそれを利用する大量のユーザの爆発的増加によって，インターネット設計当初に想定されていたような端末間通信や単純なサーバークライアント型のシステムと異なる新たな要求が突き付けられている．日進月歩で進化していくサービスにより生まれる新たな要求に即座に応え，多種多様なサービスの展開を加速させるネットワークを実現するための技術として Software Defined Network (SDN)[1] が注目されている．現在主流となっている TCP/IP や Ethernet といった Internet を構成するネットワーク技術では，機器のソフトウェアとハードウェアが一体となっているため，機器ベンダーが想定するアーキテクチャ以外のネットワーク構成は実現しにくいという問題があった．SDN では個々のネットワーク機器で制御ソフトウェアが稼働するのではなく，各装置を集めたネットワーク全体を一つの単位として一括で制御する．OpenFlow[2] はこの SDN における代表的なアーキテクチャの一つである．

OpenFlow は制御とデータ転送を別の装置に分離した機構を持つため，制御ネットワークとデータネットワークを別々に構築しなければならない．通常のデータネットワークは冗長性を持ち，これを利用することで広帯域化するアプローチがあるが，制御ネットワークに対しては広帯域化の要求が薄いため，冗長性を持たせることは少ない．一方で制御ネットワークが切断されると OpenFlow プロトコルでは制御そのものができなくなってしまうことから，いかにして安定的な制御ネットワークを構築するかが課題となっている．これについて従来の研究で制御ネットワークをデータネットワーク上に自動構築するための手法 [4] が提案されている．しかし，この自動構築には特殊な制御メッセージが必要であるため，OpenFlow スイッチとコントローラに改造が必要であった．本研究ではデータネットワークが本来持つ冗長性に主眼を置き，SDN がサポートするフロー操作によって制御パケットの経路を動的に操作する手法を提案し，OpenFlow プロトコルを用いて実装する．データネットワークは接続断などのトポロジ変化によって経路変更が必要となるため，これらの障害に対するコントロールプレーンの接続性について述べる．

2　OpenFlow プロトコル

SDN はネットワーク制御機能がデータ転送機能から分離され，プログラムによるネットワークの制御を実現可能にする新しいアプローチを持つネットワークである．プログラマブルであることによって動的に，かつ抽象化されたネットワーク制御が可能となり，これまで各ベンダーの独自実装であった制御機構が標準化されたことで，ネットワーク管理者自身が個々の環境に適した設定や新たな独自機能の実装が可能となり，柔軟性や管理における利便性が高まった．

OpenFlow は SDN のデファクトスタンダードであり，従来のネットワーク機器におけるパケット転送処理を行うスイッチと，アドレス学習やルーティングなど経路制御機能を担うコントローラの 2 種の装置によって構成される．これにより機器の制御がコントローラによって一元管理されるため中央集権型のアーキテクチャを実現可能であるという特徴を持つ．スイッチとコントローラの接続は OpenFlow チャンネルと呼ばれる通信によって実現され，MAC アドレスや IP アドレス，トランスポート番号などのフィールド情報によって決定される一連の通信単位，フローに対してアクションを指定し経路制御を行う．このフローは OSI 参照モデルにおける第 1 層から第 4 層相当の任意層を扱うことが可能であり，スイッチはコントローラに指定された振る舞いのみを処理するため各スイッチに要求される計算能力を低く抑えることができる．また，コントローラがネットワーク全体を俯瞰することによって柔軟な制御を可能とする．

Supervisor: Prof. Toshio Hirotsu

2.1 フローエントリー

フローエントリーはマッチフィールド，インストラクション，カウンタ，優先度，タイムアウト，クッキーの 6 つの要素で構成される．OpenFlow コントローラは各スイッチが複数保持しているフローテーブルに，フローとインストラクションの組であるフローエントリーを設定する．

マッチフィールドはスイッチがフローを受信した際にトラフィックを識別し特定するためのルールで，OpenFlow 1.3[3] では 40 種，レイヤー 1-4 層相当のフィールドとその依存関係が定義されている．さらに書き込むテーブルも複数存在しており，テーブルと優先度を合わせた組み合わせによって今までの枠組みに縛られない通信の制御が可能となっている．マッチングに失敗したフローについては破棄もしくはコントローラへの通知のどちらかを予め指定しておくことが可能である．OpenFlow1.1 以降はポートのグループ化によって，コントローラとの通信を挟まずに簡易的な複数経路やマルチキャストを実現可能となっている．グループのタイプとして以下 4 つが定義されている．

all　同グループに属する全てのポート
select　定義された選択法に基づいて選択されたポート
indirect　特定の一つのポート
fast failover　アクティブな最初のポート

インストラクションは，フローに対する処理定義であるアクションの集合である．マッチフィールドに一致した通信を受け取ったスイッチは，アクションの記述に従って通信を制御するが，複数のフローテーブルを跨ぐような処理はアクションセットとしてプールされ，行われるべき全アクションの追加が終わった時点で実行される．主なアクションには「出力，フィールドの書き換え，キュー，ドロップ」がある．

カウンタはテーブル，フロー，ポート毎に管理され，パケット数やバイト数，当該フローが OpenFlow スイッチ上に生成されてからの時間などの情報を記憶する．カウンタの値によって，ネットワーク情報の可視化，トラフィックの流量に応じたフローごとの帯域制御，トラフィックの傾向に応じた転送パスの切り替えなどを行うことができる．これらの情報はコントローラがスイッチに要求することで出力される．

2.2 OpenFlow メッセージ

OpenFlow プロトコルは 3 種類のメッセージタイプ (Controller-to-Switch, 非同期，同期) をサポートしている．これらのメッセージは前述のマッチフィールド，アクション，統計情報などを保持している．主な内容としては，機能やバージョン確認を行うための情報交換 (Hello)，スイッチのポート状況など状態監視 (Features)，フローテーブルでマッチするエントリーがない場合のコントローラへの転送 (Packet-In)，フローテーブルの追加や変更，削除 (Flow-Mod)，カウンターの収集が挙げられる．コントローラはこれらの通信で各スイッチを管理し，適切な設定を与える．本研究で使用するメッセージについて説明する．

2.2.1 Packet-In メッセージ

エントリーテーブルにマッチしない，あるいは出力先が CONTROLLER となっているフローをコントローラへ通知するために送信される．スイッチに十分な記憶領域があるときには，Packet-In の際に対象フローをバッファリングした上で，予め指定されたサイズで切り捨てられたパケットデータをコントローラに送信する．そのフローに対するコントローラからの返答があったとき，このバッファを利用することでメッセージサイズを抑えることが可能となる．

2.2.2 Flow-Mod・Group-Mod メッセージ

フローテーブルやグループテーブルに対してフローエントリー及びグループ情報を設定するため，コントローラが各スイッチに対して送信する．フローエントリーにはオプションとして idle- time out と hard-time out を指定できる．前者は最後にマッチしてからの経過時間，後者は該当エントリーがスイッチに追加されてからの指定時間が経過するとエントリーテーブルから削除される．またこれらオプションによりエントリーが期限切れになった時に，スイッチがコントローラに削除されたことを通知するかどうかを設定することもできる．また，グループは 2.1 節で示したようなポートを抽象化する機能であり，登録された複数のポートの中から指定されたタイプによって出力元が選択される．

2.3 OpenFlow チャンネル

OpenFlow チャンネルとは, 各 OpenFlow スイッチと OpenFlow コントローラとの間を接続する通信路を指す．スイッチはデータ転送の機能のみを持つため，既存のネットワーク機器群のようなその装置自身による経路制御を行わない．このためスイッチ単体でコントローラとの到達性を確保できず，通常は OpenFlow プロトコルによらない制御専用のネットワークを構築する必要がある．

この接続を確立する手順としては，まずスイッチが TLS または TCP セッションをコントローラに対して開始する．次に Hello メッセージによって互いのバージョンを確認して OpenFlow チャンネルが確立される．通常このときに Feature，Configuration を送受信してスイッチの情報を取得，設定を行う．定期的に Echo メッセージを送受信して接続が継続していることを確認し，仮に接続が切断されている時にはスイッチは fail secure もしくは fail standalone に切り替わる．前者は切断時にもコントローラによって定義されたフローエントリに従って該当するパケットを転送するが，後者は全てのパケットをドロップする．

2.4 OpenFlow が利用する主要プロトコル

2.4.1 Address Resolution Protocol

Address Resolution Protocol (ARP) は，EtherType が 0x0806 で，論理的な IP アドレスを物理的なハードウェア・アドレスである MAC アドレスに変換するために用いられる．TLS/TCP セッションは MAC アドレスの解決が必要なため，このプロトコルが用いられる．ただしリクエストを受信した側は変換要求元の MAC と IP アドレスを利用することが出来るので, 相互にリクエストパケットを送信する必要はない．ARP によって取得した MAC アドレスはキャッシュされ，以後の IP 通信で利用される．

2.4.2 Transmission Control Protocol

Transmission Control Protocol (TCP) は，IP プロトコル番号 6 を持つトランスポート層の通信プロトコルで，欠損パケットの再転送などのエラー訂正によって高信頼の 1 対 1 通

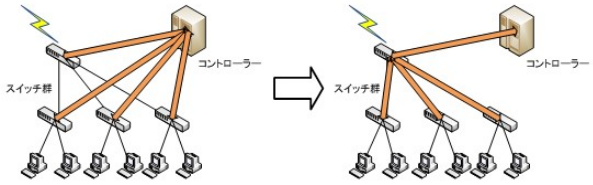

図 1　OpenFlow チャンネルのオーバーレイ

信 (セッション) を実現できる．OpenFlow1.3 ではコントローラのデフォルト待ち受けポートが 6653 であり，これに対して接続を試みたスイッチとセッションを張る．

2.4.3　Link Layer Discovery Protocol

Link Layer Discovery Protocol(LLDP) は，近隣ノードを検知するため，ネットワーク機器や端末の種類，設定情報などを近隣のノードに通知するレイヤ 2 レベルのプロトコルである．手順としてはコントローラの管理下にある全スイッチのすべてのポートから，送信側のスイッチ ID，物理ポート番号を保持した LLDP パケットを出力する．同一コントローラによる LLDP パケットを別のポートで受信したとき，コントローラは受信ポートと LLDP パケットに記録された送信元ポートが接続されていることを把握できる．

3　既存研究

文献 [4] では，OpenFlow ネットワークの制御ネットワークにおける問題点として，コントローラとスイッチ間の到達性を保障することによる柔軟性の低下や，制御ネットワークを別途用意することによるコストの増大，適用範囲に制約が出来てしまうことの 3 点を挙げている．これらは一概に制御ネットワークが自由にコントロールできないことに起因する．

これを解決するために，図 1 のように制御ネットワークをデータネットワーク上に重ねて作る，OpenFlow チャンネルのオーバーレイ化が提案されている．この手法では，EtherType フィールドを 0xF103 とする計 4 種類の特殊フレームを用いて制御パケットをラッピングすることで，既存の TCP/IP プロトコル上で通信することを可能としている．それぞれ，スイッチ間の接続関係を把握するための Request と Reply メッセージ，コントローラとスイッチ間の OpenFlow チャンネル経路を通知する Setup メッセージ，OpenFlow チャンネルをトンネリングするための Tunneling メッセージである．これらメッセージによって，コントローラからの IP による到達性がないスイッチに対して転送表を作成せずに制御メッセージを転送・制御する．ただし，この特殊なフレームの解釈のために，スイッチ・コントローラ両装置に対する変更を必要とする．

そこで本研究では OpenFlow チャンネルが TLS/TCP セッションを用いることを利用し，OpenFlow チャンネルをデータネットワーク上に構築して制御メッセージをなるべく欠損させずに転送できること，そしてデータネットワークのトポロジ変更イベントが発生しても各スイッチの制御を失わないことの 2 点を実現させる．特殊フレームを用いず，OpenFlow プロトコルの仕様に従って TCP パケットの転送表を逐次構築することによって，既存装置に変更を加えることなく実現する．

4　OpenFlow チャンネルの構築

OpenFlow スイッチのデータ転送処理機能は，予めフローテーブルやグループテーブルに設定されているか，もしくはコントローラによる制御を受けているかどちらかでなければ全く機能しない．つまり，スイッチに対するデータネットワークの物理的な接続だけではスイッチはアクションひとつ起こすことができない．制御できないスイッチにフローエントリーを設定することもできないため，コントローラからの如何なる設定も受け付けない状態に陥ってしまう．

しかし，本手法を適用することによってデータネットワークと別に制御ネットワークを構築する必然性がなくなり，またデータネットワークに存在する冗長性を利用することによる障害耐性も期待することができる．ただし，後述する OpenFlow チャンネルを構成するフローエントリーを維持し，他のフローエントリーがこれを阻害しないようにする必要があることを留意しなければならない．

OpenFlow スイッチには，OpenFlow チャンネルに接続するためのインターフェイスが存在しており，このインターフェイスに設定されたアドレスを利用して制御メッセージを入出力する．通常このインターフェイスは任意に指定できるが，本システムでは固定されている場合でも，あるポート (パススルーポート) からそのインターフェイスへループバックするように配線することも利用可能である．OpenFlow 管理下のいずれかのポートがコントローラ，あるいはコントローラへの経路が確保されている別のスイッチに接続することで OpenFlow チャンネルの物理経路を確保する．

次にソフトウェア的な接続を確立するためのフローエントリーの制御について述べる．各スイッチはどの物理ポートがコントローラへ通じているか，他のスイッチへ制御メッセージの中継を行う必要があるか，中継する場合にはどの物理ポートに繋がっているかといった情報をコントローラで集計し OpenFlow チャンネルの経路を決定する．各エントリーを起動エントリー，初期エントリー，確立エントリー，中継エントリーと呼称することとし，これを表 1 に示す．

4.1　起動エントリー

TCP セッションを開始するためにコントローラとスイッチ間の ARP，TCP パケットの転送エントリーを設定しなければならない．このためにスイッチからコントローラへ向かう通信はフラッディングし，コントローラから該当スイッチへ向かう通信はパススルーポートへ転送する．ARP パケットの交換によって互いに MAC アドレスを取得し，TCP 通信によって制御メッセージを転送する．

これらのエントリーはスイッチがコントローラの制御下にないため，各スイッチへ直接設定する必要がある．スイッチの起動時，コントローラの IP アドレスは既知情報であり，これを利用することで起動に伴う最低限の設定項目を一つに絞ることで手間を省く．

4.2　初期エントリー

初期エントリーでは，各スイッチでコントローラへ繋がっている物理ポート (以降，チャンネルポート) 番号を取得するための Packet-In メッセージを生成するフローを設定する．コン

表 1　起動，初期，確立，中継エントリー

Entry	Match Field	output
起動	etherType=ARP, IP dst=Controller	FLOOD
	etherType=ARP, IP src=Controller	パススルーポート
初期	etherType=IP dst=Controller	CONTROLLER FLOOD
確立	etherType=ARP, IP dst=Controller	チャンネルポート
	etherType=ARP, IP src=Controller	パススルーポート
中継	etherType=ARP opcode=request dst=Controller	CONTROLLER チャンネルポート
	etherType=ARP, IP src=中継スイッチ dst=Controller	チャンネルポート
	etherType=ARP, IP src=Controller dst=中継スイッチ	中継ポート

トローラとスイッチの接続が確認されたとき，該当スイッチがコントローラの管理下に入ったことで制御フローを送受信することが可能となり，フローエントリーの修正やポート情報の取得などが出来る状態になる．

接続されたスイッチがコントローラから見て初めて接続されている場合は，コントローラからスイッチへと送信されたメッセージに対してアクション CONTROLLER を追加設定し，起動・初期エントリー以外のフローを初期化する．Packet-In メッセージによって対象スイッチはコントローラと接続されている物理ポート番号をコントローラへ通知することできる．

4.3　確立エントリー

確立エントリーは，初期エントリーによってチャンネルポート番号を取得した際に設定する．フラッディングベースであった初期エントリーよりも優先度を高く設定した，チャンネルポートを出力先としたエントリーを設定することで，対象スイッチのフローによるフラッディングを抑制する．

4.4　中継エントリー

コントローラと該当スイッチ間の通信は前述の起動，初期，確立エントリーによって行われる．ただし，コントローラに直接接続されていないスイッチに対する通信は他のスイッチを中継する必要があるが，起動・初期・確立エントリーは他のスイッチからの中継フローにマッチしない．初めて他のスイッチが接続されようとしたことをコントローラへ通知するため，中継エントリーとしてコントローラのアドレスに対する ARP リクエストに対してアクション CONTROLLER を設定する．これを検知したとき，ARP，TCP/IP セッションを転送するためのフローエントリーを追加する．中継エントリーは中継を必要としているスイッチのアドレスを用いるため，コントローラに近いスイッチほど中継エントリーの数が多くなる．

データネットワークの変更によって中継経路が変わった場合，コントローラは最優先で中継エントリーを設定する必要がある．このフローエントリーが無ければ該当スイッチまで制御パケットを転送することができないからである．

表 2　冗長エントリー

Entry	Match Field	output
チャンネル グループ	type=fast failover	チャンネルポート 冗長ポート
冗長	etherType=ARP, IP dst=Controller	チャンネルグループ
	in_port=チャンネルポート etherType=IP dst=Controller	冗長ポート

5　OpenFlow チャンネルの動的変更

5.1　チャンネルポートの抽象化

コントローラが管理しているスイッチにおいて，ポートやそのポートに接続されている回線に障害が発生すると，OpenFlow スイッチは検知したリンクダウンを PortStatus メッセージによってコントローラへ通知する．これによりリンクダウンがポートのコンフィグ情報の変更として伝わるので，コントローラはその変更に適用するように新たな経路を計算し，各スイッチで Flow-Mod メッセージによってフローテーブルの書き換えを行うことができる．

しかし，4 章で示したフローエントリーでは，OpenFlow チャンネルの経路上に障害が発生すると，OpenFlow チャンネルに関するフローエントリーがタイムアウトによって削除されるまでの間，制御メッセージのトラフィックが遮断されてしまう．そのため，OpenFlow チャンネルに関するフローエントリーの書き換えも行うことができず，スイッチ側から OpenFlow チャンネルを再接続するまでの間は当該スイッチが制御不能となる．この対策として，障害が発生したときに他のポートからパケットを送出するようなフローエントリーを各スイッチに事前に設定しておくことが考えられるが，各フローに対してマッチングするフローエントリーは一つに限られるため，各スイッチでコントローラへ接続された物理ポート番号を一意に固定することができない．このため障害が発生しているポートから出力しているフローエントリーがタイムアウトするまで待機しなければならない．

そこで，予め定められたアルゴリズムによって選択された OpenFlow チャンネルに利用されるポートとは別に，そのポートで障害が発生した際に利用する迂回経路に接続されたポート (以降，冗長ポート) を選択し，Fast Failover Group としてグループ化する．出力先にチャンネルポートと冗長ポートをグループとしたものを指定することで，対象スイッチのチャンネルポートがリンクダウンしてもコントローラを介さず，即座に対応することができる．チャンネルポートの抽象化によるエントリーの修正を表 2 に示す．

各スイッチは，チャンネルポートと冗長ポートの順に Fast Failover Group とし，起動・確立・中継で設定されるフローエントリーの出力に利用する．また，コントローラへと向かう制御メッセージがチャンネルポートから入力された場合，その先の接続で何らかの障害が発生したと推定できるので，そのパケットは優先的に冗長ポートで出力する．

図 2 のネットワークトポロジを用いて，SW1 と SW3 の間で障害が発生したと仮定したときの SW6 で生成された制御パ

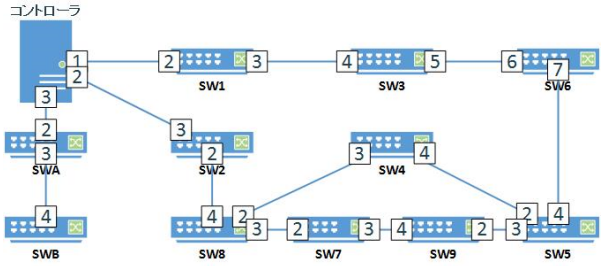

図 2　冗長経路の例

ケットを例に，障害時の動作手順を具体的に説明する．まず，障害が発生していない時には，各スイッチからダイクストラ法による最短経路の方向に OpenFlow チャンネルの制御用フローエントリーが設定されている．そのため，SW6 で発生した制御メッセージは SW3 - SW1 と伝わってコントローラに届く．しかし，障害が発生すると SW3 のポート 4 番で制御パケットの転送に失敗するため，Fast Failover Group 機能によってポート 5 番へ出力先を変更する．これによって SW6 へ制御パケットが戻されるため，コントローラへ送出したメッセージがチャンネルポートから戻ってきたパケットを冗長ポートであるポート 7 番へ出力する．以下，同じようにパススルーポートで受信したスイッチは冗長経路へ，それ以外で受信したスイッチはチャンネルポートへ転送することで最終的にコントローラへ到達する．

5.2　データネットワークの冗長経路算出

冗長ポートを算出する手順は以下の通りである．チャンネルポート以外を対象として最もコスト (中継経路との重なり，経路のホップ数) の低い経路を算出して冗長経路とし，この経路で利用するポートを冗長ポートとする．OpenFlow チャンネルで発生した障害に対して制御ネットワーク全体のフローエントリーが更新完了するまで利用される．

1. 探索元のスイッチでチャンネルポートは対象から除外する．このとき，チャンネルポート以外に直接接続されたスイッチがないならば冗長経路は存在しない
2. 探索元のスイッチから直接接続されている別のスイッチについて，自身が中継スイッチとなるものを候補から外す
3. 最もコストの低い順にソートしてトップを選択
4. 2,3 を繰り返す

冗長経路は到達性の維持を最優先とするため，経路のホップ数よりも重なりコストの比重を重くする．ただし複数のリンクが同一のスイッチ間で接続されている可能性を考慮し，同じスイッチを複数回経由しないならば象スイッチの直近が冗長経路として算出されることを許可する．

図 2 において，SW5 が SW4 - SW8 - SW2 の経路を制御経路としている場合を例に挙げる．このとき SW5 は SW6 もしくは SW9 を通る冗長経路を持つが，SW6 の経路の方が経路の重なりが少ないため SW9 が冗長経路として選ばれることはない．また SWA，SWB のようにループ接続を持たないスイッチは計算の対象外とする．

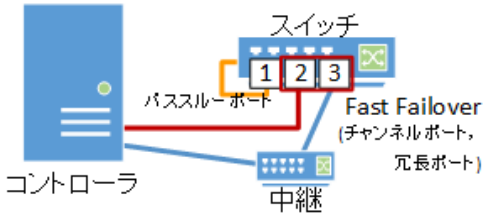

図 3　スイッチの接続構成

表 3　実行環境

ホスト	プロセッサ	Intel® Core™i7-2600　CPU 3.40GHz
	メモリ	16GB
	仮想環境	VMware vSphere5 Hypervisor
仮想	コントローラ，スイッチ	
	OS	Ubuntu 14.04, 12.04 LTS(64bit)
	メモリ	4.0GB, 1.0GB

5.3　OpenFlow チャンネルの修復

5.1，5.2 節によってスイッチからコントローラに対する接続を保障した．しかし OpenFlow チャンネルはスイッチとコントローラ相互に通信可能である必要があるため，コントローラからスイッチに対する各制御エントリーの変更が必要となる．

PortStatus メッセージによってデータネットワークのトポロジに変化があったことをコントローラが確認したとき，各スイッチに対する OpenFlow チャンネル経路の再計算を行い，コントローラからの中継スイッチ数が少ない順にフローエントリーの再設定を行う．コントローラから近い順に OpenFlow チャンネルを再構築しなければ，中継スイッチの制御を失っていた場合，以降のスイッチに対して制御メッセージを転送することができないからである．

6　実装

4 章で述べた手法を，OpenFlow1.3 をサポートしているソフトウェアスイッチで構成した仮想ネットワークに，本手法を実装したコントローラを接続して動作を確認する．1 台のスイッチをコントローラと直接接続し，残りのスイッチはそれまでに接続されたスイッチに対して順繰りに接続していく．最後にあるスイッチから別のスイッチに接続することでループ接続 (図 4) を実現し，予め必要なフローエントリーはスイッチに設定する．コントローラとソフトウェアスイッチにはグループ機能に対応した Trema-edge，ofsoftswitch13 を利用し，これを表 3 の環境で実行した．

また，ofsoftswitch13 は管理下にあるポートから入力されたメッセージを直接処理することはできない．この場合，4 章で述べた通り，スイッチの実装によっては制御チャネルの通信をパスするポートによって，OpenFlow メッセージを処理するカーネルなどのコアシステムへループバックさせる必要がある．そこで本実装では図 3 に示す通り，OpenFlow スイッチの 1 番ポートを利用し，全スイッチで OpenFlow メッセージの交換を成した．

6.1　導通実験

監視には tshark を用い，監視対象となる冗長経路で利用されているイーサアダプタの上を流れるパケットをキャプチャリ

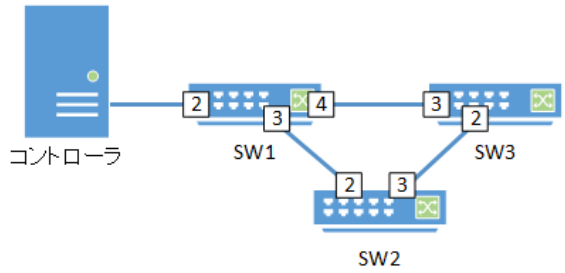

図 4　テスト接続環境

```
stat_repl{type="gdesc", flags="0x0", stats=[{type="ff", group="100", buckets=[{w
="0", wprt="2", wgrp="any", acts=[out{port="2"}]}, {w="0", wprt="3", wgrp="any",
 acts=[out{port="3"}]}]}]}
```

図 5　SW2 グループテーブル

```
stat_repl{type="flow", flags="0x0", stats=[{table="0", match="oxm{in_port="2", e
th_type="0x806", arp_tpa="192.168.250.251"}", dur_s="4", dur_ns="210000", prio="
155", idle_to="0", hard_to="0", cookie="0x32", pkt_cnt="0", byte_cnt="0", insts=
[apply{acts=[out{port="3"}]}]}, {table="0", match="oxm{in_port="2", eth_type="0x
800", ipv4_dst="192.168.250.251"}", dur_s="4", dur_ns="210000", prio="155", idle
_to="0", hard_to="0", cookie="0x33", pkt_cnt="0", byte_cnt="0", insts=[apply{act
s=[out{port="3"}]}]}, {table="0", match="oxm{in_port="1", eth_type="0x800", ipv4
_src="192.168.250.12", ipv4_dst="192.168.250.251"}", dur_s="22", dur_ns="497000"
, prio="133", idle_to="0", hard_to="0", cookie="0x6", pkt_cnt="658", byte_cnt="1
65876", insts=[apply{acts=[group{id="100"}]}]}, {table="0", match="oxm{in_port="
```

図 6　SW2 フローテーブルの一部

ングする．まず初めに，制御経路のために設定されるフローエントリは冗長経路を利用せずにコントローラと通信できることを確認する．

図 5，6 は SW2 のグループテーブルとエントリーテーブルの一部である．このトポロジで SW2 はチャンネルポートで SW1 と，冗長ポートで SW3 と接続されている．図を確認すると，グループの出力ポートが 2，3 の順に設定されており，それぞれチャンネルポートと冗長ポートのポート番号と一致することからグループの設定は正しくなされていることが確認できた．また，in-port が 2 で宛先がコントローラとなるエントリーの出力が 3 番ポートとなっていることから，冗長エントリーも正しく設定されている．SW2 にエントリーが設定されていることから中継するスイッチが制御パケットを転送していることが分かるので，コントローラの SW1 の各エントリーも設定されていることが確認できた．

6.2　障害時のフェイルオーバー

ここで SW1 と SW2 の間で障害が発生したと仮定し，SW2 側で SW1 と接続されているインターフェイスのステータスを変更し，切断状態とすることでスイッチに対してダウンしたことを認識させる．ダウンしたアダプタでパケットの疎通がなくなることを確認し，SW3 の冗長経路に接続されたアダプタを監視する．このとき SW2 と SW3 で生成されたメッセージが確認されたため，SW2 で転送に失敗したパケットが SW3 へ返却され，冗長経路へと再転送されたことを確認できた．

7　考察

本アルゴリズムを適用した OpenFlow ネットワークにおいて，すべてのスイッチがコントローラに認識されるまでの時間は，スイッチ数ではなく，コントローラからスイッチに対する最大の中継数に依存すると考えられる．各スイッチはコントローラの配下になるまで自身に関する制御パケット以外を転送しない．そのため，中継が必要な制御パケットの交換が完了しないからである．その上で，各スイッチの Hello パケットの転送間隔に従って OpenFlow チャンネルの構築時間が決定する．実験に用いたスイッチは最大で 4 秒ごとに Hello メッセージを再送する．実際に，スイッチ 3 台を用意し直列接続した状態で構築完了時間を 11 回計測したところ，それぞれの平均が 3.1，6.7，10.8 秒となったのに対し，図 4 の接続では 3.2，6.9，7.3 秒となり，2・3 台目の接続にほぼ差がなくなっていた．

また，前節では障害が 1 個所の場合を検証したが，障害が同時に複数個所で発生した場合について考察する．各スイッチには OpenFlow チャンネルで利用される通常の経路と，冗長経路を利用した緊急退避用の経路が用意されるため，二つに同時に影響しなければ制御パケットは失われない．具体的に図 2 において，SW1-3 間・SW4-5 間の 2 個所で発生した場合を用いて説明する．この場合，SW8-7-9-5 の経路が生存しているが，各スイッチの OpenFlow チャンネル経路と冗長経路が SW1-3-6-5-4-8-2 に集約されているため，実際には利用されずにループしてしまい，制御を失ってしまう．対処としては冗長経路が複数ある場合に，そのすべてのポートを Fast Failover グループで構成してしまうことが考えられる．SW5 では通常経路が 2 番ポート，冗長経路が 4，3 番ポートに存在するため，これら 3 つをグループ化することによって直近の障害に対しては即座に反応することが可能である．しかし，障害が直近でなかった場合，複数個所からパケットが戻されたことの条件分岐をスイッチが行うことはできないので，事前に設定されたフローエントリーが削除されない限り制御が戻らない．次善策として，スイッチからの制御パケット転送が途絶えたとき，コントローラが把握している別の冗長経路を利用して制御フローエントリーを書き換えることで，制御フローエントリーの削除を待たずに，強制的に復帰させることが挙げられる．

8　まとめ

本研究は，SDN コントロールプレーンをデータネットワーク上に構築する手法について，リンクダウンなどの障害によってトポロジが変化したときに，動的な経路変更によって制御を失わないためのアルゴリズムを提案した．文献 [4] では使用する OpenFlow 装置全体に対して改造が必要であったのに対し，本システムは本来の OpenFlow が備える機能を利用して OpenFlow チャンネルを構築するため，既存のソフトや実機がそのまま利用できることが利点となっている．その上でデータネットワークの冗長性を制御ネットワークが併用できるため，柔軟性と高信頼性を保証できることを示した．

文　献

[1] OPEN NETWORKING FOUNDATION ”Software-Defined Networking: The New Norm for Netwroks White Paper” April 13, 2012.

[2] ”Open Networking Foundation” https://www.opennetworking.org/

[3] ”OpenFlow Switch Specification Version 1.3.4 Implemented (Protocol 0x04)” March 27, 2014

[4] 小出俊夫，下西英之　”OpenFlow ネットワークにおける制御ネットワークの構築自動化に関する一検討”　電子情報通信学会　信学技報 (NS2012-168) p.133-138.