グラフ理論を応用した Software Defined Networking の設計と実装に関する研究

Studies on graph theoretical approaches for design and implementation of Software Defined Networking

07D5204　長野 純一　　　指導教員:　篠宮 紀彦

Abstract

Control planes of Software Defined Networking (SDN) have faced scalability issues because they have been widely applied for large scale networks. It is known that the issues are caused from three points: (1) heavy loads of controllers which dominate all switches in an SDN network, (2) long delay between a controller and a switch and (3) limitation of the number of control rules in a switch. In order to reduce the number of the control rules for failure recovery which is a necessary function of a network, this paper proposes a rule decision method by utilizing the fundamental tie-sets of the graph theory. For alleviating the heavy loads and effects of the long delay, our switch clustering method can construct an effective control plane with multi-controllers sharing global information.

Keywords: Software Defined Networking, failure recovery, fundamental tie-sets, switch clustering.

1　はじめに

近年，インターネットを介して様々なサービスが提供可能となっており，情報通信ネットワークの重要性が増しており，通信データ量も増大している．このため，ネットワークの障害や性能低下が及ぼす影響は大きく，これらを抑えるには，冗長構成を取った信頼性の高いネットワーク設計や，変化に迅速に対応可能な運用，管理が重要となる [1].

しかし，大規模なネットワークの運用管理には多大な労力が必要となる．ネットワーク機器は，多様化する要望を満たすため，様々なプロトコルを実装しており，機器自体の設定項目が多くなっている．さらに，それぞれの機器の設定の依存関係が強く，多くの機器により構成されるネットワークでは，正確な設定は難しい．また，新たに効率的な制御を作成したとしても，実際のネットワークへの導入には，各機器の持つ制御アルゴリズムを変更が必要になってしまう [2].

そこで，ネットワーク機器の制御アルゴリズムや設定を柔軟に変更可能な制御機構として，ソフトウェアによる制御を可能とする SDN (Software Defined Networking) が提唱された [3]．この SDN を実現するプロトコルとして OpenFlow が ONF(Open Networking Foundation）によって標準化されている．これまでのネットワーク機器には，情報の転送経路を決定するネットワーク制御機能と実際に情報を転送するデータ転送機能が同じ機器に組み込まれていたが，SDN では，2 つの機能を分離し，ネットワーク制御機能はコントローラと呼ばれる制御サーバに担当させ，データ転送機能は SDN 対応スイッチにより実行する．この 2 つの機能を分けることにより，コントローラの制御プログラムを変更することで，制御アルゴリズムや機器設定の変更が可能となり，比較的容易にネットワーク制御を変更可能である．

SDN はコントローラへネットワーク制御機能を集中させているため，大規模なネットワークを制御するためには，以下の課題があることが指摘されている [4, 5]．一つ目の課題は，(1) コントローラへの負荷集中である．制御するスイッチ数が多くなり，コントローラの制御能力を超えてしまうと，ネットワーク全体の制御が停止してしまうため，コントローラへの負荷を削減する機構が必要となる．次の課題は，(2) コントローラとスイッチ間の制御チャネルの遅延の影響である．この遅延が大きいと状況変化を把握してから制御が反映されるまで時間がかかってしまう．このため，遅延を短くするためのコントローラの配置や，障害復旧制御などの短時間で対応しなければならない制御は，できるだけメッセージの往復回数を少なくする必要がある．最後の課題は，(3)SDN 対応スイッチに登録可能な制御ルール（フローエントリ）数の制約である．SDN 対応スイッチは，コントローラから登録されたフローエントリに従いパケットを処理する．現存する SDN 対応スイッチは，小容量かつ高速アクセス可能な TCAM（Ternary Content-Addressable Memory）にフローエントリを保存している．登録したフローエントリが TCAM の容量を超える場合，二次的な低速のメモリが使用されるため，転送性能が劣化してしまう．このため，より少ないフローエントリ数による制御の実現が求められる．

そこで，本研究では，大規模なネットワークを制御可能な高信頼な SDN の設計を目的とし，まず，基本的なネットワーク制御である障害復旧制御に着目し，コントローラへの負荷を抑えたフローエントリ数削減手法の提案する．また，コントローラへの負荷を分散し，制御チャネルの遅延の影響を少なくするため，複数のコントローラの制御範囲の決定アルゴリ

ズムを提案する．

2　フローエントリ数を削減する障害復旧方式

情報通信ネットワークでは短時間の障害でも通信サービスを利用するユーザに大きな影響を与えるため，高速かつ効率の良い障害復旧は重要な制御である．高速な障害復旧の実現のためには，一般的に，情報を転送する現用経路に対し，予備経路をあらかじめ作成するプロテクション方式が取られており，障害発生時には経路の切替のみで復旧可能である．

SDN において，プロテクション方式の障害復旧を実現する場合，現用経路をスイッチに登録する際に予備経路も同時に登録するため，現用経路数に従い予備経路用のフローエントリ数が多くなってしまう．SDN 対応スイッチのフローエントリ数には限りがあるため，予備経路用のフローエントリ数を抑えることが重要となる．本節では，現用経路数に寄らない予備経路の作成が可能案障害復旧方式を提案する．この方式では，グラフ理論の基本タイセット系に基づきフローエントリをスイッチへ設定する．

1 章で述べたように，SDN により制御されるネットワークでは，全てのスイッチの制御をコントローラにおいて計算するため，障害復旧制御の計算量を最小限に抑えることが望ましい．また，SDN ネットワークでは，各スイッチとコントローラが制御メッセージを送受信することで，障害発生の通知，経路の変更を行う．SDN ネットワークにおける障害復旧時間は，コントローラとスイッチ間の通信により大きな影響を受けることが報告されている [4]．このため，コントローラへの障害の通知から予備経路がスイッチへ送られるまでの制御メッセージの送受信は，障害復旧の高速化の観点から，最小限に留める必要がある．以下，本節では，2.1 節で基本タイセット系に関する定義を述べ，2.2 節においてタイセットを用いた障害復旧方式の動作概要を述べ，2.3 節において上記要求条件を満たす障害復旧方式の実装方法を示す．

2.1　諸定義

本節ではグラフ理論によって定義される基本タイセット系の概念を述べる．SDN 対応スイッチをノードとし，その集合を V と示す．また，スイッチをつなぐリンクの集合を E と表し，SDN ネットワークをグラフ $G=(V,E)$ として表現する．グラフ $G=(V,E)$ における任意の初等的な閉路をタイセットと呼ぶ．また，グラフ G 内のどのタイセットも部分集合として含まないような極大なリンク集合 $T\subset E$ を G の木とし，木 T に含まれないリンク集合 $\overline{T}=E-T$ を T の補木とする．木 T に含まれるリンクの数 $|T|$ は $|V|-1$ であり，補木 $\overline{T}$ に含まれるリンクの数は $|\overline{T}|=|E|-|T|=|E|-|V|+1$ となる．

また，グラフ G において，補木に含まれるリンク $e\in\overline{T}$ に対して，$T\cup\{e\}$ は一つのタイセット L を必ず含むことが知られており，これを基本タイセットと呼ぶ．全ての補木のリンク $e\in\overline{T}$ に対して作成できる $|\overline{T}|$ 個の基本タイセットの集合を基本タイセット系と呼ぶ．

2.2　動作概要

本方式は，まず，ネットワーク内のノードとリンクの情報をグラフとして把握し，そのグラフを元に基本タイセット系を作成する．次に，基本タイセット系に含まれるタイセットを利用した予備経路を設定する．現用経路上のリンクに障害が発生した場合，障害が起きたリンクを含むタイセットを一つ選び予備経路に用いる．例えば，図 1 のネットワークにおいてタイセット L_1,L_2,L_3,L_4 が作成されているとする．このとき，リンク e_4 に障害が発生した場合，リンク e_4 を含むタイセット L_3,L_4 のどちらかを使用することで障害リンクを迂回する．ここでタイセット L_3 を選択した場合，障害リンク e_4 を通る通信メッセージは，予備経路 (v_3,v_5,v_4) および (v_4,v_5,v_3) を経由することで障害リンクを回避する．

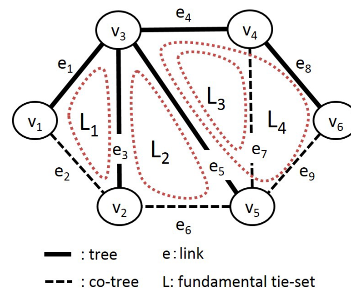

Fig.1　An example of fundamental tie-sets.

2.3　フローエントリ数を削減するための基本タイセット系の算出

本方式は，予備経路のためのフローエントリを各ノードに設定する．作成した基本タイセット系に含まれる各タイセットに沿って予備経路用のフローエントリを登録する．登録される予備経路のためのフローエントリ数は，全タイセットに含まれるリンク数の合計となる．含まれるリンク数が最小となる基本タイセット系を導出することは難しく [6]，コントローラの計算量も考慮し，より簡単に基本タイセット系を求める必要がある．そこで，タイセットに含まれるリンク数の上限が導出に使用する木の深さにより定まることを利用し，タイセットのリンク数の上限値を最小にすることで，タイセットに含まれるリンク数の総計の上限値を抑える方法をとる．よって，フローエントリの少ない予備経路を作成するため，深さが最小の木を用いた基本タイセット系を作成する．

上記動作とフローエントリ数を削減するための処理の計算量は以下のようになっており，多項式時間以内に計算可能である．

- 深さ最小の木を求める O($|V||E|$)
- 木より基本タイセット系を求める O($|\overline{T}||V|$)
- 予備経路を設置する O($|\overline{T}||V|$)
- 発見された障害リンクを復旧する O($|\overline{T}|$)

よって，コントローラの計算量を抑えつつ，フローエントリ数を削減することが可能となる．

2.4　実験結果

OpenFlow を用いた SDN の開発環境 Trema を用い，提案した実装方法に基づいたコントローラを作成した．これを用い大規模ネットワークにおける特性検証実験を行った．実験では，ノード数を 10 から 290 まで 40 刻みで変化させ，木や予備経路を実現するために必要な制御メッセージ数，障害復旧

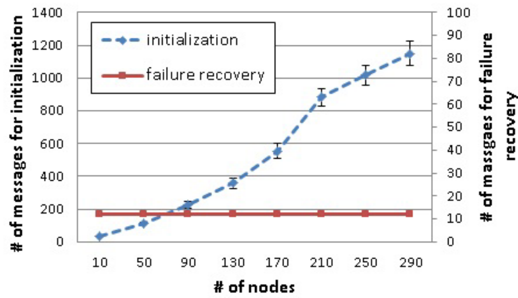

Fig.2　Nodes vs. secure channel massages.

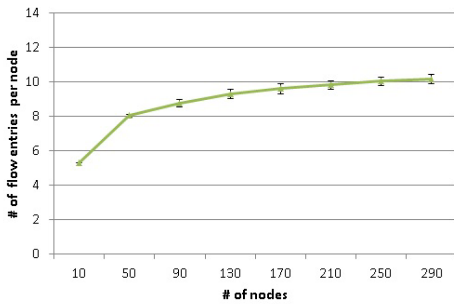

Fig.3　Nodes vs. average of flow entries.

時に必要な制御メッセージ数，フローエントリ数を計測する．

まず，制御メッセージ数を図 2 に示す．initialization は，現用経路と予備経路を作成するためのメッセージ数であり，failure recovery は，障害発生から復旧処理の終了までに送られるメッセージ数である．図 2 の左縦軸は initialization の値を示し，右縦軸は failure recovery の値を示している．障害復旧に必要なメッセージ数は，initialization に必要なメッセージ数に比べ非常に少ない．また，全てのノード数において，一定のメッセージ数で障害復旧できている．

次に，予備経路のための平均フローエントリ数を図 3 に示す．図 3 より，予備経路のフローエントリは，ノードが増加しても，緩やかに増加し，フローエントリ数は 290 ノードでも 10 程度であった．

以上の結果より，提案方式がネットワーク規模に関係なくメッセージ数が一定であることを示し，障害復旧に使用するフローエントリ数も抑えられていることが分かった．

3　複数のコントローラの制御範囲の決定手法

前章において，計算量を抑えた障害復旧方式の実装について述べたが，ネットワークが大規模になったとき多項式時間の計算時間でも現実的な時間で処理が終了しない可能性がある．また，広域なネットワークにおいては，コントローラとスイッチ間の遅延が制御に大きな影響を与える．そこで，ネットワークを複数の部分に分割し，各部分を違うコントローラによって制御することで，計算時間の増加と遅延の影響を抑えることが出来る．この方法では，コントローラ間で共有する情報に不整合があってはならないため，DHT（Distributed Hash Table）などの分散データベースを用いたコントロールプレーンの構築方法が提案されている [5]．

この手法では，分散データベースを用いてトポロジ情報や

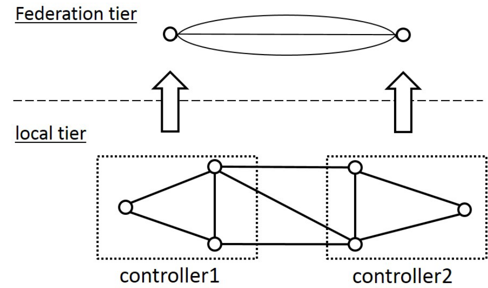

Fig.4　An example of the federation tier and the local tier.

発生したイベント情報が共有されている．しかし，制御するネットワークが大規模になった場合，コントローラ間で共有する情報が膨大となり，各コントローラへの負荷が大きくなってしまう．そこで，Onix[5] は，一つのコントローラがもつトポロジ情報を共有する際，複数のスイッチを一つのスイッチとして分散データベースに登録し，共有情報を削減する方法を提案している．この方法を用いればトポロジ情報を他のコントローラへ隠蔽可能であるが，発生したイベントの情報を公開してしまう可能性がある．

例えば，コントローラの管理するネットワークが木状のトポロジであり，その中にあるリンクに障害が発生した場合，この障害は単一のコントローラによって処理することは出来ないため，この情報は他のコントローラと共有される．一方，コントローラの管理するトポロジがサイクル（2 連結）である場合，任意の単一リンク障害をローカルに復旧可能であるため，他のコントローラとの協調動作は必要ない．そこで，3.1 節では，一つのコントローラが管理するネットワークの 2 連結成分に着目し，ローカルに持つべき情報とグローバルに共有するべき情報を明確にし，共有情報を削減する方法を提案する．

3.1　コントローラ間の共有情報

複数コントローラを用いたネットワークにおいて，各コントローラは，管理下にあるスイッチの制御機能と協調動作によるネットワーク全体の制御機能を持つ．前者をローカルコントローラと呼び，後者をフェデレータと呼ぶ．

前節で述べたように，ローカルコントローラにより復旧可能な障害は，単一コントローラの管理下にある 2 連結グラフ成分内の障害であり，それ以外の成分の障害はフェデレータにより処理される．そこで，各ローカルコントローラは，管理下にあるネットワークのトポロジのみを把握し，フェデレータは全ローカルコントローラ内にある 2 連結成分を一つのノードとし，そのノードの接続関係をリンクとしたフェデレーショングラフを共有する．図 4 にローカルコントローラの持つグラフとフェデレーショングラフの例を示す．各ローカルコントローラは 3 つのノードがサイクル状に接続されたネットワークを管理しており，フェデレータはローカルコントローラのネットワークを一つのノードとし，その接続関係を把握する．

3.2　サイクルクラスタリング

フェデレータグラフのサイズ，つまり，コントローラ間の共有情報の量は，コントローラが管理する範囲の決め方（クラスタリング）によって大きく変化するため，本研究では，フェデレータグラフのサイズを小さくするクラスタリング手法を

提案する．フェデレータグラフ $G^f = (V^f, E^f)$ のサイズは，フェデレータノード数 $|V^f|$ とフェデレータリンク数 $|E^f|$ によるが，元のグラフ G が 2 連結であり，一つのコントローラがネットワーク全体を管理していなければ，$|V^f| \leq |E^f|$ となるため，目的関数は以下のようになる．

$$\text{Minimize } |E^f| = f(G, \mathscr{C}) \quad s.t. |V_i| \leq k, V_i \in \mathscr{C} \qquad (1)$$

ここで，k は一つのコントローラの管理できるノード数の上限であり，$\mathscr{C}$ はクラスタリングを表す空でないノード集合 (クラスタ) の族であり，クラスタ同士は共通の要素を持たず，$\mathscr{C}$ に含まれる全クラスタの和集合は V と等しくなる．つまり，ノードはどれか一つのクラスタに所属し，かつ，二つ以上のクラスタに含まれることはない．

サイクルクラスタリングは，より多くのリンクをクラスタに含めるため，1. 基本タイセット系などの全ノードを被覆するサイクル集合を求め，3. そのサイクルリストからサイクルを取り出し，4. そのサイクルをクラスタに加え，5. 選択したサイクルをサイクルリストから削除する．3.,4.,5. の動作をサイクルリストが空になるまで繰り返す．

1. CycleList $\leftarrow$ list of cycles that cover all nodes in a graph
2. while CycleList $\neq \phi$
3. Select a cycle$L$ which has the smallest $IN(L)$
4. Add the cycle to a cluster V_i
5. Delete the cycle from CycleList
6. end while

3. において，サイクルを選択する基準として孤立点の数 $IN(L)$ を用いる．サイクル L を一つクラスタに入れると，そのサイクルとノードを共有するサイクルは分離されてしまう．$IN(L)$ はこの分離によって生じる，サイクルリスト中のどのサイクルにも属さないノードの数を表す．この数を抑えることにより，多くの 2 連結成分をクラスタ内に入れることが出来る．

3.3 実験結果

サイクルクラスタリング，HCS(Highly Connected Subgraphs) クラスタリング [7]，Connected クラスタリングによるフェデレータグラフのサイズを比較した．Connected クラスタリングは，シンプルなクラスタリング手法であり，単純な木の探索アルゴリズムを用いてクラスタを作成する．実験には，Connected Caveman グラフと Newman Watts Strogatz (NWS) ランダムグラフを用い，k は 15 に設定した．

図 5 より，Caveman グラフにおいて，サイクルクラスタリングと HCS クラスタリングが大きくフェデレータグラフのサイズを削減することができている．また，図 6 より，NWS グラフにおいて，サイクルクラスタリングと Connected クラスタリングがフェデレーショングラフのサイズを抑えることができている．以上より，サイクルクラスタリングは，グラフの規模やグラフタイプに関わらず，共有情報を削減することが可能であった．

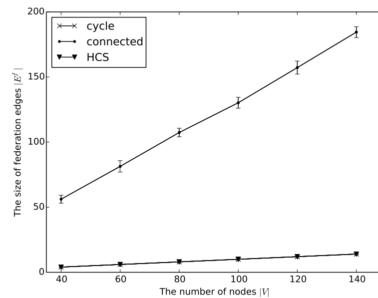

Fig.5 The # of federation edges $|E^f|$ (Caveman graph).

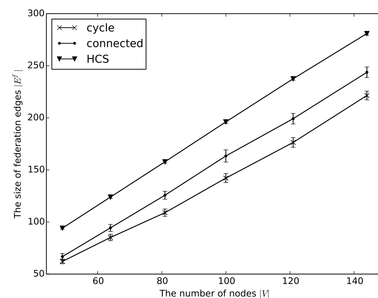

Fig.6 The # of federation edges $|E^f|$ (NWS graph).

4 まとめ

本稿では，まず，大規模なネットワークを制御可能な高信頼な SDN を実現するためには，(1) コントローラへの負荷の削減，(2) コントローラとスイッチ間の制御チャネルの遅延の影響の低減，(3)SDN 対応スイッチに登録するフローエントリ数の削減が必要となることを示した．次に，障害復旧制御に着目し，コントローラへの負荷を抑えつつ，フローエントリ数削減するための方式を提案した．また，この方式を OpenFlow を用いて実装する方法を示し，実際にフローエントリ数を削減可能であることを示した．最後に，制御負荷を分散し，制御チャネルの遅延の影響を少なくすることが可能な分散データベースを用いた制御手法において，複数コントローラ間の情報共有のための負荷を削減するためのサイクルクラスタリングを提案した．実験結果より，共有データベースを用いた協調動作のためのコントローラの負担をサイクルクラスタリングにより軽減可能であった．

参考文献

[1] J.-P. Vasseur *et al.*, *Network Recovery*. Morgan Kaufmann, 2004.

[2] P. Goransson *et al.*, *Software Defined Networks: A Comprehensive Approach*, 2014.

[3] M. Kobayashi *et al.*, *Computer Networks*, vol. 61, pp. 151–175, Mar. 2014.

[4] D. Staessens *et al.*, in *2011 18th IEEE Workshop on LANMAN*, Oct. 2011, pp. 1–6.

[5] T. Koponen *et al.*, in *Proc. of the 9th USENIX Conf. on OSDI'10*, vol. 10, 2010, pp. 1–6.

[6] K. Kadena *et al.*, in *2011 IEEE Workshops of International Conf. on AINA*, Mar. 2011, pp. 467–472.

[7] E. Hartuv *et al.*, *Information Processing Letters*, vol. 76, pp. 175–181, 2000.