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

2016年3月

長野 純一

目 次

図 目 次

第1章　はじめに

1.1　背景

近年，ネットショッピングやSNS (Sosial Networking Service) の利用，動画視聴などのインターネットを介したサービスを多くの人が利用し，ネットワークは我々の生活に密接に関わるようになってきた．これに伴い，ネットワークを流れる情報量が増加しており，図1.1に示すように，2005年から2015年の間，日本では1年で3割程度増加している [1]．また，今後は，スマートフォンやウェアラブルデバイス，車，家電などの多くのモノがネットワークに接続されることにより，より便利で効率的なサービスの提供が可能になってくる．IoT (Internet of Things) やビッグデータの技術の活用により様々なサービス考えられている．例えば，ウェアラブルデバイスや家電などのデータを用いたヘルスケアや，ロボットや車などのセンシングデータを用いた業務や流通の効率化などのサービスが考えられている [2]．このような生活に密着したサービスを支えるため，大量のセンシングデータを転送可能で信頼性の高い情報通信ネットワークの構築が不可欠である．

高速に情報を転送するネットワークでは，短時間の障害や輻輳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制御の実現が求められる [3]．一般的なネットワーク制御方式として，SONET/SDHなどのリング型のトポロジを構築し，運用する技術が存在する [3]．この方式は，単純なネットワーク構成を用いるため，障害が発生した場合においても，通信リンクの両端ノードのみで復旧処理を行ことで，高速に障害を復旧可能である．

また，増え続けるトラフィックに対処するため，分散制御による拡張性の高いネットワークが構築されてきた．分散制御を用いることにより，ネットワークの全体の情報を収集すること無く，部分的な情報のみで，ネットワーク制御を実現することが出来る．分散制御に基づき動作する機

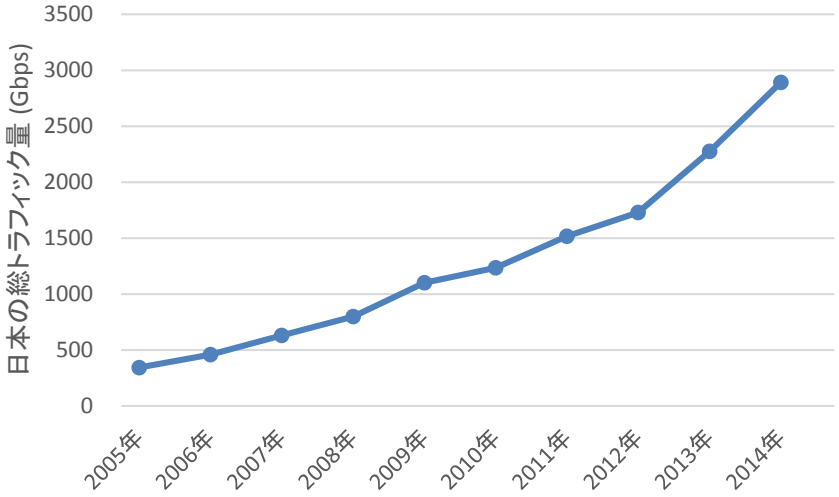


図 1.1: 日本の総トラフィック量の試算 [1]

器は，接続された他の機器とのメッセージ交換により周辺の情報を把握し，自身の動作を決定する．これにより，機器の追加や故障による状況の変化に対応して，機器の動作を変更することが可能となる．大規模なネットワークを制御するためには，効率的なメッセージのやり取りや収束時間の短縮が重要な課題となり，多くの分散アルゴリズムが研究されている [4]．通信ネットワークでは，通信経路を決定する RIP(Routing Information Protocol)[5] や BGP(Border Gateway Protocol)[6] ，通信ループを含まない通信路を構築する STP(Spanning Tree Protocol)[7] などに応用されている．分散制御により新たな機器の追加が容易となり，大規模なネットワークの構築が可能となった．

1.2 集中制御と分散制御

分散制御によるネットワークは 1960 年代後半に構築された ARPANET において初めて採用され，それ以前は，ネットワーク全体を把握するコントロールサーバによる集中制御により運用されていた．ネットワークの制御手法を集中で行うか分散で行うかは，多くの議論がなされてきた．

集中制御では，コントロールサーバがネットワーク全体を把握する事ができるため，容易に最適な制御が実現でき，状況変化への対応も早く行うことが出来る．しかし，コントロールサーバへの負荷が集中してしまうため，ネットワークの規模を大きくすることが難しい．

それに対し，分散制御は，ネットワーク全体を把握するサーバは存在せず，機器全てが周りの状況に従い処理を決定するため，ネットワークの大規模化に強い．しかし，集中制御と比べ制御への制約が多く，制御の収束までに時間がかかってしまう可能性がある．集中制御と分散制御には，利点と欠点があり，制御するネットワークの特性に合わせて制御方法を選ぶことが望ましい．現在では，IP ネットワークが分散制御を用いており，光ネットワークでは集中制御が用いられている．

1.3 Software Defined Networking

ネットワークを介するサービスの増加に伴い，セキュリティや通信品質の確保などの観点からネットワークへの要求が多様化してきた．分散制御を前提として発展してきたネットワークでは，多様化する要望を満たすため，様々なプロトコルが実装されており，各機器の設定項目も多くなっている．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，制御変更のための機器設定には多大な労力が必要となる [8]．また，ネットワーク経由でサービスを提供するクラウドコンピューティングの進展に伴い，サービスを提供するサーバが集まるデータセンタに多くの通信が集中するようになった．データセンタ内のネットワークは，広域ネットワークに比べて障害のような計画しないネットワークの変化は起きにくいが，ルータは 30 %の処理能力を使って変化の少ないネットワークの経路最適化を行っている [9]．そこで，ネットワークを効率的に管理する新たな制御機構が検討されるようになった．

新たな制御機構として，ネットワークの状態を集中管理し，ソフトウェアによる柔軟な制御を可能とする SDN（Software Defined Networking）が提唱された [10]．SDN のネットワーク制御モデルを図 1.2 に示す．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の変更を容易にしている．この SDN を実現

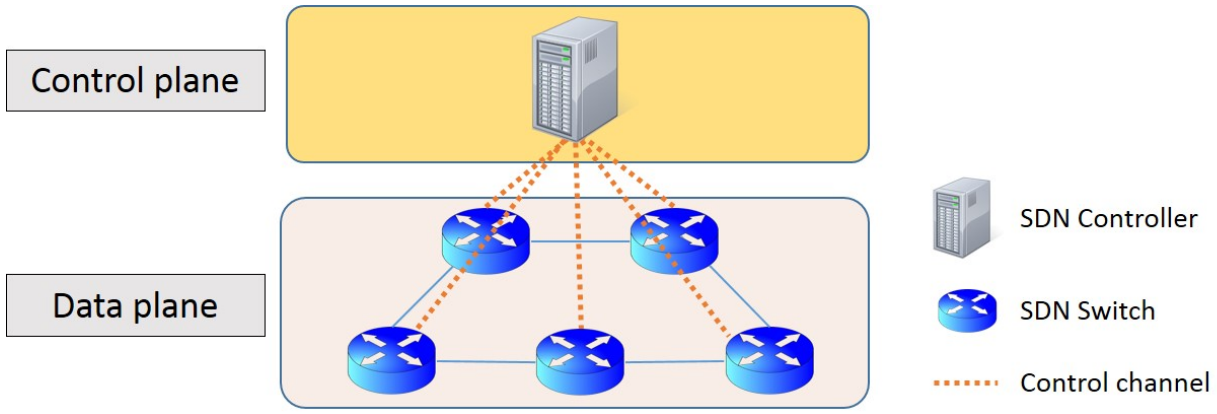


図 1.2: SDN の概要

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．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，集中制御のデメリットである大規模なネットワークの制御に，以下のような問題があることが指摘されている [11, 12]．

1. コントローラへの負荷集中

   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．

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

   コントローラとスイッチ間の遅延が大きいと，対処しなければならない状況変化が発生してから制御の変更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．

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

   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．

SDN が抱える集中制御による問題点に対し，分散データベースを活用した対策が提案されている．これからのネットワーク制御は，分散制御か集中制御かといった単純な二者択一ではなく，実現するべきことに合わせて部分的なネットワークの制御を選択可能となるべきである．

1.4 研究の目的

SDN を用いたネットワークの実装が進んでおり，Google のデータセンタを結ぶ専用ネットワークのトラフィックエンジニアリング [13] のような比較的大規模なネットワークへの適応が進められてきた．しかし，今後，ネットワークに接続される端末が増え，トラフィックが大きく増加していくと，集中制御による問題点が無視できなくなってくる．そこで，本論文では，大規模なネットワークを制御可能な高信頼な SDN の設計を目的とする．コントローラへの負荷集中と制御チャネルの遅延の影響，制御ルール数の制約に着目し，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，および，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の複数コントローラの制御範囲の決定手法を提案する．

第2章　関連技術

2.1　データセンタネットワークの現状

クラウドコンピューティングの進展やスマートフォンなどのモバイル端末の増加により，多くの通信がデータセンタに集中している．図 2.1 に示すように，データセンタ内ネットワークでは，サーバが格納されているラックが Aggregation スイッチにつながっており，Aggregation スイッチはより高速に処理が可能な Core スイッチに接続されているという単純な構成となっている．ラック内のサーバは ToR (Top of Rack) スイッチに接続され，相互にアクセスが可能となっている．1 つのデータセンタには，12 万以上の物理サーバが設置されており [9]，それぞれの物理サーバに最大で 20 台程度の仮想サーバを起動することが出来るとすると，240 万台ものサーバを動作させることが出来る．

データセンタのネットワークにより転送される通信には，外部からサーバへアクセスする通信，サーバから外部へのデータの送信に加えて，サーバ間でやり取りされる通信がある．このサーバ間の通信量が非常に多くなってきており，データセンタのトラフィックの 80 %になると予測されている [14]．

それぞれのサーバは，信頼性の高いリンクで接続されており，障害発生の可能性は，広域ネットワークに比べて低く，トポロジの変化も少ない．ネットワークのトポロジを変更する場合には，計画的に変更が行われる．データセンタのルータは，分散制御により動作しているため，管理された変化の少ないネットワークにおいて，常に経路を最適化している．この最適化のために，ルータの CPU の 30 %が使われていることが指摘されている [9]．

データセンタネットワークは，広域ネットワークとは違った特徴を持つネットワークであり，以下の様な特徴を持つ．これらの特徴から，より効率的な制御が求められるようになってきた．

- トポロジの安定性

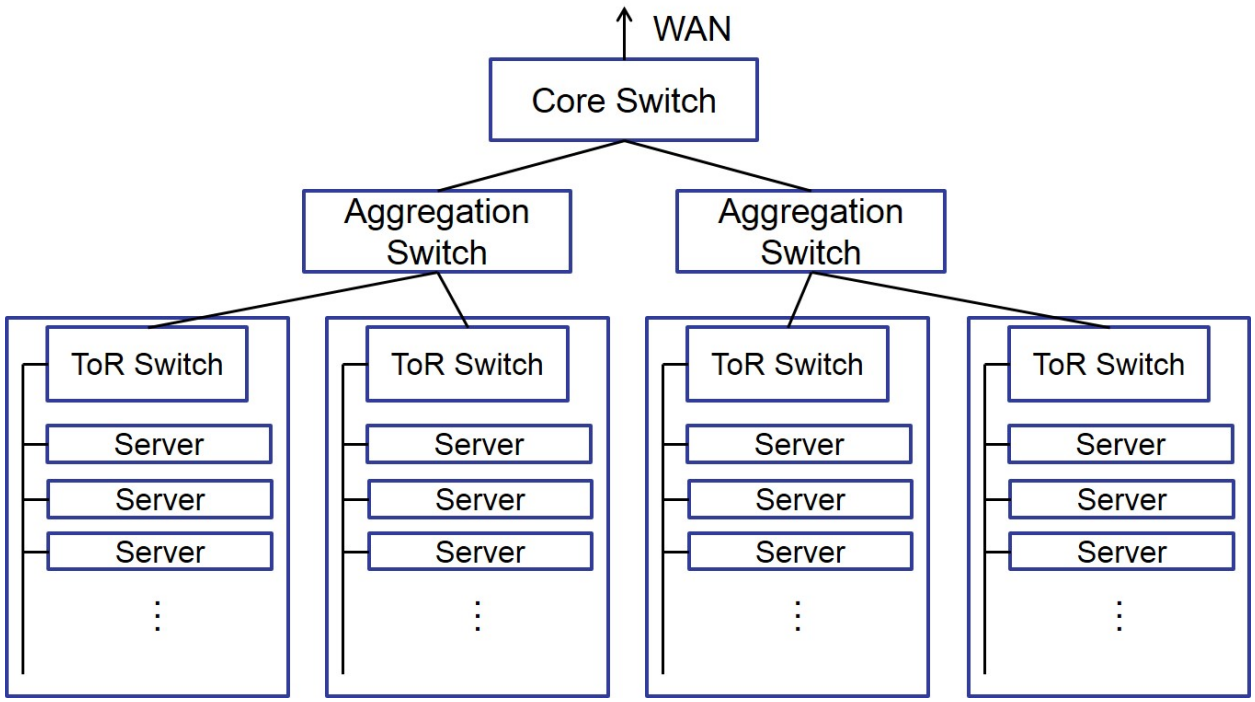


図 2.1: データセンタのネットワーク

- トラフィックパターン
- 接続ホスト数の多さ

2.2 Software Defined Networking

ネットワークが広く利用されるようになったため，ネットワークへの要求が多様化してきたため，情報通信ネットワークは通信キャリアやデータセンタなどを運用する組織では，QoS（Quality of Service）やセキュリティなどのポリシーを策定し，施行している．これらのポリシーでは，音声通話などのネットワーク品質の影響を受けやすいアプリケーションを優先的に転送することや，企業の情報資産を守るために外部との通信は全て検疫サーバを通さなければならないなどを策定している．ポリシー実施のためには，ルータやスイッチなどのネットワーク機器に対し制御ルール，例えば，フォワーディングテーブル，パケットフィルタリング，及び，キューイングなどを作成する必要がある．しかし，現在のネットワークでは，ポリシーを実現するために多くの制御ルールが必要になってしまうことが指摘されている [15]．さらに，ネッ

トワーク機器に実装されているプロトコルを変更することは容易ではなく，機器の制約により，ポリシーの実現が不可能な場合もある [8].

そこで，これまでのネットワーク制御にとらわれない，新たなネットワーク制御が検討されてきた [16]．Open signaling (OPENSIG) working group[17] は，1995 年より，オープンで拡張性の高いプログラマブルなネットワークが必要であると述べている．プログラマブルネットワークの実現のためには，ネットワーク機器から制御ソフトウェアを分離し，ネットワーク機器にオープンなインターフェースでアクセス可能にすることが必要となる．また，機器の制約の少ないネットワークサービスを実現するためにネットワーク機器を外部プログラムにより制御する Active network が提唱された [18]．しかし，Active network はセキュリティやパフォーマンスなどに問題が有り，これまで実現されてこなかった [19]．2004 年には，4D approach [8] というネットワークのグローバルビューを用いたプログラマブルネットワークが提案されている．この手法では，グローバルビューを持つコントローラがフローの管理やグローバルビューの維持を行うことで実現される．この手法がグローバルビューを用いているため，この論文により提案されたプログラマブルネットワークが SDN の始まりだと認識されている．

図 2.2 に 4D approach の制御アーキテクチャを示す．4D approach は，Decision plane による集中型のネットワーク制御を提案しており，Decision plane により，ロードバランスやアクセス制御，セキュリティなどの設定の全てが決定される．Dissemination plane の役割は，Discovery plane で発見された情報を Decision plane に送り，Decision plane で決定された制御を data plane に伝えることである．また，Discovery plane は，新たに接続されたネットワーク機器や接続リンクを発見し，ID を付与し，その特性を調べる．Discovery plane によって解明された情報は，Decision plane により nework-wide view を作成するために使用される．Data plane は，到着したパケットを Decision plane の決定に従い処理する．

2.2.1 OpenFlow

SDN を実現するプロトコルである OpenFlow[20] は，ONF(Open Networking Foundation）において，標準化されているプロトコルであり，SDN と同様，OpenFlow を用いて動作するネットワークは，ネットワーク制御機能を提供するコントローラと，転送機能のみを提供する複数

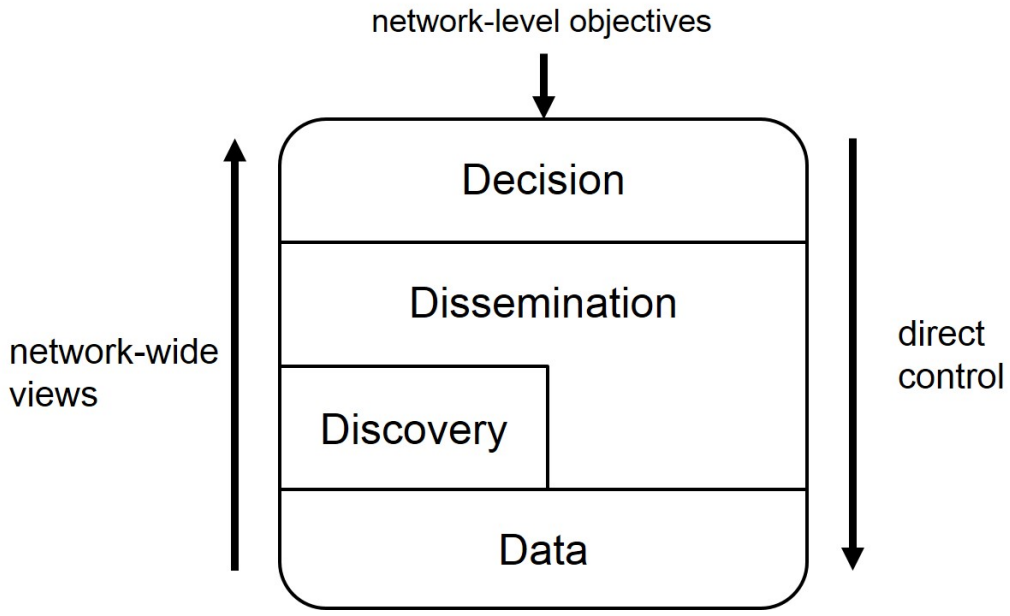


図 2.2: 4D architecture [8]

の OpenFlow スイッチで構成される．コントローラと OpenFlow スイッチは，オープンフローチャネルと呼ばれる制御用ネットワークで接続されており，OpenFlow プロトコルに従い制御メッセージがやりとりされる．OpenFlow は，制御の変更が難しい既存のネットワークに変わり，新たに研究，開発された制御手法を容易に実装可能にすることを目的に提案され，スタンフォード大学の一つのビルのネットワークを OpenFlow を用いて構築することで実証実験が行われた [20]．OpenFlow スイッチは，コントローラから受け取った転送制御ルール（**フローエントリ**）に従いパケットを処理し，フローエントリの存在しないパケットを受信した場合にはコントローラへそのパケットの情報を送信する．コントローラは OpenFlow スイッチから送られてくる情報を元にフローエントリを作成し，OpenFlow スイッチへ送信する．フローエントリは，ルール，アクションを含み，ルールに適合したメッセージに対し，アクションが実行される．ルールには，表 2.1 に示すように，メッセージを受け取ったポートの情報，Ethernet ヘッダ，IP ヘッダ，TCP ヘッダを使用することが出来る．アクションでは，ポートを指定したメッセージの転送やヘッダ情報の書き換え等が可能である．

NOX[21] や Maestro[22], OpenDaylight[23]，Trema[24] などのコントローラを作成するためのフレームワークが提案されている．しかし，SDN において，コントローラに障害が発生した

表 2.1: OpenFlow のルール

In Port	VLAN ID	Ethernet			IP			TCP	
		Src	Dst	Type	Src	Dst	Proto	Src	Dst

り，コントローラがパフォーマンスボトルネックになってしまった場合，SDN は機能しない．そこで，Maestro ではコントローラの処理を並列化することによりコントローラの性能を向上させる手法を提案しているが，1 章で述べたとおり，OpenFlow ネットワークを構築するためには以下の課題があることが知られている．

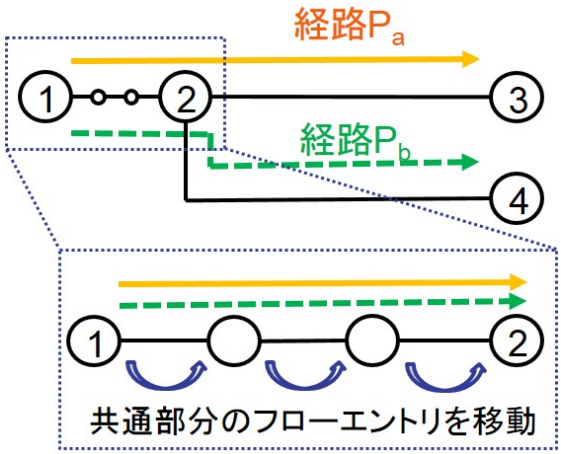
1. コントローラへの負荷集中
2. コントローラとスイッチ間の制御チャネルの遅延の影響
3. SDN 対応スイッチに登録可能な制御ルール（フローエントリ）数の制約

2.2.2 フローエントリ数の制約を満たすための制御

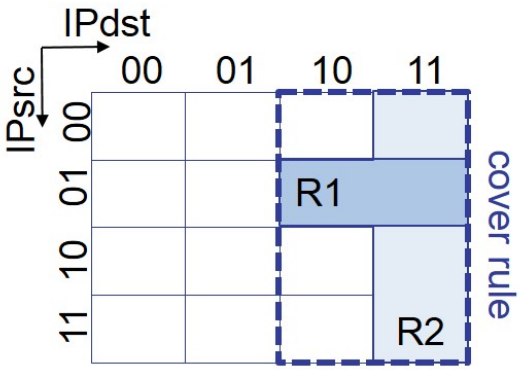
フローエントリ数の制約を満たしつつ，目標とする制御を実現するため，フローエントリの作成方法 [25, 26] が提案されている．Kang ら [26] は，目標とする制御（フロールール）を満たしつつ，フローエントリの移動を可能とする手法を提案している．フロールールを表 2.2 に示すアクセスコントロールポリシーと図 2.3(a) の上部に示す P_a や P_b のようなルーティングポリシーに分割している．ルーティングポリシーは，データを通す経路を示しており，ルーティングポリシーが共通して使用する経路上においてフローエントリを移動することが出来る．図 2.3(a) の P_a や P_b の場合，ノード 1 と 2 の間の区間でフローエントリを移動することが出来る．移動のために使用するのが，図 2.3(b) に示す cover rule であり，複数のアクセスポリシー包含するようなルールを用いることで，フローエントリの移動が可能となる．図 2.3(a) のノード 1 に入れるべきであった R1 と R2 用のフローエントリを cover rule 一つにし，R2 を Drop する場所をノード 1 以降に移すことが出来る．しかし，この手法では，共通部分の少ないフロールールを作成してしまうと，フローエントリ数を削減することは出来ない．そこで，フローエントリを削減するようなフロールールの作成方法が必要となる．

表 2.2: アクセスコントロールポリシー

Name	Src IP	Dst IP	Permission
R1	11	1*	Permit
R2	*	11	Drop



(a) ルーティングポリシーとフローエントリの移動

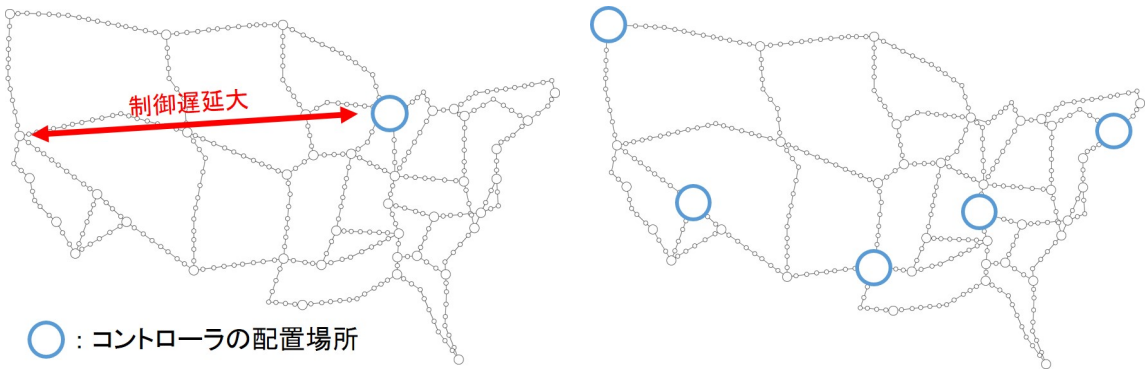


(b) アクセスコントロールポリシーと cover rule

図 2.3: フローエントリ数の制約を満たすための制御

2.2.3 複数コントローラによるコントロールプレーン

単一コントローラによる実装では，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 [27, 28, 29]．また，図 2.4(a) に示すように，単一のコントローラにより広域ネットワークを制御する場合，コントローラとスイッチの遅延が大きくなり，スイッチの制御が難しくなる．通信遅延の影響を少なくするため，図 2.4(b) に示すように，複数のコントローラ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 [30]．複数のコントローラを用いる場合，図 2.5 に示すように，コントローラ間で制御情報を共有する必要があり，SDN の利点であるプログラマビリティを活かしつつ，スケーラビリティを確保するため，DHT（Distributed Hash Table）などの分散データベースを用いた制御方法が提案されている [12]．これらの研究では，ネットワークが大規模になった場合，トポロジ情報や発生したイベント情報等の共有情報が増大してしまう可能性がある [27]．



(a) Single controller　　(b) Multi controllers

図 2.4: 広域ネットワークにおけるコントローラの配置

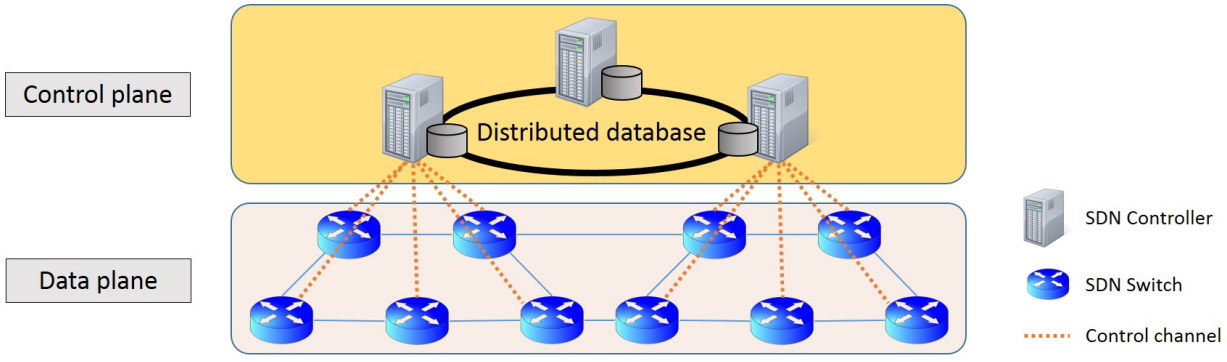


図 2.5: 分散データベースを用いたコントロールプレーン

第3章　諸定義

この章ではネットワークの効率的な制御を考案する上で重要となるグラフ理論について，本論文で用いる定義を述べる．

3.1　グラフ

コントローラにおいて，制御するスイッチの接続関係をグラフとして表す．コントローラがネットワークの一部のスイッチを制御する場合，コントローラはネットワーク全体を表すグラフの部分グラフを保持する．以下に，グラフと部分グラフの定義を示す．

定義 3.1. グラフ

ルータやスイッチ等のネットワーク機器をノード集合 V とし，ネットワーク機器をつなぐリンクの集合を E と表し，情報ネットワークをグラフ $G = (V, E)$ として表現する．

定義 3.2. 部分グラフ

グラフ $G = (V, E)$ と $G' = (V', E')$ があり，$V' \subseteq V, E' \subseteq E$ である場合 G' を G の部分グラフと呼ぶ．また，部分グラフのノード集合を $V(G')$ とし，リンク集合を $E(G')$ と表す．

定義 3.3. 誘導部分グラフ

グラフ $G = (V, E)$ において，ノード集合 $V' \in V$ が与えられたとき，V' に含まれるノードを両端点に持つ全てのリンクの集合を E' とする．G の部分グラフ $G' = (V', E')$ を誘導部分グラフと呼び，$G' = G[V']$ と表す．

定義 3.4. グラフの縮約

グラフ $G = (V, E)$ において，ノード集合 $V' \in V$ を一つのノードにまとめることをグラフ G をノード V' で縮約するという．V' に含まれるノードを両端点に持つ全てのリンクは除去され，一つの端点が V' に含まれるリンクは縮約されたノードを端点とする．

任意のノード集合 U が与えられたとき，グラフ G より U に含まれるノードとそれらのノードに接続されているリンクを除去することがある．このときに得られるグラフを $G-U$ と表す．表記の簡略化のため，$U=v$ のように，U に含まれるノードが一つの場合は，$G-v$ とする．

定義 3.5. ノードの次数

ノード v に接続されているリンクの数をノード v の次数といい，$\delta(v)$ と表す．

3.2 パスと連結度

2 つのノード間で通信データを転送する場合，コントローラにおいて 2 つのノード間を結ぶ通信路を作成する必要がある．この通信路をパスとし，以下のように定義する．

定義 3.6. パス

グラフ G において，$l+1$ 個のノード $v_0, v_1, \ldots, v_l$ と l 個のリンク $e_1, e_2, \ldots, e_l$ を交互に並べた系列 $P=(v_0, e_1, v_1, e_2, v_2, \ldots, e_l, v_l)$ において，全ての $i(1 \leq i \leq l)$ に対して，$e_i=(v_{i-1}, v_i)$ であるとき，この P をノード v_0 からノード v_l へのパスという．v_0 を P の始点と呼び，v_l を P の終点と呼ぶ．また，同じノードを 2 度以上通らないパスを初等的なパスという．

信頼性の高い通信を実現するためには，データを流すパス上のノードやリンクに障害が発生した場合，代替となる別のパスを算出する必要がある．あるパス上の始点と終点以外の任意のノードやリンクに障害が発生しても，代替パスとして利用可能なパスを点素パスといい，以下のように定義する．

定義 3.7. 点素パス

2 本の初等的なパス P_1，P_2 があるとき，それらのパスが始点と終点以外のノードを共有しない場合，P_1 と P_2 は点素である．

グラフ全体の信頼性を表す連結度という指標があり，以下のように定義される．

定義 3.8. 連結度

グラフ G において任意の異なるノード s，t に対し，s を始点とし t を終点とする k 本（k は正整数）の点素なパスが存在する場合，グラフ G は k 連結であるという．これを $\kappa(G)=k$ と表す．なお，$k=1$ のときは単に G は連結であると表す．

3.3 木と基本タイセット系

定義 3.9. タイセットとサイクル

グラフ G において，始点と終点が同じであり，それ以外のノードを 2 度以上通らないパスを閉路と呼び，ある閉路に含まれるリンクの集合をタイセットと呼ぶ．また，ある閉路を構成する全ノードと全リンクのみから成る部分グラフをサイクルと呼ぶ．

定義 3.10. 木と補木

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 の木とする．また，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．

定義 3.11. 基本タイセット系

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．

基本タイセット系は木と一対一対応の関係にあり，この系に含まれるタイセット L のサイズ $|L|$ は，対応する木の深さ（起点となるノードから最も離れているノードまでのリンク数）を d としたとき，$2 \times d + 1$ 以下となることが分かっている．

第4章　OpenFlowを用いた効率的な障害復旧制御の実現

本章では，ネットワークの基本機能である障害復旧制御を集中型のOpenFlowネットワークに実装する際の要求条件を整理し，その条件を満たす実装方法を提案する．

4.1　障害復旧制御

これまで，信頼性の高いネットワークを実現するために，リング型のトポロジを構築し，運用する技術が存在する [3]．しかし，上記方式は，複数のリングが連なった形状に限って適用可能なため，複雑なネットワークにおいてトポロジに内在するリング構造の集合，すなわち，サイクル集合を求め，このサイクル単位でネットワークを運用する手法が提案されている [31, 32]．これらの手法では，予備帯域を減少させるため，可能な限り少ないサイクル数によりネットワーク内の全てのリンクを被覆することを目標としている．しかし，この問題を解くには膨大な計算量が必要になるため，近似解法が提案されているが，ネットワークの規模や形状などの条件によって計算時間が長くなってしまう．さらに，これらの手法により求まるサイクル集合は，サイクルサイズ（各サイクルに含まれるリンク数）を大きくする必要がある．このようなサイクルは予備経路を長くしてしまう可能性が高く，障害発生時のメッセージ転送に関わるノード数が多くなり，遅延も大きくなることが懸念される．

これまで，小出らにより全てのリンクを被覆し，かつ，多項式時間内で計算可能な基本タイセット系をネットワーク管理に用いる手法が提案されている [32]．さらに，中山らによりサイクル単位の自律分散制御によって，障害復旧や混雑回避を可能にする制御アルゴリズムが提案されている [33]．タイセット毎の分散制御をフローエントリを用いて表現することができれば，コントローラの負荷をスイッチへオフロードすることが可能となるため，本研究では基本タイ

セット系を用いた障害復旧手法を用いる．

4.2 障害復旧制御を実装するための要求条件

次に，OpenFlow を用いて障害復旧制御を実現するための要求条件を整理する．キャリアネットワークやデータセンタ内のネットワークは多くのユーザにサービスを提供しているため，ネットワーク機器に障害が発生した場合には，迅速な復旧が必要となる．障害発生から復旧までの時間は，50ms と非常に短く抑えるべきだと言われている [34]．

要求 1　コントローラの計算量

集中制御による OpenFlow ネットワークにおいて，全てのスイッチの制御はコントローラが計算するため，障害復旧の高速化の観点から，障害復旧制御の計算量は最小限に抑えることが望ましい．

要求 2　障害通知からの制御メッセージ数の抑制

OpenFlow 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．また，OpenFlow 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 [11]．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，最小限に留める必要がある．

要求 3　予備経路のフローエントリ数の削減

OpenFlow 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．高速な障害復旧の実現のためには，現用経路と予備経路の両方のフローエントリをスイッチへ登録する．予備経路のフローエントリは，通常使用されないものであるため，より少ないフローエントリ数で予備経路を実現することが求められる．

以下では，4.3 節で，タイセットを用いた障害復旧方式の動作概要を述べ，4.4 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．

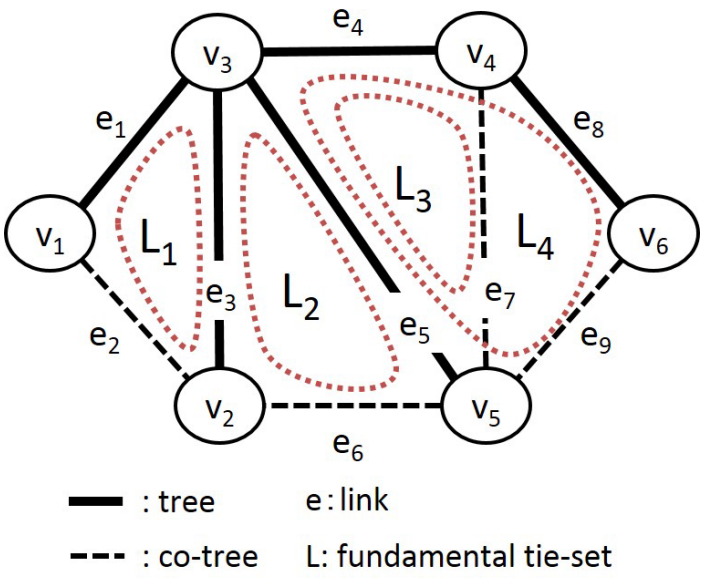


図 4.1: 基本タイセット系の例

4.3 基本タイセット系を用いた障害復旧の動作概要

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 4.1 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．

以下に，障害復旧の準備処理と復旧処理を列挙する．

- 事前準備
  1. トポロジの把握
  2. 木および基本タイセット系の作成
  3. 予備経路を実現するフローエントリの作成
- 障害時

1. 障害リンクの検出
2. 復旧用のタイセットの選択
3. 現用経路から復旧経路へ切り替えるためのフローエントリの作成

4.4 要求条件を満たすための実装方法

本節では，まず，要求条件を満たすために必要な実装について概略を述べ，その後に 4.3 節で示した各動作に対しての実装方法を述べる．

本研究では，要求 1 と要求 2 を満たすため，基本タイセット系の特徴を活かし，全リンク障害に対応可能な予備経路用のフローエントリを各ノードにあらかじめ設定する．これにより，障害時のコントローラの計算負荷とメッセージ数を削減する．このためには，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必要がある．フローエントリの作成方法の詳細は 4.4.3 に示す．

また，要求 3 を実現するため，予備経路用にスイッチに登録されるフローエントリ数を削減する．予備経路のための総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 [35]，要求 1 の計算量の制約も考慮し，より簡単に基本タイセット系を求めるため，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，要求 3 に対して，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．

上記動作と要求 2,3 を満たす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 $O(|V|^2 + |V||E|)$（4.4.2 項参照）
- 木より基本タイセット系を求める $O(|\overline{T}||V|)$（4.4.2 項参照）
- 予備経路を設置する $O(|\overline{T}||V|)$（4.4.3 項参照）

- 発見された障害リンクを復旧する O($|\overline{T}|$)（4.4.4 項参照）

よって，要求 1 を満たしつつ，要求 2,3 を実現するための実装が可能となる．

4.4.1 トポロジの把握

ここでは，ネットワークのトポロジを把握する方法について述べる．基本タイセット系を用いた障害復旧手法は，特別なトポロジの把握方法を必要としない．そのため，ネットワークの状態把握には，NOX や Trema でソースコードが公開されている LLDP（Link Layer Discovery Protocol）パケットを使用する方法を流用可能である．これらの方法では，コントローラはスイッチが接続された際に，そのスイッチの全てのポートに対して送信元のスイッチ ID とポート番号を格納した LLDP パケットを送出する．その後，LLDP パケットは，パケットを受信したスイッチの ID とポートの番号の情報を付加され，コントローラへ転送される．これらの情報を元に，コントローラはスイッチの接続関係を把握することが出来る．

4.4.2 木および基本タイセット系の作成

ここでは，要求 1 の制約を満たしつつ，要求 3 で望まれるフローエントリ数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の作成に要する計算量について述べる．

深さが最小の木を求めるためには，BFS（Breadth First Search）という単純な木の探索法（BFS の計算量は O($|V|+|E|$)）をノード数 $|V|$ 回繰り返せば良い．これに必要な計算量は O($|V|^2+|V||E|$) であり，多項式時間に抑えられている．

また，基本タイセット系を木より導出する方法について述べる．木のリンク集合に補木のリンクを一つ加えると，その集合には必ず一つ基本タイセットが含まれるという特徴を利用し，基本タイセット系を作成する．補木のリンク一つを取り出し，その片方のノードから木のリンクをたどり，もう一方のノードへのパス状のリンク集合を作成する．この作成した集合に取り出した補木のリンクを加えると，基本タイセットを作成することが可能である．全ての補木のリンクに対して，基本タイセットを作成すれば，基本タイセット系が作成可能である．

よって，基本タイセット系を作成するためには，補木 $\overline{T}$ に含まれる一つのリンクに対して，一つの基本タイセットを作成するための計算量は多くとも $O(|V|)$ である．基本タイセット系を作成するために必要な計算量は $O(|\overline{T}||V|) = O(|V||E| - |V|^2)$ となり，多項式時間で求めることが出来る．

4.4.3 予備経路を実現するフローエントリの作成

4.3 節で示したように，タイセットを用いた障害復旧手法はタイセットに沿って予備経路を作成することで障害リンクを回避する．このため，予備経路を実現するフローエントリを作成するためには，タイセットに沿ってパケットを転送するフローエントリを作成する必要がある．しかし，要求 3 より，現用経路一つに対し予備経路を一つ作成することはできないため，複数の現用経路を一つの予備経路に集約可能なフローエントリを作成する必要がある．これを実現するため，集約に使用する通信メッセージへの付加情報とスイッチでの転送テーブルの作成方法を考える必要がある．

通信メッセージへの付加情報（タイセットヘッダ）

現用経路のネットワークフローを集約するためヘッダ（タイセットヘッダ）を作成する．タイセットヘッダは，各基本タイセットに付与される ID（タイセット ID）と回転の向きを含む．予備経路では，このヘッダのみを参照してメッセージを転送する．また，リンク障害時にメッセージを現用経路から予備経路に切り替える際，タイセットヘッダをメッセージに格納し，現用経路へ戻す場合にはこれを取り除く．簡単のため，タイセットに回転の向きを任意の方法で定め，順方向を Forward とし，逆方向を Backward と表す．タイセットヘッダを保存するフィールドは，現用経路のマッチ条件に使用しないフィールドを使用する．

ノードの転送テーブル

各ノードは内部に現用経路の転送テーブルと予備経路用のタイセットテーブルを保持する．現用経路の転送テーブルは送信先となるホストや外部ネットワークへのアドレスを検索キーと

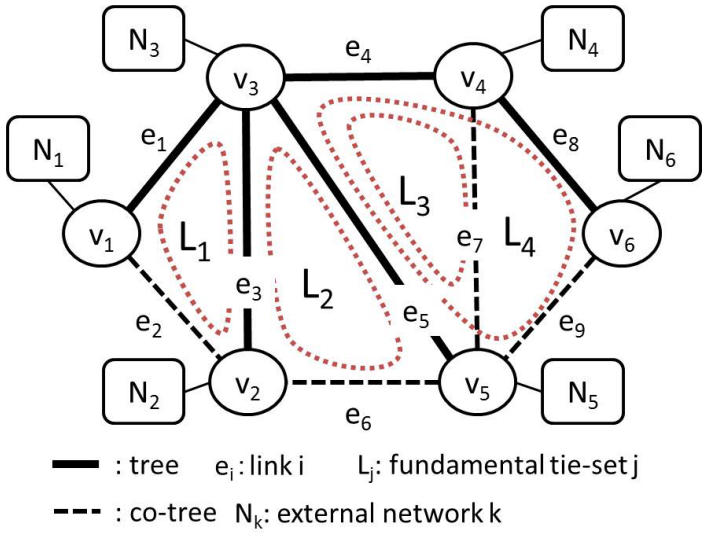


図 4.2: サンプルネットワーク

し，メッセージを転送するポートの番号を返す．図 4.2 のネットワークにおけるノード v_4 が図 4.3 のようなポート番号を持つ場合，ノード v_4 が保持する転送テーブルを表 4.1 に示す．

タイセットテーブルは，メッセージに付加されたタイセット ID とその回転の向きを検索キーとし，予備経路への転送ポート番号とメッセージに付加されたタイセットテーブル情報ををり除くかどうかを返す．各ノードは，自ノードを経由するタイセットに関する情報のみを保持する．例えば，図 4.3 のノード v_4 はタイセット L_3, L_4 のテーブルのみを保持する．また，タイセット上をメッセージがループすることを防止するため，送出先リンクが補木の場合はタイセットヘッダを削除する．上記したことを踏まえて図 4.3 におけるノード v_4 が持つタイセットテーブルを表 4.2 に示す．なお，L_3, L_4 の回転の向きは，図 4.3 における時計回りとした．ノード v_4 の p_3 が補木のリンクへ接続されているため，L_3 Forward をキーとするタイセットテーブルは，タイセットヘッダを削除している．その他のタイセットテーブルは転送先ポートが木のリンクであるため，タイセットヘッダを削除せず, パケットの転送先ポートのみを返す．

以上のようなタイセットヘッダ，テーブルを用いたノードのメッセージ転送処理を図 4.4 に示す．メッセージを受け取ったノードはタイセットヘッダに自ノードが含まれるタイセットが格納されていない場合，現用経路の転送テーブルに従ってメッセージを転送する．そうでない場合，タイセットテーブルに従ってメッセージを転送する．だだし，送信先のリンクが補木リンクの場合，タイセットヘッダを削除し，メッセージを転送する．

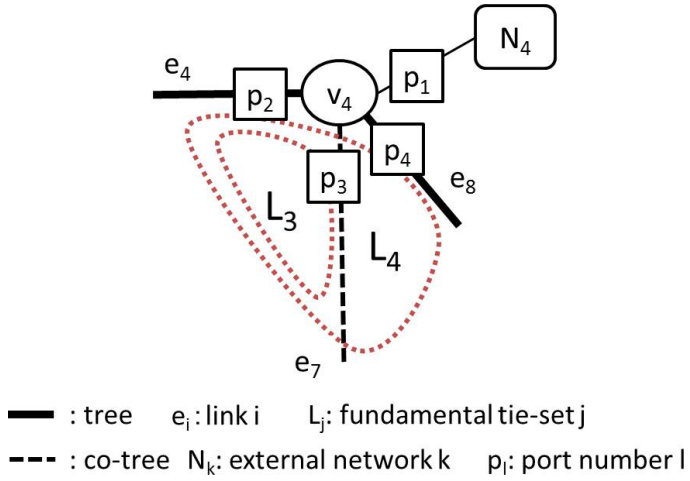


図 4.3: ノード v_4 のポート番号

表 4.1: ノード v_4 の現用経路のテーブル

Key (destination address)	Output port
N_1	p_2
N_2	p_2
N_3	p_2
N_4	p_1
N_5	p_2
N_6	p_4

表 4.2: ノード v_4 のタイセットテーブル

Key (tie-set ID)	Action	Output port
L_3 Forward	strip tie-set header L_3 Forward	p_3
L_3 Backward	none	p_2
L_4 Forward	none	p_4
L_4 Backward	none	p_2

4.4.4 障害時の動作の実装

障害時の各動作（障害リンクの検出，復旧用のタイセットの選択，現用経路から復旧経路へ切り替えるためのフローエントリの作成）について，以下に手順を示す．この手順はネットワーク $G(V, E)$ においてリンク $e_f \in E$ 上に障害が発生した場合を想定している．またリンク e_f に

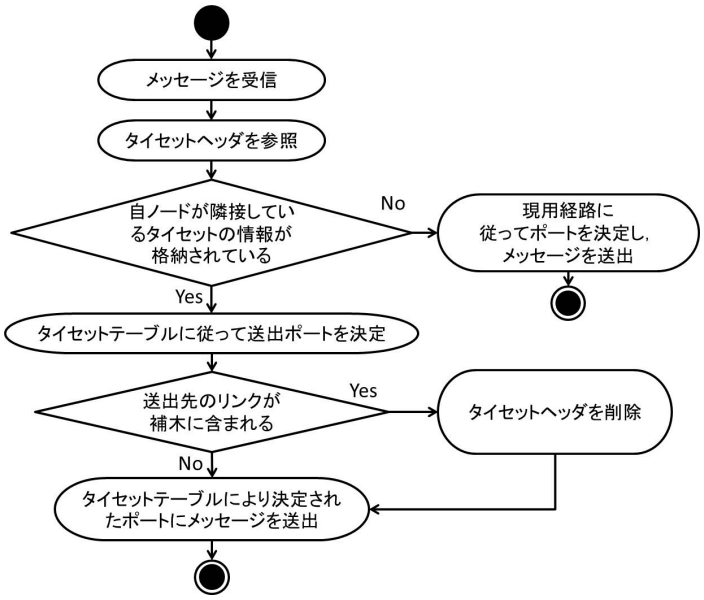


図 4.4: ノードのメッセージ転送処理

接続されている両端のノードをそれぞれ $v, w \in V$ とし，現用経路に木 T を使用する．

Step1 リンク e_f を経由するエントリの抽出

ノード v, w は障害を検出し，コントローラへ障害ポート番号を通知する．コントローラは，通知されたポートを用いて転送するエントリをノード v, w の現用経路のテーブルから抽出する．

Step2 復旧に使用するタイセット L_v^*, L_w^* の選択

コントローラはノード v, w それぞれに対し，障害が起きたリンク e_f を含むタイセットの中から最適なタイセット L_v^*, L_w^* を選択する．タイセットを選択する際は，適用するネットワークの性質に応じて，タイセットに含まれるリンク数やタイセット上の各リンクの余剰帯域などを選択基準とし，最適なタイセットを選択する．

Step3 タイセット L_v^*, L_w^* を用いたエントリの更新

コントローラは，Step1 で抽出したノード v のエントリの送出ポートをリンク e_r へ接続されているポートに書き換える．リンク e_r は，図 4.5 に示すように，タイセット L_v^* に含まれ，かつ，ノード v に接続されている 2 つのリンクの内，障害リンク e_f でないリンクである．ノード w に関しても同様の操作を行う．

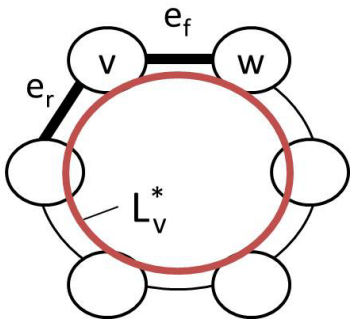


図 4.5: リンク e_r の例

Step4 タイセットヘッダの付加

ノード v はタイセット L_v^* の ID とノード w からノード v への向きが書きこまれているタイセットヘッダを作成し，転送するメッセージへ付加する．ただし，リンク e_r が補木に含まれる場合はタイセットヘッダを付加しない．ノード w に関しても同様の操作を行う．

この手順において最も計算量が多くなるのは step2 である．最適なタイセットを求める基準値をあらかじめ求めておけば，stap2 の計算量はたかだか $O(|\overline{T}|)$ となる．

4.5 OpenFlow による障害復旧方式の実装

4.5.1 フローエントリの作成

タイセットに基づく障害復旧方式は事前に現用経路と予備経路を作成し，障害リンクの両端スイッチの転送ルールを変更する．ここでは，本方式を実現するためのフローエントリの実装について述べる．

コントローラはスイッチのトポロジを把握すると，図 4.6 のように木と基本タイセット系を作成し，保持する．スイッチよりフローエントリが設定されていないパケットが送られてきたとき，コントローラは受信したパケットのヘッダ情報と保持している木を用いて現用経路を作成する．フローエントリのルール（Match 条件）には，多くのパケットヘッダ領域，L3 においては IP アドレス、プロトコル番号，ToS フィールド，また，L4 においてはポート番号などの OpenFlow によって使用可能なパケットのヘッダ情報を，任意の組み合わせで使用可能である．

ただし，後述する予備経路に使用する VLAN_ID フィールドは Match 条件に使用できない．

図 4.6 において，host1 から host2 へパケット転送したとき，各スイッチへ登録されるフローエントリ を表 4.3 に示す．なお，Match 条件には送信先 MAC アドレスのみを使用している．表 4.3 中の dl_dst は送信先 MAC アドレスを，p はスイッチのポート番号を表す．priority は，フローエントリが適応される優先度を示し，これが高いと他のフローエントリよりも先に到着したパケットと適合するか確認される．これらのフローエントリは，木のリンクを経由し，全てのスイッチから host2 へ向けてパケットを転送している．例えば，host1 から host2 へ向けてパケットを送信した場合，使用される経路は $(host1, v_1, v_0, host2)$ となる．また，コントローラは現用経路で作成したパケットの Match 条件と，そのパケットがどのスイッチのどのポートから送られてきたかを示すデータベース（fdb）を持つ．

次に，タイセットに沿って回転する予備経路のフローエントリの作成法について述べる．タイセットヘッダのタイセット ID と回転の向きは，VLAN_ID フィールドへ書き込むことで実現する．1 つのタイセットには回転の向きが 2 つあるため，タイセット 1 つにつき 2 つのタイセット ID が必要になる。つまり，VLAN-ID が 12bit までだとすると，2048 個までのタイセットに適用可能であることになる。もし，タイセット ID が枯渇する可能性がある場合は，送信元 MAC アドレスのような，適用するネットワークにおいて他の通信に影響を与えにくいフィールドを選んで，複数のフィールド領域の組み合わせからタイセット ID を作成することも可能である．なお，タイセット ID は，複数のタイセットに同じ番号を使用しなければ，任意の番号付けを行うことが可能であり，タイセット ID は，タイセットを作成した順番に番号付けする方法をとっている．また，予備経路のフローエントリの優先度は現用経路よりも高く設定する．図 4.6 のタイセット L_3 上を時計回りに回転するフローエントリを表 4.4 に示す．タイセット ID と回転の向きは，補木のリンクの両端スイッチを用いて，v_4v_0 と表記した．予備経路のフローエントリは，タイセットヘッダが書き込まれたパケットをタイセットテーブルに従ったポートへ転送している．ただし，転送に使用するリンクが補木に含まれる場合，転送する前にタイセットヘッダを削除する．

リンク障害発生時には，復旧に使用するタイセット ID をパケットの VLAN_ID フィールドへ書き込む．図 4.6 中のリンク (v_1, v_0) に障害が発生した場合の v_1 のフローエントリを表 4.5 に示

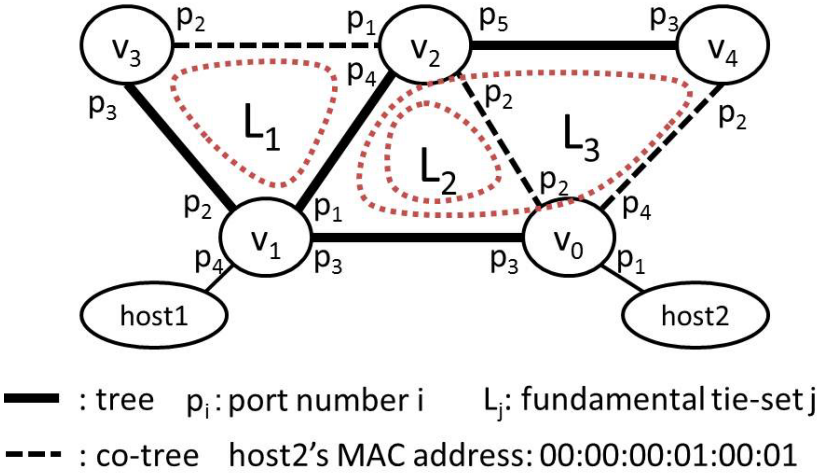


図 4.6: ノード数 5 のネットワークの例

す．予備経路への切り替えは，フローエントリを v_1 と v_0 へ登録することで可能である．例えば，v_1 へのフローエントリは，host2 宛のパケットへタイセット L_3 の ID と回転の向き（v_4v_0）を書き込み，ポート p_1 へパケットを送出する．ID がつけられたパケットは，予備経路 (v_2, v_4, v_0) を通り，スイッチ v_4 において ID が取り除かれる．

4.5.2 モジュール構成とモジュール間メッセージ

OpenFlow コントローラと OpenFlow スイッチによる仮想ネットワークを作成可能なフレームワークである Trema[24] を用いて，提案方式を tie-set switch モジュールとして実装した．Trema でのモジュール構成を図 4.7 に示す．switch manager と packet in filter は Trema のコアモジュールであり，topology と topology discovery は追加のアプリケーションとして公開されている．switches は Trema で生成可能な OpenFlow スイッチを表し，switch manager と OpenFlow メッセージをやりとりする．

switch manager は switches から送られてきた制御メッセージを解析し，各モジュールへメッセージを割り振る．フローエントリが存在しないパケットを含む制御メッセージは packet in filter へ渡し，トポロジの変更を表す state notify または port status メッセージは topology へ渡す．

packet in filter は受け取ったメッセージに含まれるパケットが LLDP である場合は topology

表 4.3: 現用経路のフローエントリ

Node	Rules	Actions	priority
v_0	dl_dst=00:00:00:01:00:01	output:p_1	65533
v_1	dl_dst=00:00:00:01:00:01	output:p_3	65533
v_2	dl_dst=00:00:00:01:00:01	output:p_4	65533
v_3	dl_dst=00:00:00:01:00:01	output:p_3	65533
v_4	dl_dst=00:00:00:01:00:01	output:p_3	65533

dl_dst: Destination MAC address

表 4.4: v_4 から v_0 の向きに回転するタイセット L_3 のフローエントリ

Node	Rules	Actions	priority
v_4	dl_vlan_ID=v_4v_0	strip_vlan,output:p_2	65534
v_0	dl_vlan_ID=v_4v_0	output:p_3	65534
v_1	dl_vlan_ID=v_4v_0	output:p_1	65534
v_2	dl_vlan_ID=v_4v_0	output:p_5	65534

dl_vlan_ID: VLAN_ID field of a ethernet header

表 4.5: リンク障害復旧用のフローエントリ

Node	Rules	Actions	priority
v_1	dl_dst=00:00:00:01:00:01	mod_vlan_vid (ID=v_4v_0), output:p_1	65534

mod_vlan_vid: modify the VLAN_ID field of ethernet header to specified number

discovery へパケットを渡し，他のパケットを tie-set switch へ渡す．

topology discovery は，LLDP を解析し，topology へスイッチの接続情報を渡す．topology は新たな接続が発見された場合，link add メッセージを tie-set switch へ送る．また，state notify や port status メッセージがリンクの障害発生を示す場合，link down メッセージを tie-set switch へ送る．

tie-set switch は，link add メッセージと packet in filter から受け取るパケットを元に，現用経路や予備経路を設定するためのフローエントリを作成し，このテーブルを switch manager を介して switches へ送る．link down を受け取ったとき，tie-set switch は現用経路から予備経路への切り替えるためのフローエントリを送信する．

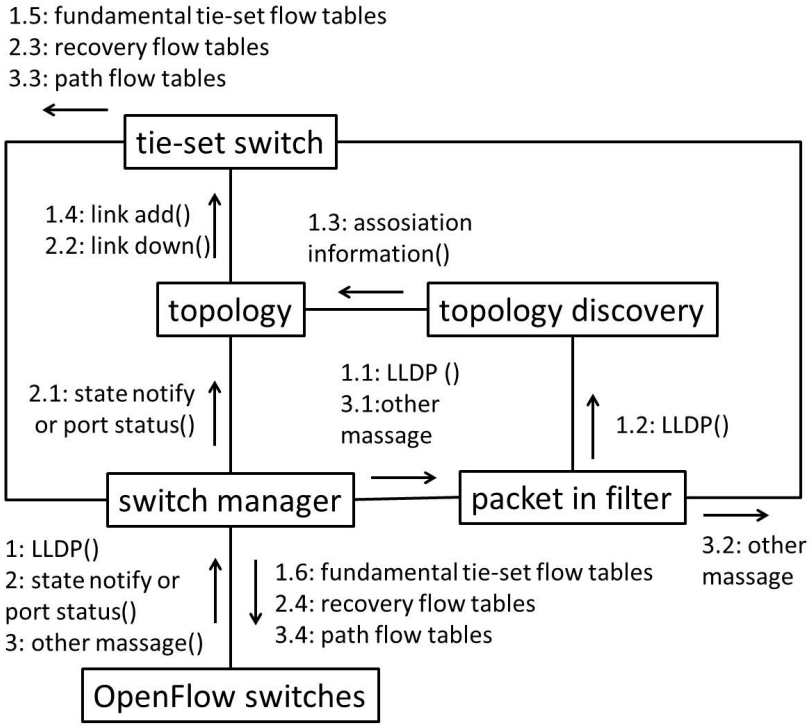


図 4.7: コントローラ内のモジュールと送受信されるメッセージ

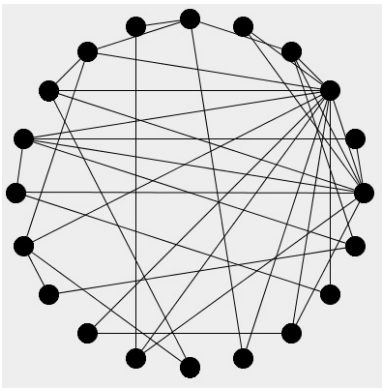
4.6 特性検証実験

4.6.1 実験環境

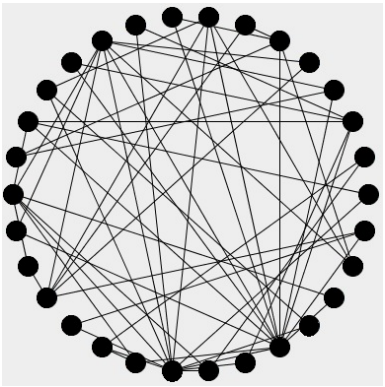
特性検証実験では，大規模なネットワークにおける本実装の特性を検証するため，ノード数を 10 から 290 まで 40 刻みで変化させ，それぞれのノード数において，5 種類のランダムグラフを用いた．このランダムグラフは，JUNG(Java Universal Network/Graph framework) の BarabasiAlbertGenerator クラスを用いて作成しており，グラフが 2 連結になるように工夫している．図 4.8 にノード数 20(a) とノード数 30(b) のランダムグラフの例を示す．作成したグラフを元に，trema の仮想ネットワーク構築機能を用いて，マシン上に仮想ネットワークを構築する．なお，trema は，OpenFlow スイッチとして，オープンソースとして公開されている Open vSwitch[36] を用いている．

実験は以下の手順で行った．

1. 二つの異なる送信元，送信先ノードを決める



(a) 20 nodes network.



(b) 30 nodes network.

図 4.8: ランダムグラフの例

2. 現用経路と予備経路の設定終了を待ち，パケットの転送を開始する

3. 一定時間待ち，パケット転送中にリンク障害を起こす

4. パケットの転送終了を待ち，制御メッセージ数と送信先ノードで受信したパケット数を計測する

転送するパケットはUDP，総送信パケット数は20000，1秒あたりの送信パケット数は1000pps，パケットサイズは50バイトとした．ノードのリンク速度を10Mbpsとしたが，転送する情報量に対して十分な帯域である．

4.6.2 実験結果

ここでは，実装した障害復旧方式の大規模ネットワークにおける特性を検証するための実験を行った．なお，図4.9から図4.11に示されている平均値の信頼区間は信頼係数95 %に基づき計算している．

まず，ノード数による障害復旧時のパケットロス数の変化率を図4.9に示す．この変化率は，ノード10におけるパケットロス数の平均値 AVR_{10} を基準とし，計測値 m としたとき，

$$\frac{m}{AVR_{10}}$$

で求められる値の平均値である．図 4.9 より，ノード数が増えたとしてもパケットロス数は変化していないことが分かる．なお，今回の実験における AVR_{10} は 43.54 パケットであった．

まず，制御メッセージ数を図 4.10 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．この図の左縦軸は 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，障害復旧に使用したタイセットのリンク数を図 4.11 に示す．この値は，復旧前の経路と比べて，障害復旧後の経路が最大でどれだけ長くなるかを示している．タイセットはリングを構成するリンクの集合であるため，これに含まれるリンクの最小値は 3 となる．予備経路のリンク数は，タイセットのリンク数を $|L|$ としたとき，最大で $|L|-1$ になるため，タイセットのリンクを最小値である 3 に近い値にすれば，予備経路の長さを抑制することができる．また，タイセットのリンク数の最大値はノード数であり，このようなタイセットを用いて障害復旧した場合，復旧後の経路は全てのノードを経由してしまう．図 4.11 中の average は，障害復旧に使用したタイセットのリンク数の平均であり，max はその最悪値である．図 4.11 より，ノード数が増加しても，使用するタイセットのリンク数の増加は緩やかであることが分かる．つまり，予備経路長が抑えられている．

最後に，予備経路のための平均フローエントリ数を図 4.12 に示す．図 4.12 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．また，図 4.13 より，切り替えのためのフローエントリ数は一定であった．

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．

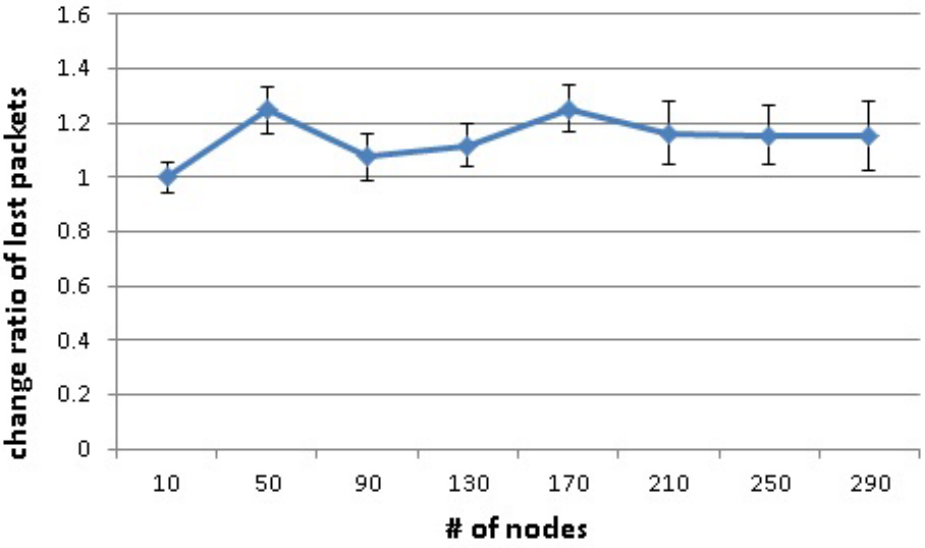


図 4.9: ノード数によるパケットロス数の変化率

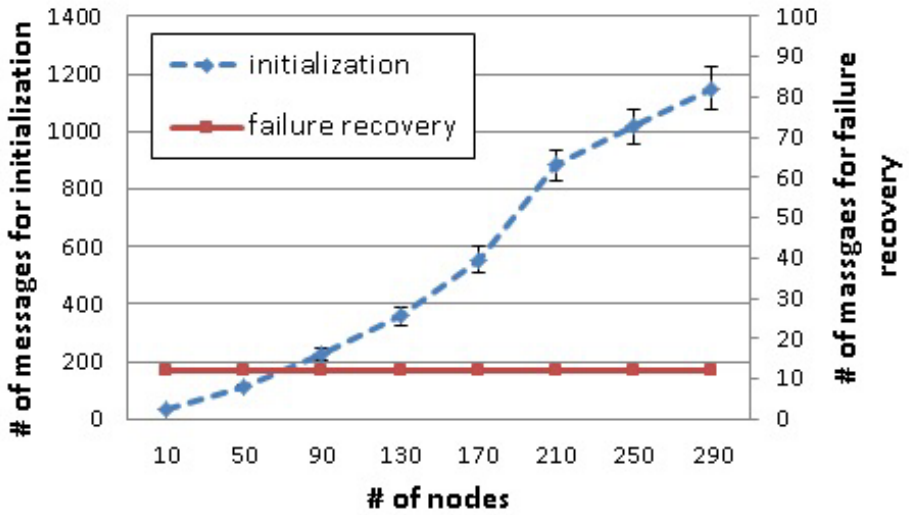


図 4.10: ノード数とセキュアチャネルのメッセージ数

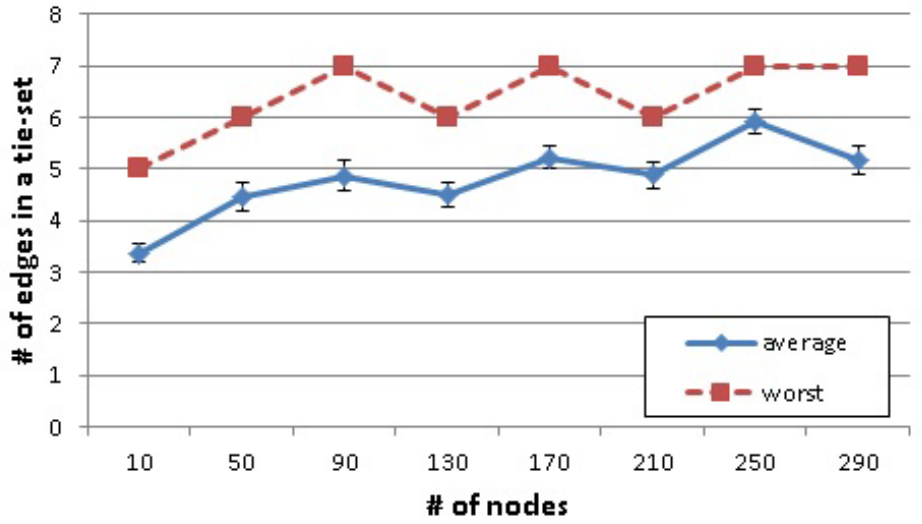


図 4.11: 障害復旧に使用したタイセットのリンク数

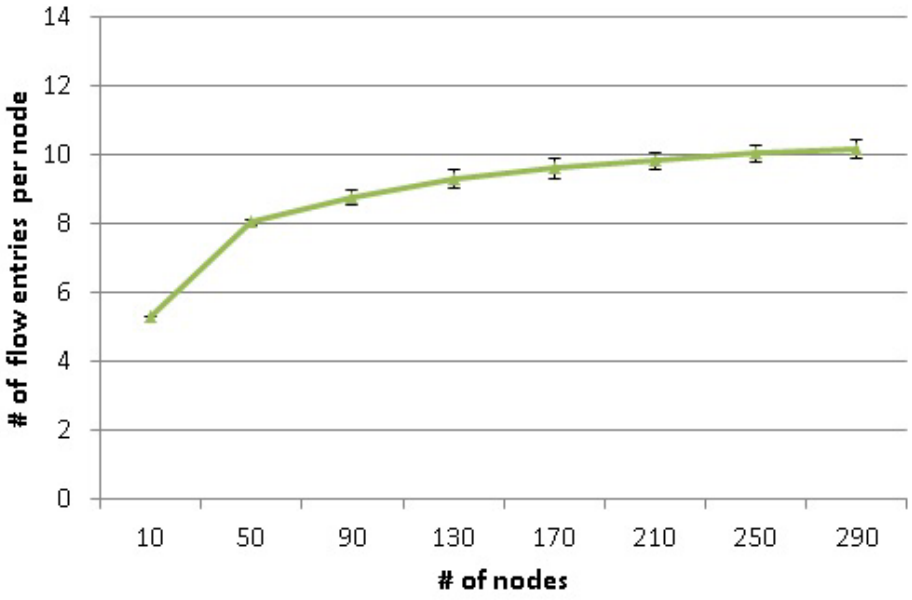


図 4.12: ノード数と平均フローエントリ数

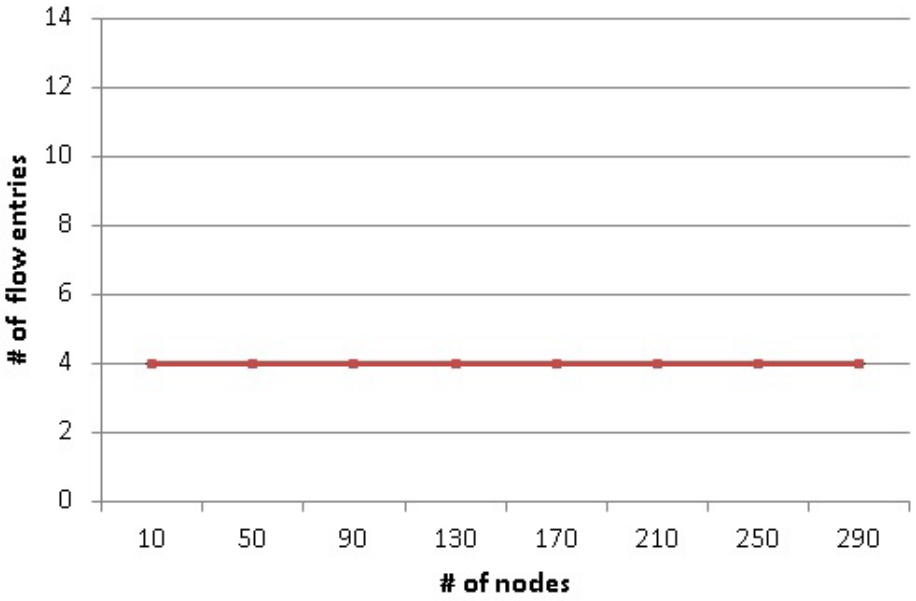


図 4.13: ノード数と経路切り替えのためのフローエントリ数

4.7 まとめ

本章では，現実的なネットワークに近い環境でサイクル構造に着目した障害復旧を実現するために，方式を OpenFlow へ適応した．適応するためには，制御ノードであるコントローラの計算量と障害復旧時の制御メッセージ数，フローエントリ数を抑える必要があった．そこで，予備経路を求めるための計算量（基本タイセット系を求めるための計算量）と障害発生時の予備経路への切り替えのための計算量が多項式時間に抑えられていることを示した．また，検証実験では，提案方式がネットワーク規模に関係なくパケットロス数，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予備経路の長さとフローエントリ数も，抑えられていることが分かった．以上より，提案した実装法を用いれば，大規模ネットワークにおいても基本タイセット系を用いた障害復旧方式が動作可能であることが期待される．

第5章　信頼性を考慮した複数のコントローラ間の情報共有のための負荷軽減手法の提案

前章では，単一コントローラによるネットワークの制御手法について述べたが，スイッチが増えることによる負荷の増大や，コントローラとスイッチ間の遅延の影響により，制御が困難な場合がある．そこで，複数のコントローラ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 [30]．複数のコントローラを用いる場合，コントローラ間で制御情報を共有する必要があり，SDN の利点であるプログラマビリティを活かしつつ，スケーラビリティを確保するため，DHT などの分散データベースを用いた制御方法が提案されている [12]．

これらの研究では，ネットワークが大規模になった場合，トポロジ情報や発生したイベント情報等の共有情報が増大してしまう可能性がある [27]．そこで，Onix[12] 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，複数のスイッチを集約し，一つのスイッチとして分散データベースに登録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，隠蔽するトポロジの信頼性を考慮する必要がある．例えば，図 5.1 の左側に示すようなネットワークをコントローラが管理しており，その中にあるリンクに障害が発生した場合，集約の方法によりその障害の影響が大きく変わってしまう．図 5.1 の右上のように全てのノードを集約し一つのノード (集約ノード) とした場合，単一リンクの障害が集約ノードの障害として扱われてしまう．図 5.1 の右下のような 2 連結成分のみを集約する場合，リンク障害は共有されているリンクの障害として扱うことが可能である．また，2 連結成分内の単一障害であれば，他のコントローラへ障害を隠蔽することが出来る．集約ノードを作成する場合，単一コントローラが管理する部分ネットワークに内在する 2 連結成分が大きければ，共有

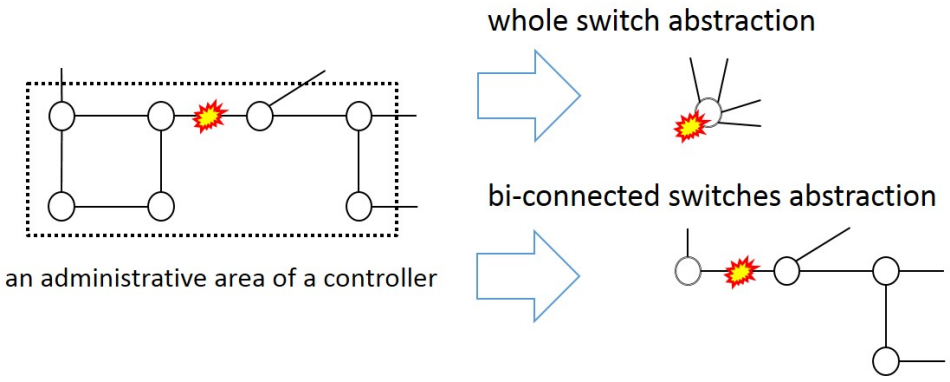


図 5.1: 集約方法と集約後ノードの信頼性

情報を効率的に削減することが出来る．

しかし，これまでの研究では，各コントローラが把握するネットワークの信頼性については議論されていない．ある研究では，コントローラの数とネットワーク上のどこに配置すればよいのかを決定する問題について議論がなされている [30]．この研究において，コントローラが管理するスイッチは，コントローラとスイッチ間の遅延のみを見て決定されている．また，頻繁に発生するネットワークイベント（フローセットアップリクエストやネットワークの状態を把握するためのイベント）の負荷が増大するため，コントローラを上位と下位に階層化し，下位コントローラが細かいイベントの情報をまとめて上位へと伝えることで上位コントローラへの負荷を削減する手法（Kandoo）が提案されいてる [37]．この研究では，下位コントローラの管理するスイッチ数のバランスにのみ注目しており，信頼性の議論はなされていない．そこで，本研究では，コントローラ間で共有される集約ノードの信頼性を考慮し，共有情報を削減するためのスイッチとコントローラの対応付け問題を定式化し，共有情報を削減するサイクルクラスタリングアルゴリズムを提案する．

5.1 問題の定式化

本節では，コントローラの管理範囲と集約ノード信頼性，共有情報の情報量を定義し，問題を定式化する．まず，コントローラが管理するスイッチの決定は，ネットワークをモデル化したグラフ上においてクラスタリングを求めることと同義である．クラスタリングを以下のように定義する．

定義 5.1. クラスタリング $\mathscr{C}$

グラフ $G = (V, E)$ が与えられた時，G のクラスタリング $\mathscr{C}$ は空でないノード集合 (クラスタ)$V_i \in V$ の集合とする．ここで，l はクラスタの総数を表す．ただし，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．

次に，ノードの集約を定義する．

定義 5.2. 2連結集約ノードと全体集約ノード

グラフ G とクラスタリング $\mathscr{C}$ が与えられたとき，クラスタ $V_i \in \mathscr{C}$ 内の 2連結成分を用いて G を縮約して作成するノードを 2連結集約ノードと呼び，V_i に含まれる全ノードを縮約したものを全体集約ノードと呼ぶ．特に区別しない場合は，単に集約ノードと呼ぶ．

次に，集約ノードの信頼性を測るため AP という指標を定義する．集約前のネットワークにおいて発生した障害が，集約ノードの異常を引き起こしてしまう場合，正常に動作しているノードやリンクも使用できなくなってしまう．そこで，集約ノードの重要度を集約ノードの次数とし，あるリンク e に発生した障害の影響度 AP を以下のように定義する．

定義 5.3. 単一リンク障害の影響度 AP

障害が発生したリンクを含むクラスタを V_e と表し，V_e を縮約した全体集約ノードを v_e とする．ただし，e がクラスタ間を結ぶリンクであった場合，V_e は空集合とし，v_e は存在しない．

$$AP(e) = \begin{cases} \delta(v_e) & (\kappa(G[V_e] - e) = 0) \\ 0 & (otherwise) \end{cases} \tag{5.1}$$

なお，2連結集約ノードの AP は常にゼロである．

次に，この AP を用いて，クラスタリングの信頼性を評価する指標 TotalAP を定義する．

定義 5.4. クラスタリングの信頼性の指標 $TotalAP$

グラフ G に含まれる全てのリンクに対する AP の総計を $TotalAP$ とする．

$$TotalAP = \sum_{e \in E} AP(e). \tag{5.2}$$

$TotalAP$が低くなれば，リンク障害の影響を少なくすることが出来るため，クラスタリングの信頼性が高くなる．

次に，コントローラ間において共有される情報について述べる．クラスタリングにより決定された管理範囲全体を集約する場合は，隠蔽できる情報量が多いが，集約ノードの信頼性が低くなってしまう．集約ノードの信頼性を確保したい場合は，2 連結の集約により信頼性を高めることは出来るが，隠蔽できる情報量が少なくなってしまう．情報を隠蔽できない場合，コントローラ間で共有される情報量が多くなり，トポロジの変化やノードやリンクの統計情報の更新によって発生する負荷が大きくなってしまう．ここでは，2 連結集約を行った場合にコントローラが共有する情報量について定義する．

コントローラが持つべき情報には，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がある．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と協調動作によるネットワーク全体の制御を行う必要がある．前者の機能をローカルコントローラと呼び，後者の機能をフェデレータと呼ぶ．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 5.2 にローカルコントローラ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している．フェデレーショングラフを以下のように定義する．

定義 5.5. フェデレーショングラフ G^f

グラフ G とクラスタリング $\mathscr{C}$ が与えられたとき，クラスタ内の 2連結成分を全て縮約したグラフをフェデレーショングラフ G^f と呼ぶ．このグラフを構成するノード集合を V^f とし，リンクの集合を E^f と表す．フェデレーショングラフの作成する操作を $f(G,\mathscr{C})$ と表す．

コントローラ間の共有情報の量はフェデレーショングラフのサイズと同じであり，クラスタリングによって変化する．これまで，Conductance などのクラスタリングを作成するための尺度が検討されてきた [38]．これら評価尺度はコントローラ内のリンク数やコントローラ間のリンク数を元に計算されている．しかし，これらの尺度はグラフのトポロジではなく，リンク数

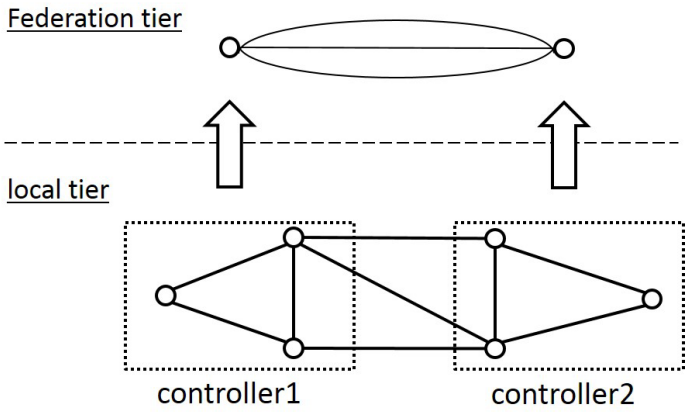


図 5.2: フェデレーション層とローカル層の例

にのみ注目しているため，フェデレーショングラフのサイズを削減するためには使用できない．例えば，図 5.3(a)，(b)，(c) は，同じグラフを 3 種類のクラスタリングによって分割した場合のフェデレーショングラフを示しており，(a) と (b) のクラスタリングではクラスタ間を結ぶリンクとクラスタに含まれるノード数が同じであるが，導出されるフェデレーショングラフのサイズは (b) の方が大きくなっている．(a) と (b) のクラスタリングの違いはクラスタ内のリンク数のみであり，(b) は (a) と比べて，クラスタ内に含まれるリンク数の偏りが大きい．そこで，クラスタ内のリンク数を等しくするクラスタリングが良いと考えられるが，図 5.3(c) に示すように，クラスタ内リンク数が等しいクラスタリングを用いても，フェデレーショングラフのサイズを小さくできるとは限らない．

以上より，信頼性が高く，共有情報の少ないクラスタリングを作成するためには，クラスタ内の 2 連結成分を大きくする必要がある．そこで，2 連結集約をしたフェデレーショングラフ $G^f = (V^f, E^f)$ のサイズを指標として用いる．フェデレータグラフ G^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．

$$\text{Minimize } |E^f| = E(f(G, \mathscr{C}))\ \ s.t. |V_i| \leq k, V_i \in \mathscr{C} \tag{5.3}$$

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る．

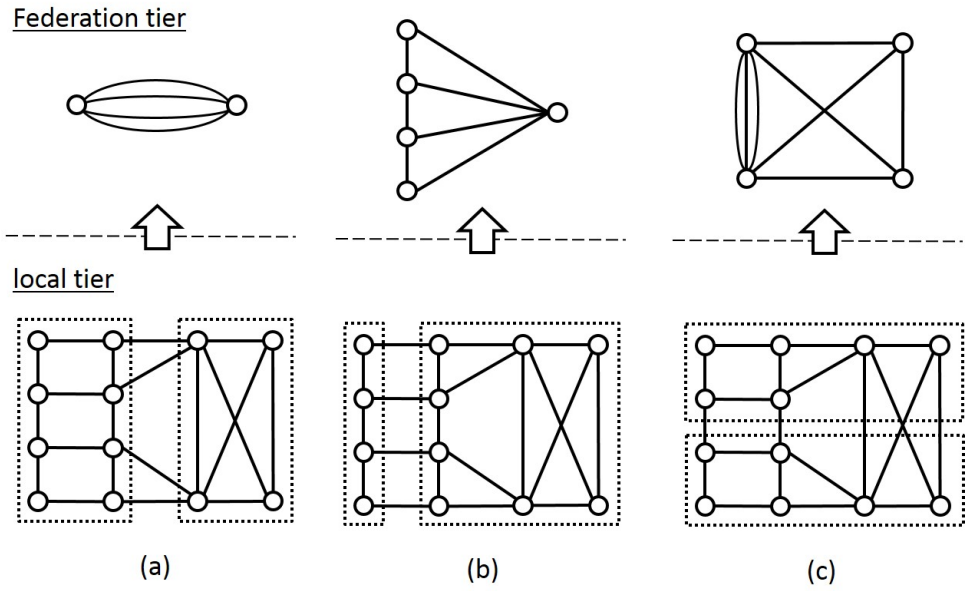


図 5.3: クラスタリングによるフェデレーショングラフの違い

5.2 サイクルクラスタリング

フェデレーショングラフのサイズを少なくするためのサイクルクラスタリングアルゴリズムを提案する．サイクルクラスタリングは，グラフ内の単純な 2 連結成分であるサイクル単位でクラスタリングを行う．そのため，候補となるサイクル集合よりサイクル同士の隣接関係を表すサイクルグラフを作成し，このグラフを用いてサイクルを選択し，選択したサイクルよりクラスタを作成する．

まず，サイクルグラフを定義する．

定義 5.6. サイクルグラフ $\underline{G}$

グラフ $G = (V, E)$ においてサイクル集合 $\mathcal{L}$ が与えられたとき，サイクルグラフ $\underline{G} = (\mathcal{L}, \underline{E}, \omega)$ は，$\mathcal{L}$ に含まれるサイクルをノードとし，サイクル同士のノードの共有関係をリンクとするグラフである．リンクコスト $\omega : \mathcal{L} \times \mathcal{L} \to \mathcal{P}(V)$ は，2 つの異なるサイクル $L_i, L_j \in \mathcal{L}$ が共有しているノード集合 $V(L_i) \cap V(L_j)$ を表す．ここで，$\mathcal{P}(V)$ は V の冪集合である．

サイクル集合とサイクルグラフの例を図 5.4 に示す．図 5.4(b) のリンクの傍に書かれいてる文字はリンクコストを表す．

次に，クラスタとするサイクルの選択方法について説明する．ノードが複数のクラスタに含

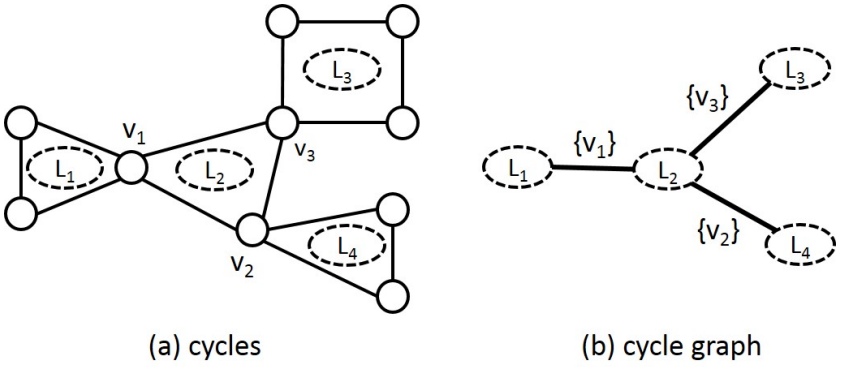


図 5.4: サイクルグラフの例

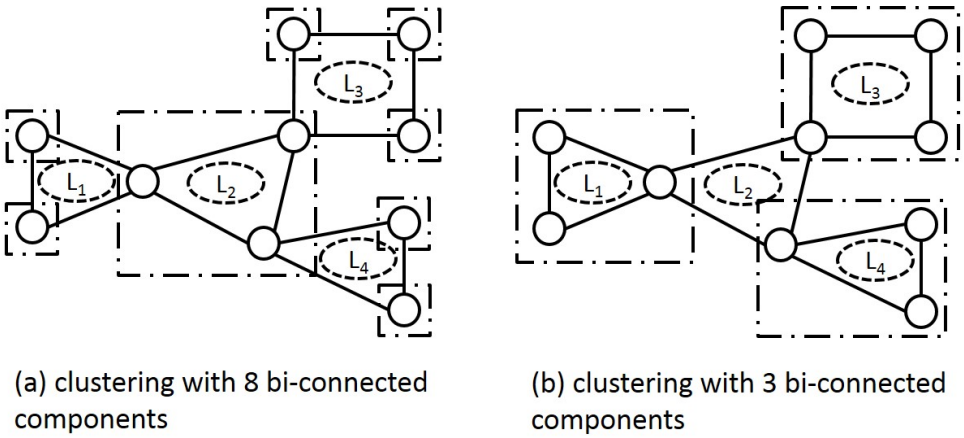


図 5.5: サイクルの選択順による 2 連結成分の違い

まれないという条件より，サイクルの選択の際に工夫が必要となる．もし，あるサイクルを選択しクラスタとした場合，そのサイクルとノードを共有しているサイクルは分断されてしまい，2 連結なサブグラフではなくなってしまう．例えば，図 5.4(a) のグラフにおいて L_2 をクラスタとした場合，サイクル L_1，L_3，L_4 は分割されてしまう．さらに，最初に L_2 を選択した場合，図 5.5(a) のようにクラスタ内の 2 連結成分が増加し，フェデレーショングラフのリンク数が多くなってしまう．以上より，図 5.5(b) に示すようなクラスタリングを行うためには，サイクルを選択したときに分割されるサイクルに含まれ，かつ，分割されないサイクルに含まれないノード数が少ないことが重要となる．これらのノードを孤立ノードと呼び，以下のように定義する．

定義 5.7. 孤立ノード $IN(L_a)$

サイクルグラフ $\underline{G} = (\mathcal{L}, \underline{E}, \omega)$ において，サイクル L_a が選択されたとき，孤立ノード $IN(L_a)$

は

$$IN(L_a) = \cup_{L_i \in N_{L_a}} \left(V(L_i) - \cup_{L_j \in N_{L_i} - N_{L_a}} \omega(L_i, L_j) \right) \tag{5.4}$$

とする．ここで，N_{L_a} はサイクルグラフ $\underline{G}$ 上でサイクル L_a と隣接関係にあるサイクルの集合である．

図 5.5 において，L_1，L_3，L_4 の孤立ノードは無く，L_2 は 7 ノードとなる．

孤立ノードを指標として使用して選択したサイクルに含まれるノード数がクラスタのノード制限 k よりも少ない場合，分断したサイクルを用いてクラスタを拡張することが出来る．L_i を用いたクラスタを分断されたサイクル L_j によって拡張した場合，孤立ノード $IN(L_i) \cap L_j, L_j \in N_{L_i}$ を 2 連結成分内に含めることが出来る．しかし，この拡張により新たな孤立ノード $INE(L_i, L_j)$ が生じてしまう．この拡張後孤立ノードは以下のように表すことが出来る．

$$INE(L_i, L_j) = IN(L_i) \cup IN(L_j) - V(L_i) - V(L_j). \tag{5.5}$$

この孤立ノードを用いたサイクルクラスタリングアルゴリズムを 1 に示す．このアルゴリズムは，手順 6 において，孤立ノードの数を最小とするサイクル L_a を選択し，手順 11 において，作成したクラスタを拡張するために拡張後孤立ノードの数を最小とするサイクル L_b を選択している．その後，L_a と拡張に用いた全てのサイクルに含まれるノード $V(C)$ を縮約する．縮約する際に，縮約後のノードに $|V(C)|$ という重みを付与する．手順 5 と手順 18 の繰り返しにおいて，全てのサイクルが削除されるまで縮約を繰り返す．その後，縮約されていないノードの重みを 1 とし，重み付けされた縮約グラフを用いてクラスタリングを求める．このために，algorithm 2 に示す Connected クラスタリングを用いる．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，重みの大きいノードをルートとし，ルートから BFS を用いてグラフを探索することで，ノード重みの合計が k 以下のクラスタを作成していく．Connected クラスタリングを用いることで，サイクルに含まれないノードのクラスタリングが可能となる．

図 5.6 に 16 ノードの格子状グラフにおけるサイクルクラスタリングの実行例を示す．この例では，クラスタのノード制限 k は 7 であり，9 つのサイクルが与えられている．まず，与えられたグラフでは $IN(L_1) = 0$ であるため，図 5.6(a) のように L_1 を選択する．次に，選択したサイ

Algorithm 1 Cycle clustering

Require: $G = (V, E)$, $\mathcal{L}$, and k

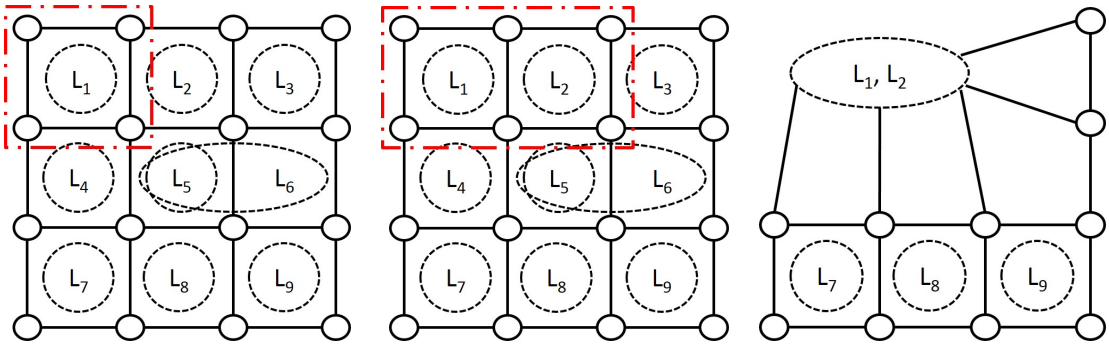
Let N_L be adjacent cycles of L in the graph $\underline{\mathrm{G}}$.
k is upper bound of a cluster size.
Create cycle graph $\underline{\mathrm{G}} = (\mathcal{L}, \underline{\mathrm{E}}, \omega)$ from G and $\mathcal{L}$.
Clustering $\mathscr{C} \leftarrow \phi$.
while $\mathcal{L} \neq \phi$ **do**
    Select a cycle L_a with minimum $IN(L_a)$ from $\mathcal{L}$.
    $C \leftarrow L_a$.
    $Candidates \leftarrow N_{L_a}$.
    Delete cycles with larger than $k - |V(C)|$ nodes from $Candidates$.
    while $Candidates \neq \phi$ **do**
        Select a cycle L_b with minimum $INE(C, L_b)$ from $Candidates$.
        Combine L_b to C.
        $Candidates \leftarrow N_C$.
        Delete cycles with larger than $k - |V(C)|$ nodes from $Candidates$.
    end while
    Contract $V(C)$ of G and set the weight of the contracted node $|V(C)|$.
    Delete C from $\mathcal{L}$.
end while
return Connected_clustering(the contracted graph with node weights, k)

クルに含まれるノード数は 4 であり，クラスタのノード制限よりも低いため，INE を指標として，拡張するサイクルを選択する．ここでは，図 5.6(b) に示すように，L_2 を選択している．L_1 と L_2 を選択した後，これ以上のクラスタ拡張が不可能であるため，これまでに選択した L_1 と L_2 のサイクルを縮約する．縮約後のグラフを図 5.6(c) に示す．同様に，IN と INE を指標とし，図 5.6(d) に示すように L_7 と L_8 を選択し，図 5.6(e) のようにノードを縮約した．図 5.6(e) のグラフは，これ以上縮約できるサイクルが無いため，次に，Connected クラスタリングをを用いて，クラスタリングを決定する．Connected クラスタリングでは，クラスタ内のノード数が $k = 7$ に近づくようにクラスタを作成し，作成されたクラスタリングが図 5.6(f) である．

Algorithm 2 Connected clustering

Require: $G = (V, E, \omega)$ and k

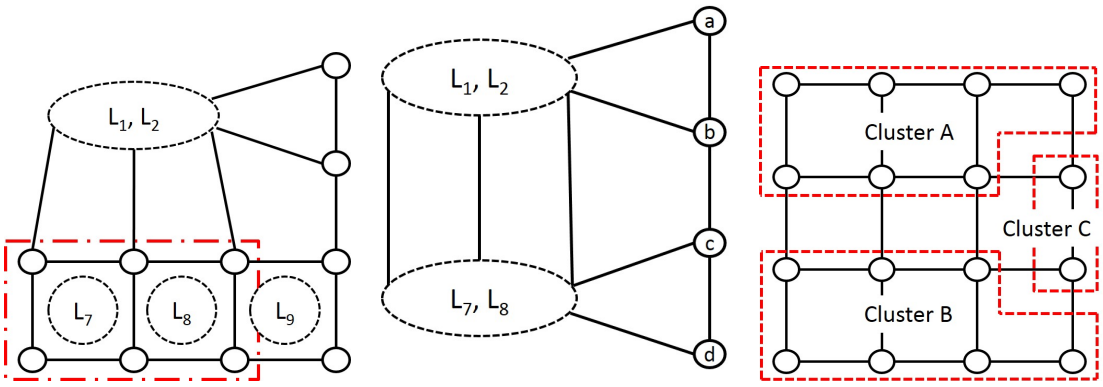
ω indicates the weight of node.
Clustering $\mathscr{C} \leftarrow \phi$.
while $V \neq \phi$ **do**
    Select node $r \in V$ whose weight is muximum and $\omega(r) \leq k$.
    $C \leftarrow r$.
    while a node in C has an adjacent node $v \in V$ whose $\omega(v) \leq k - |C|$ **do**
        $C \leftarrow v$.
    end while
    Clustering $\mathscr{C} \leftarrow C$.
    Delete C from V.
end while
return $\mathscr{C}$



(a) L_1 を選択 $(IN(L_1) = 0)$

(b) L_2 を用いてクラスタを拡張 $(INE(L_1, L_2) = 2)$

(c) L_1 と L_2 を縮約



(d) L_7 を選択後，L_8 でクラスタを拡張

(e) サイクル縮約グラフ

(f) サイクル縮約グラフを用いて作成したクラスタリング

図 5.6: サイクルクラスタリングにおけるサイクル選択とサイクル縮約 $(k = 7)$

5.3 特性検証実験

5.3.1 実験環境

サイクルクラスタリングの特性を評価するため，サイクルクラスタリング，HCS(Highly Connected Subgraphs) クラスタリング [39]，Connected クラスタリングによる縮約後ノードの信頼性 (TotalAP) とフェデレーショングラフのサイズを比較した．Connected クラスタリングにおいて，ノードの重みは全て 1 とした．また，HCS クラスタリングを Algorithm 3 に示し，このクラスタリング手法は，最小カットを用いて，グラフを繰り返し二分割していくことで，リンク密度の高いクラスタを作成する手法である．これらのクラスタリングは Python と NetworkX を用いて実装した．

実験には，Connected Caveman グラフと格子状グラフと Newman Watts Strogatz（NWS) ランダムグラフ，America グラフ，JPN48 グラフを用い，それぞれのグラフの例を図 5.7 に示す．Connected Caveman グラフ (図 5.7(a)) は，Cave と呼ばれるクリークから 1 つリンクを取り除いたグラフをサイクル状に連結したグラフである．このグラフは，データセンターなどのネットワークに見られる特徴を示しており，ラック内では密にサーバが接続されているが，ラック間をつなぐリンクは疎となっている．Caveman グラフはクラスタ係数が 1（最大）に近いグラフの中で最もリンク数が少ないことから，クラスタリングアルゴリズムの性能を計測するために用いられる．なお，グラフのクラスタ係数は各ノードのクラスタ係数の平均で表され，ノード v のクラスタ係数はノード v の隣接ノードが密に接続されていれば高い値になる．本実験で用いる Caveman グラフの Cave サイズは 10 とした．NWS ランダムグラフ (図 5.7(c)) は，信頼性の高いネットワークが構築されていることを想定し，グラフの連結度が 2 以上となるランダムグラフを作成するために使用した．グラフを作成するためには，ノード数，近傍ノードとの接続数，リンクの発生確率が必要となり [40]，今回の実験で用いたグラフでは，近傍ノードとの接続数は 1 とし，リンク発生確率はリンク数がノード数の倍になるように調整している．America グラフ (図 5.7(d)) と JPN48 グラフ (図 5.7(e)) は，実際の広域ネットワークをグラフモデル化したものを用いた．

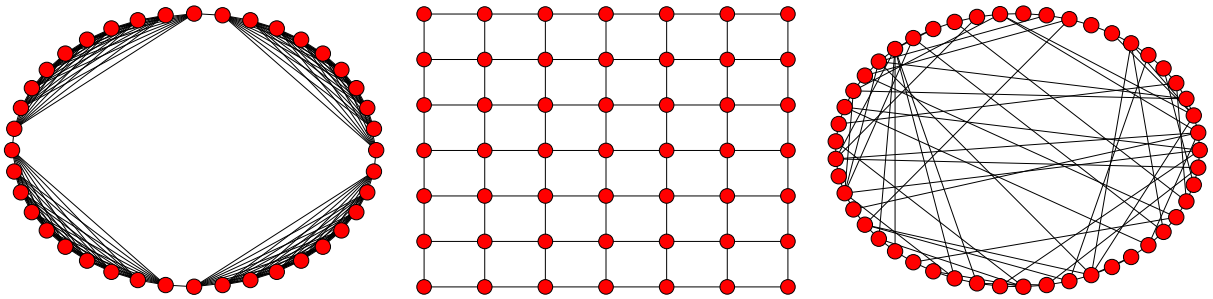
大規模なネットワークでの特性を検証するため，グラフのサイズが可変である Caveman グ

Algorithm 3 HCS clustering

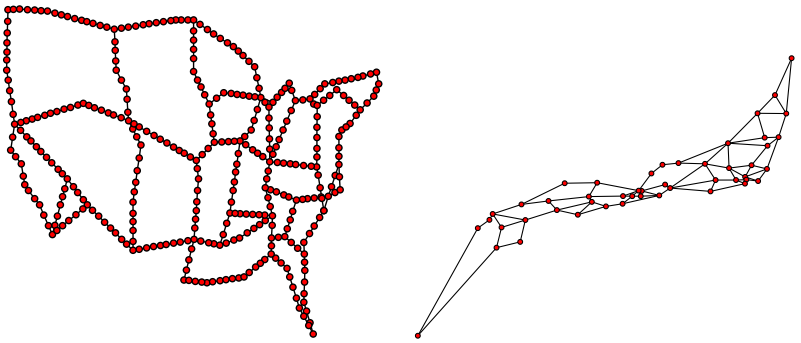
Require: $G = (V, E)$ and k

function HCS(G, k)
  $(H_1, H_2, C) \leftarrow MinimumCut(G)$.
  if G is highly connected and $|V(G)| \leq k$ **then**
    $\mathscr{C} \leftarrow V(G)$.
  else
    $HCS(H_1, k)$.
    $HCS(H_2, k)$.
  end if
end function
Clustering $\mathscr{C} \leftarrow \phi$.
HCS(G, k).
return $\mathscr{C}$.

ラフ，格子状グラフ，NWS ランダムグラフについて，表 5.1 に示すようにノード数を増加させる**スケーラビリティ実験**を行った．また，5 つのグラフ全てについて，クラスタのノード制限 k を変化させた場合の**管理範囲変更実験**も行った．なお，スケーラビリティ実験ではクラスタのノード制限 k を 15 に固定し，使用するグラフのノード数を増加させ，管理範囲変更実験ではノード数を格子状グラフと NWS グラフでは 144 に，また，Connected Caveman グラフでは 140 に固定している．



(a) Connected caveman　(b) 格子状　(c) NWS



(d) America [41].　(e) JPN48 [42].

図 5.7: 実験で使用したグラフ

表 5.1: 実験で使用したグラフのノード数とリンク数

	Connected caveman グラフ					
The # of nodes	40	60	80	100	120	140
The # of edges	180	270	360	450	540	630
The # of nodes in a cave	10	10	10	10	10	10
	格子状グラフ					
The # of nodes	49	64	81	100	121	144
The # of edges	84	112	144	180	220	264
	NWS グラフ					
The # of nodes	49	64	81	100	121	144
The # of edges	98	128	162	200	242	288
	America					
The # of nodes	365					
The # of edges	772					
	JPN48					
The # of nodes	48					
The # of edges	82					

5.3.2 実験結果

本節では，スケーラビリティ実験と管理範囲変更実験において計測した TotalAP とフェデレーショングラフのサイズを示し，サイクルクラスタリングの有効性について検証する．本節では，まず，各実験で計測した集約ノードの信頼性 (TotalAP) について述べ，次に，フェデレーショングラフのサイズを示す．

縮約ノードの信頼性

(i) スケーラビリティ実験の結果

図 5.8–図 5.10 にノード数を増加させた場合の TotalAP を示す．図中の cycle (bi-connected) と cycle (whole node) は，サイクルクラスタリングの結果を示し，connected (whole node) は connected クラスタリングの結果を示す．それぞれ括弧の中は，スイッチの集約方法を示しており，bi-connected は 2 連結成分を一つのノードとし，whole ノードはコントローラの管理範囲内のスイッチを一つのノードとした場合の結果である．TotalAP は，単一リンク障害が発生した際に集約ノードが影響を受ける場合に上昇するため，値が低いほうが信頼性が高い．そのため，2 連結成分を集約した場合，TotalAP はゼロとなる．HCS クラスタリングは，Caveman グラフ以外において作成されるクラスタが少数のノードで構成されるため TotalAP が計測できない．

図 5.8 より，Caveman グラフにおいて，サイクルクラスタリングは非常に信頼性の高いクラスタを作成できている．また，図 5.9 と図 5.10 に示すように，格子状グラフと NWS グラフにおいては，Connected クラスタリングよりも TotalAP を下げることが出来ている．

(ii) 管理範囲変更実験

次に，ネットワークの規模を変更せずにコントローラの管理できるスイッチの数を変更した実験の TotalAP を示す．この実験では，実際のネットワークをモデル化した America グラフや JPN48 グラフも用いており，図 5.11–図 5.15 に 5 つのグラフにおける結果を示す．

図 5.11 に示すように，Caveman グラフにおいて，k が変化した場合でも，サイクルクラスタリングが大きく TotalAP を削減できている．k の常勝に従いサイクルクラスタリングの TotalAP は増加傾向にあるが，上昇量は少なく抑えられている．また，ノード 20 と 30 の場合に connected クラスタリングの値が大きく減少しているが，これはクラスタのノード制限が Cave ノード数

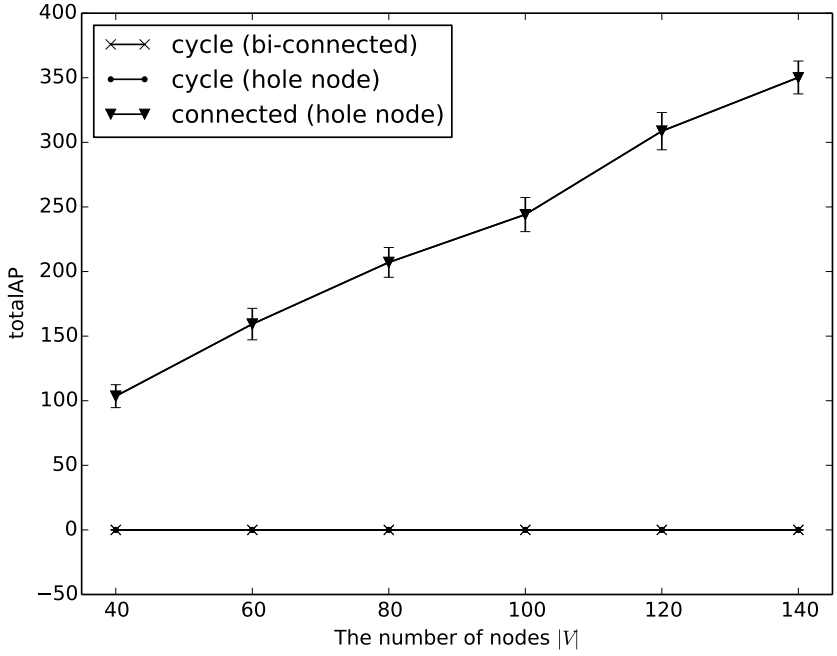


図 5.8: Caveman グラフのノード数を変化させたときの TotalAP

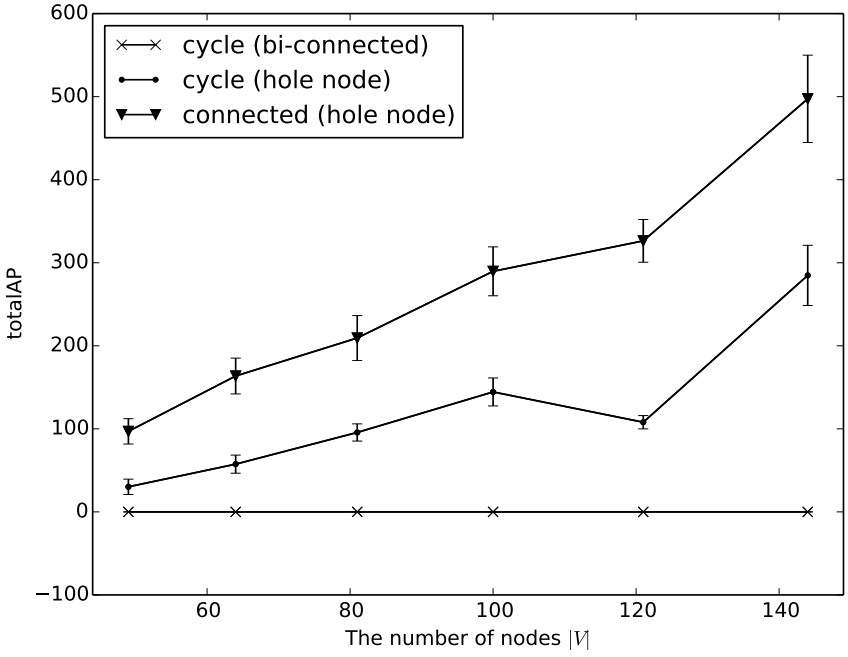


図 5.9: 格子状グラフのノード数を変化させたときの TotalAP

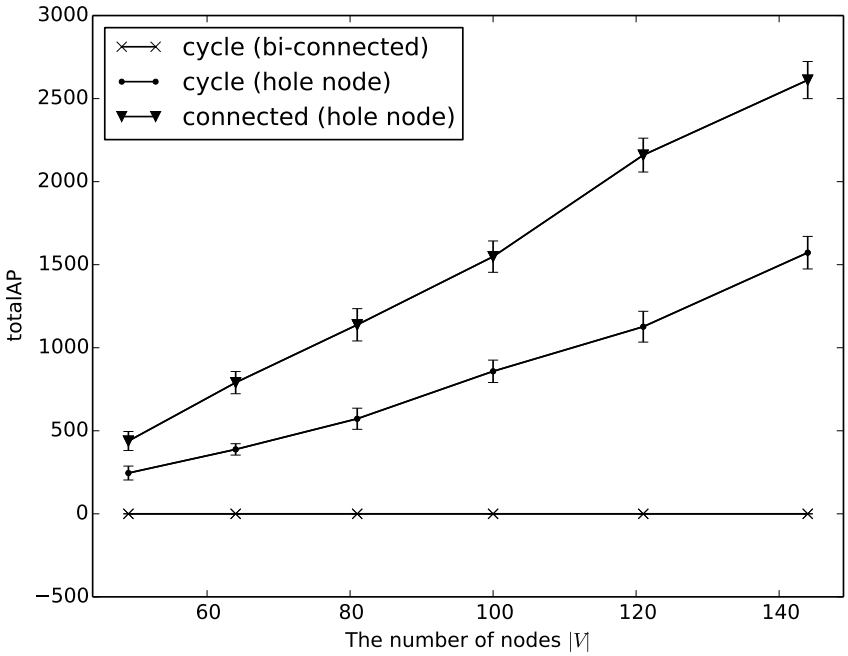


図 5.10: NWS グラフのノード数を変化させたときの TotalAP

（10 ノード）の整数倍になっているため，クラスタの探索が Cave を分断しない状態で終了しやすくなるためである．

格子状グラフにおける TotalAP を図 5.11 に示す．他のグラフにおける結果と比べて，サイクルクラスタリングの TotalAP 削減量が少なくなっている．これは，格子状グラフの構造上の性質により，Connected クラスタリングによっても効率的なクラスタを作成しやすいためである．

また，図 5.13–図 5.15 に示すように，NWS グラフ，America グラフ，JPN48 グラフにおいて，サイクルクラスタリングが大きく TotalAP を削減できている．実際のネットワークに近いモデルにおいても，サイクルクラスタリングが信頼性の高いクラスタを作成出来ていることが分かる．なお，America グラフと JPN48 グラフは，グラフのノード数が他のグラフのものと違うため，グラフのノード数に応じて k の値を調整している．

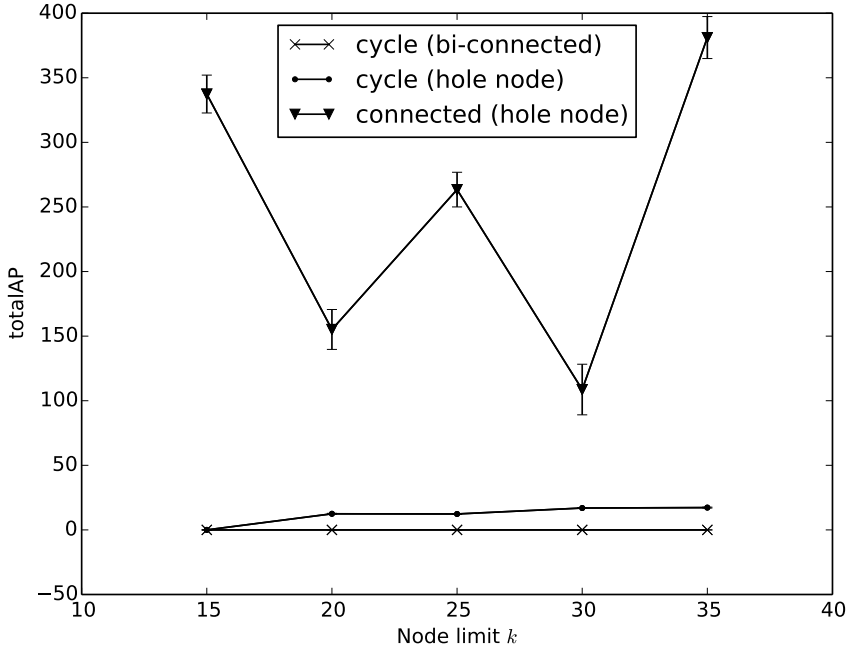


図 5.11: Caveman グラフにおいてクラスタのノード制限 k を変化させたときの TotalAP

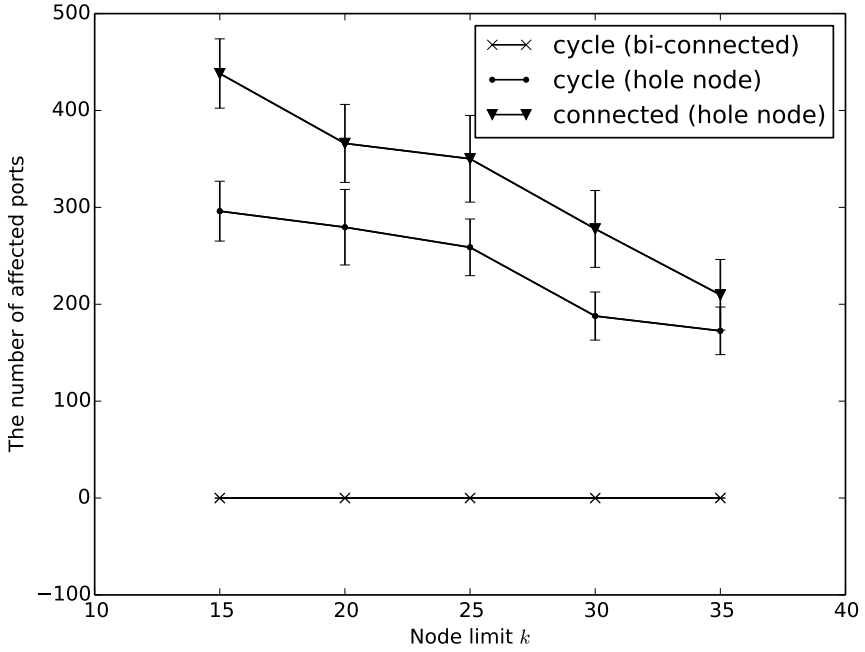


図 5.12: 格子状グラフにおいてクラスタのノード制限 k を変化させたときの TotalAP

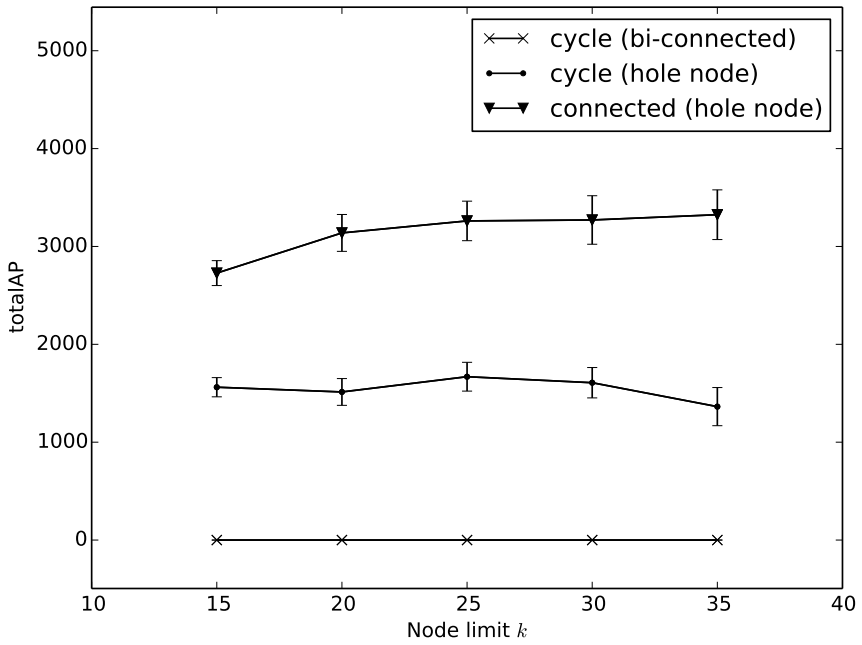


図 5.13: NWS においてクラスタのノード制限 k を変化させたときの TotalAP

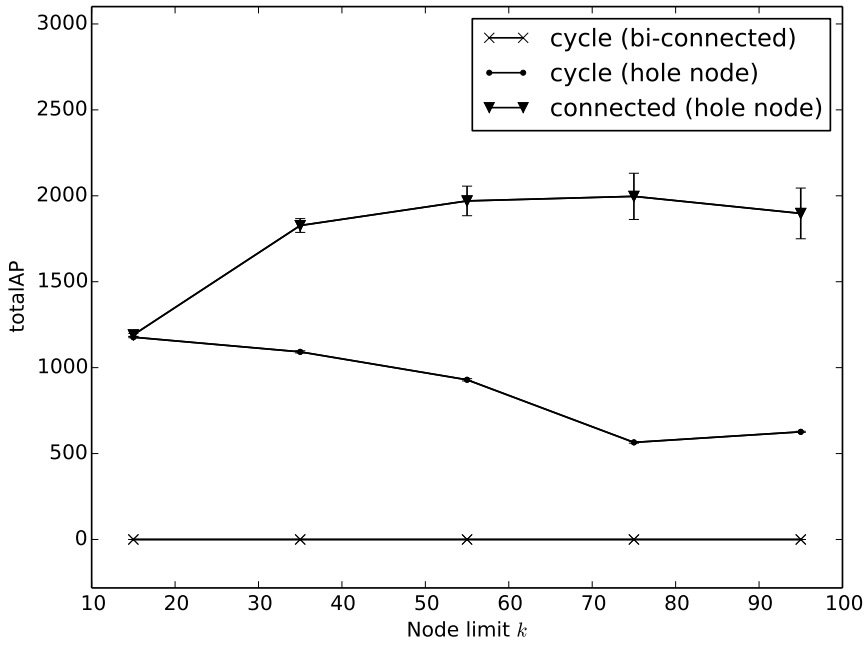


図 5.14: America グラフにおいてクラスタのノード制限 k を変化させたときの TotalAP

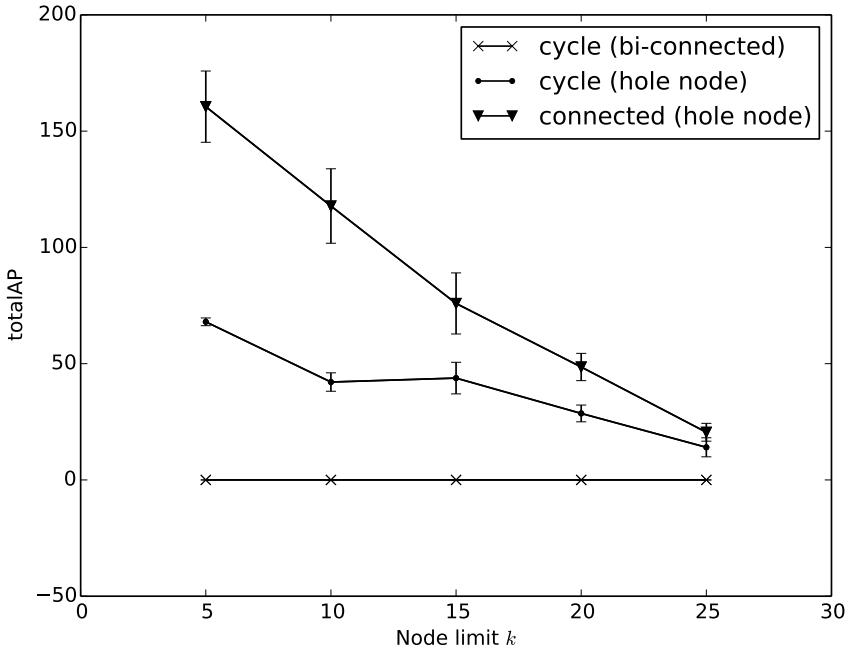


図 5.15: JPN48 グラフにおいてクラスタのノード制限 k を変化させたときの TotalAP

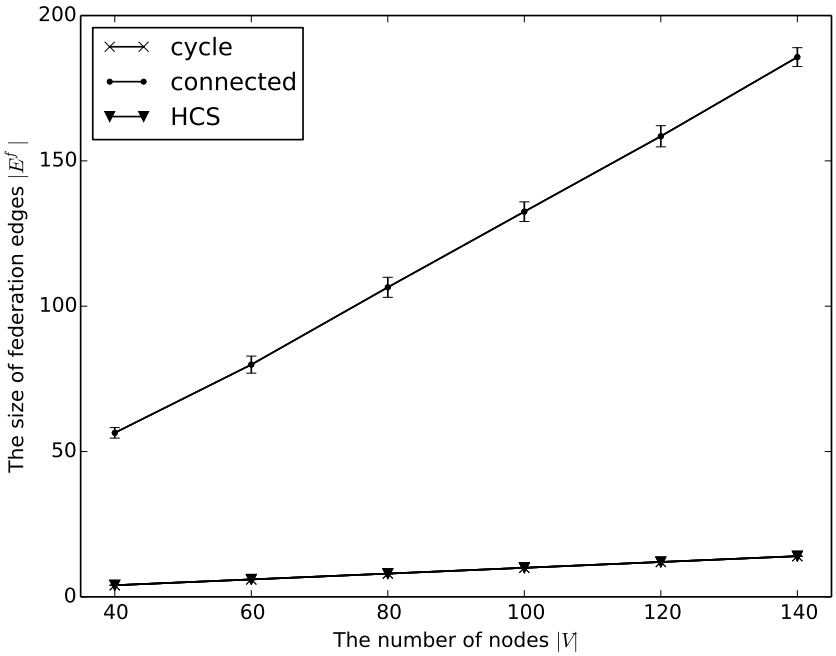


図 5.16: Caveman グラフのノード数を変化させたときのフェデレーショングラフのリンク数 $|E^f|$

フェデレーショングラフのサイズ

(i) スケーラビリティ実験の結果

図 5.16–図 5.18 にノード数を増加させたスケーラビリティ実験の結果を示す．図 5.16 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ショングラフのサイズを削減することができている．それに対し，図 5.17 より，格子状グラフにおいては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サイズを抑えており，HCS クラスタリングはグラフサイズが大きくなっている．図 5.18 に示す NWS グラフの結果では，各クラスタリングの差が小さくなっているが，サイクルクラスタリングが最もグラフサイズを抑えることができており，次に Connected クラスタリングとなり，最悪の値となったのが HCS クラスタリングであった．以上より，どのグラフにおいてもサイクルクラスタリングが他の手法よりもフェデレーショングラフのサイズを小さくすることができていた．

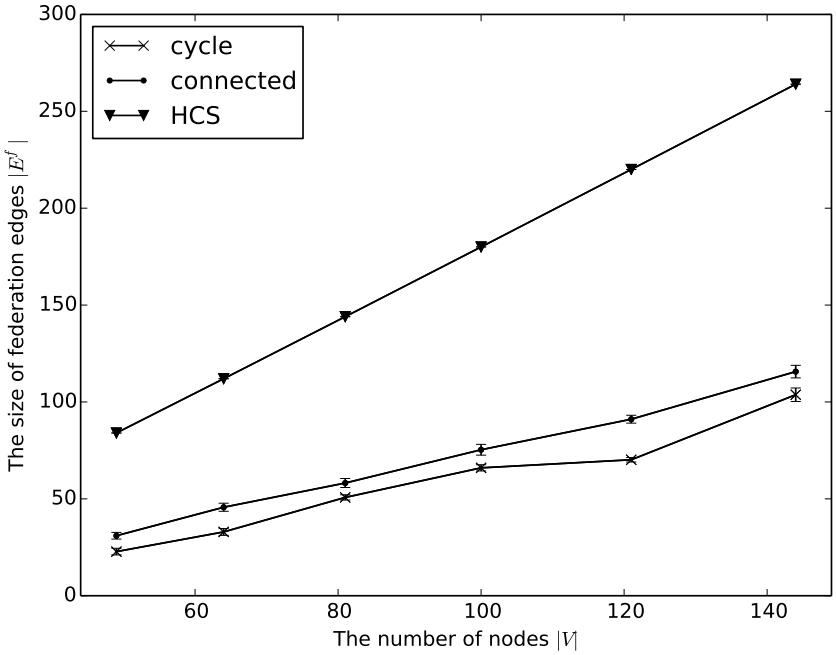


図 5.17: 格子状グラフのノード数を変化させたときのフェデレーショングラフのリンク数 $|E^f|$

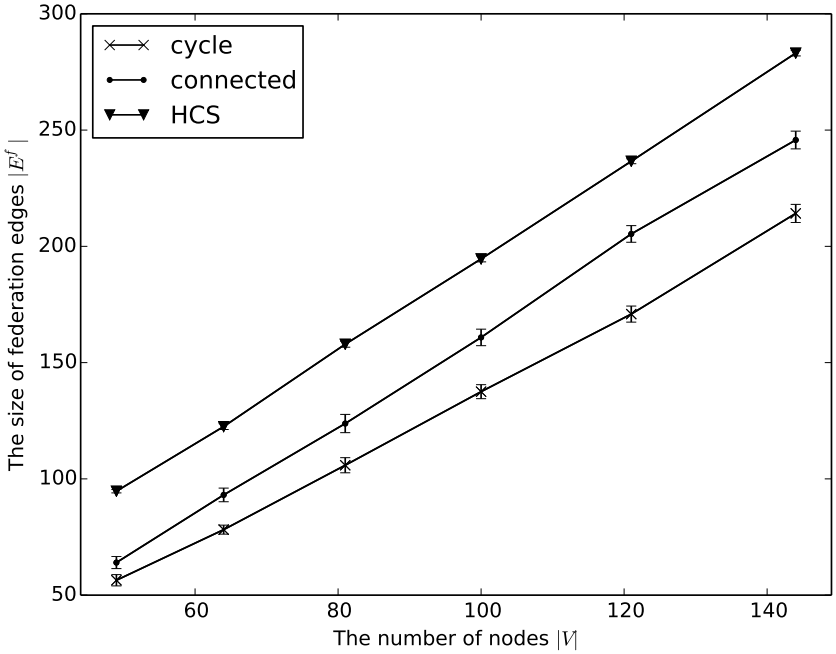


図 5.18: NWS グラフのノード数を変化させたときのフェデレーショングラフのリンク数 $|E^f|$

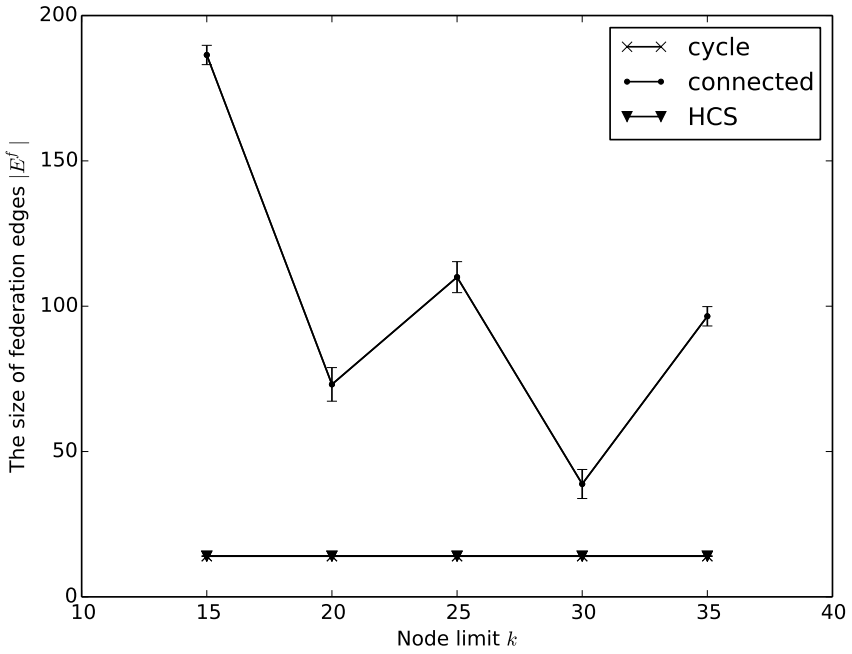


図 5.19: Caveman グラフにおいてクラスタのノード制限 k を変化させたときのフェデレーショングラフのリンク数 $|E^f|$

(ii) 管理範囲変更実験

図 5.19–図 5.23 にクラスタのノード制限数を変化させた管理範囲変更実験の結果を示す．これらの結果より，クラスタの上限を変化させたにおいても，サイクルクラスタリングはフェデレーショングラフサイズを小さく抑えることができている．図 5.19 では，ノード 20 と 30 の場合に connected クラスタリングの値が大きく減少しているが，これは TotalAP の時と同じく理由により，クラスタの探索が Cave を分断しない状態で終了しやすくなるためである．

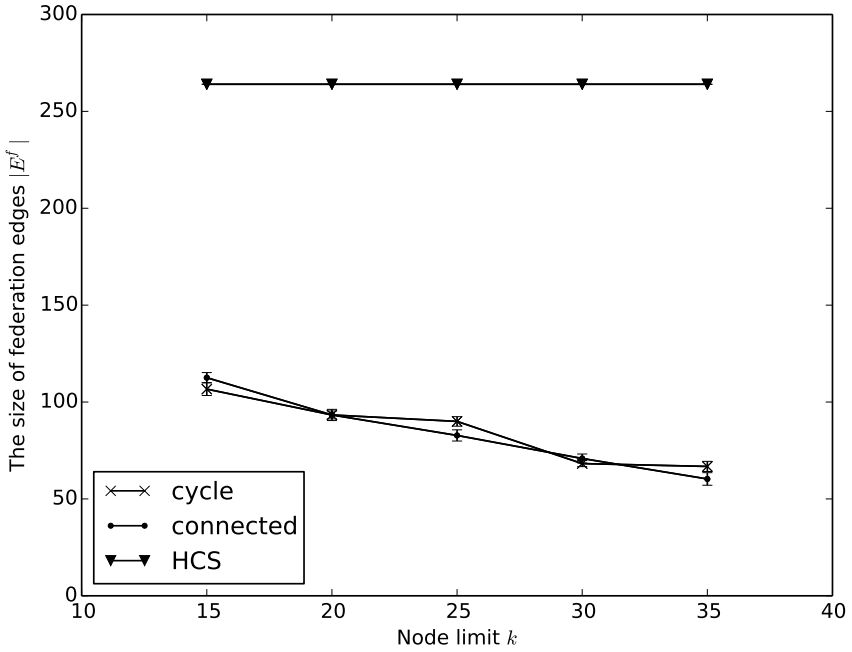


図 5.20: 格子状グラフにおいてクラスタのノード制限 k を変化させたときのフェデレーショングラフのリンク数 $|E^f|$

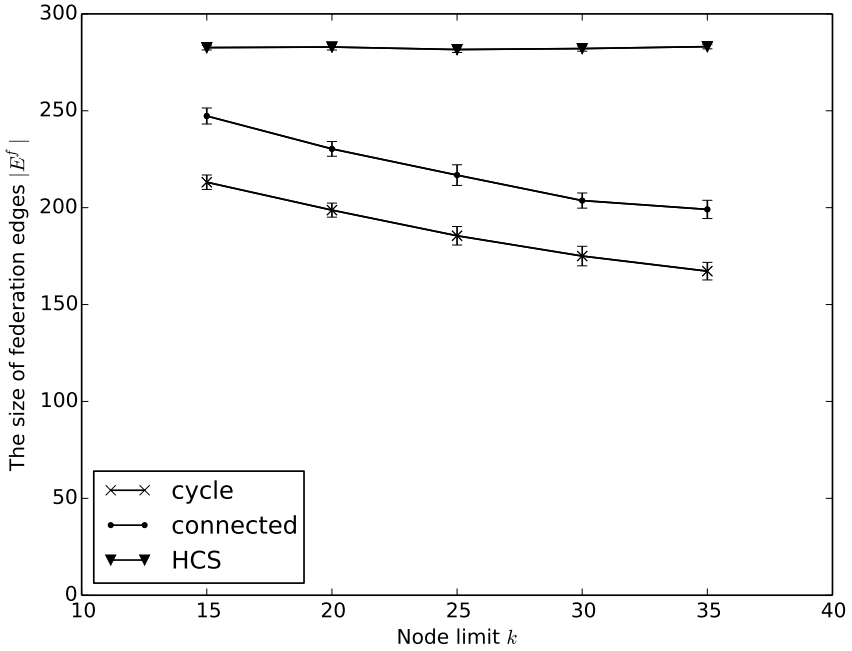


図 5.21: NWS グラフにおいてクラスタのノード制限 k を変化させたときのフェデレーショングラフのリンク数 $|E^f|$

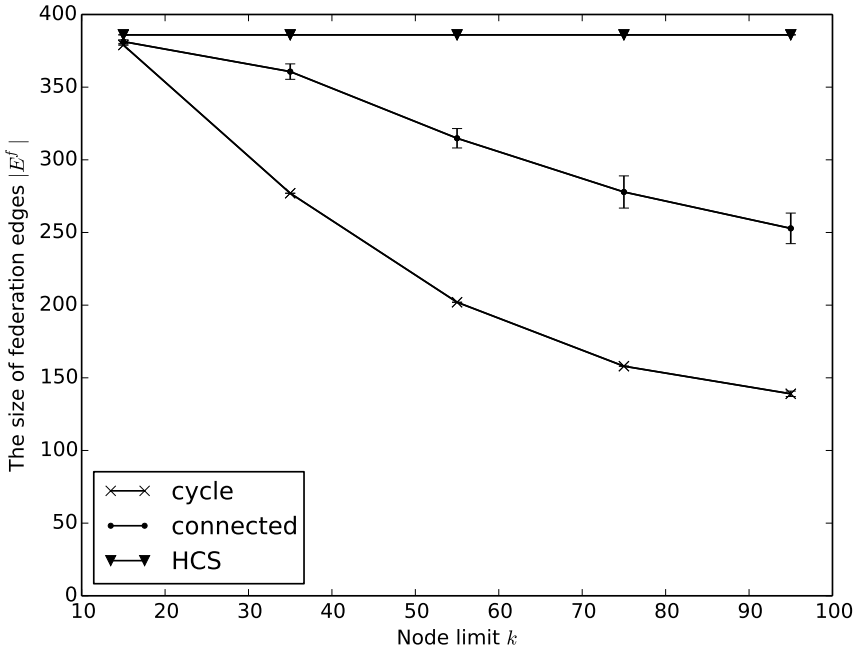


図 5.22: America グラフにおいてクラスタのノード制限 k を変化させたときのフェデレーショングラフのリンク数 $|E^f|$

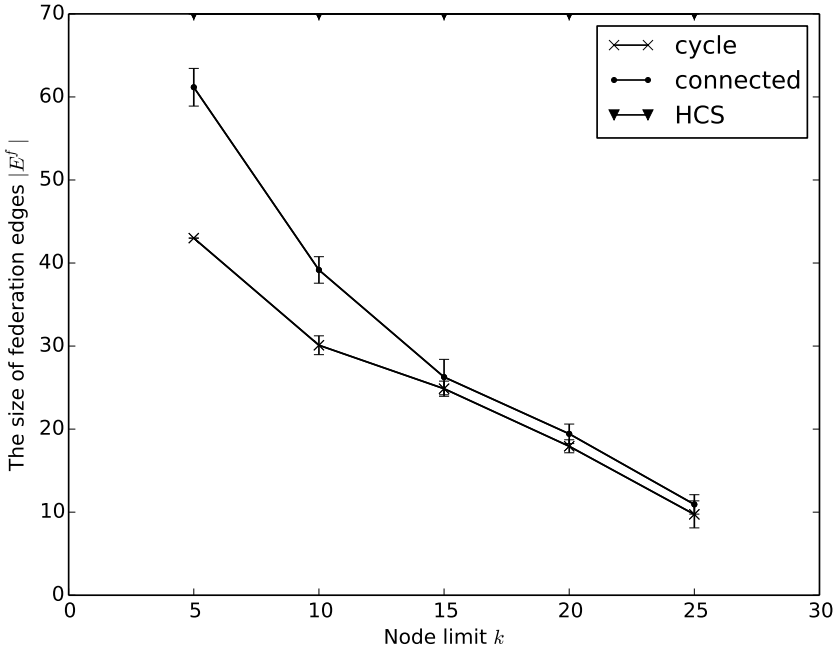


図 5.23: JPN48 グラフにおいてクラスタのノード制限 k を変化させたときのフェデレーショングラフのリンク数 $|E^f|$

5.4 まとめ

大規模なネットワークを制御する際，複数コントローラ間の情報共有により制御を実現することが考えられているが，制御するネットワークによっては共有する情報が多くなり，多量の情報を供するための負荷がコントローラへかかってしまう．そこで，共有される情報を隠蔽化し，負荷を軽減するため，スイッチを集約する手法が考えられてきたが，既存の手法では集約後のノードの信頼性が考慮していなかった．信頼性を考えて集約した場合には，隠蔽可能な情報が限られてしまう．そこで，本研究では，集約ノードの信頼性を高め，かつ，共有情報を削減するサイクルクラスタリングを提案した．実験結果より，サイクルクラスタリングは，ランダムグラフやデータセンターを模したグラフ，広域ネットワークをモデル化したグラフにおいて，集約ノードの信頼性を高め，かつ，共有する情報量を削減することが可能であることが分かった．

第6章　まとめ

本論文では，大規模化しているネットワークを制御可能なSDNの設計手法について述べた．SDNは，分散制御を用いたことにより発展してきたネットワーク制御に，集中制御を導入することにより制御変更の柔軟性を組み込もとしている．SDNを用いた実装例も多く出てきているが，より大規模なネットワーク制御を実現するためには，(1) コントローラへの負荷集中，(2) 制御チャネルの遅延の影響，(3) 制御ルール数の制約という問題点があった．

そこで，(3) の問題に着目し，ネットワークの規模が大きくなったとしても，フローエントリ数の増加が緩やかなフローエントリの作成方法を提案した．また，問題 (1) と (2) に対応するため，複数コントローラによるコントロールプレーンの構築方法について議論し，コントローラ間の情報共有を削減可能なサイクルクラスタリングを提案した．シミュレーションにより，サイクルクラスタリングは，ランダムグラフやデータセンターを模したグラフ，広域ネットワークをモデル化したグラフにおいて，集約ノードの信頼性を高め，かつ，共有する情報量を削減することが可能であることが分かった．以下に残された課題について述べる．

6.1　残された課題

6.1.1　タイセットに基づく障害復旧手法における最大フローテーブル数の解析

SDNではスイッチに登録可能な制御ルール数に限りがあるため，フローエントリ数を抑える必要があり，これまで，基本タイセット系を用いることで予備経路用のフローエントリを削減する手法を検討してきた．しかし，この手法は平均フローエントリ数の低減にのみ着目しており，その最悪値は考慮していなかった．多くのフローが通過するノード（中心ノード）のフローエントリ数が高くなる可能性がある．また，基本タイセット系の作成に用いる木の特徴によっても，平均フローエントリ数 (図 6.1) に対して，最大フローエントリ数 (図 6.2) が変化すること

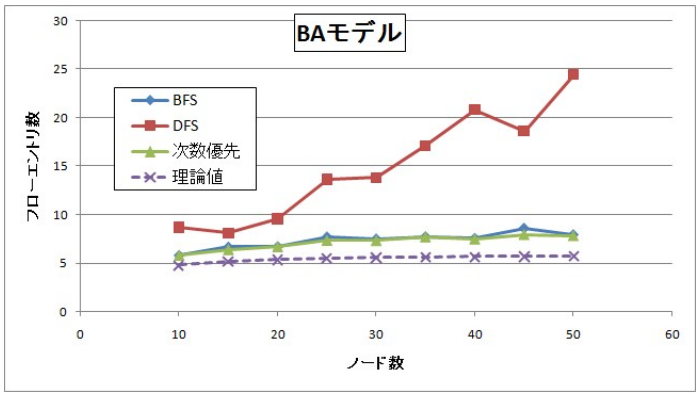


図 6.1: 平均フローエントリ数

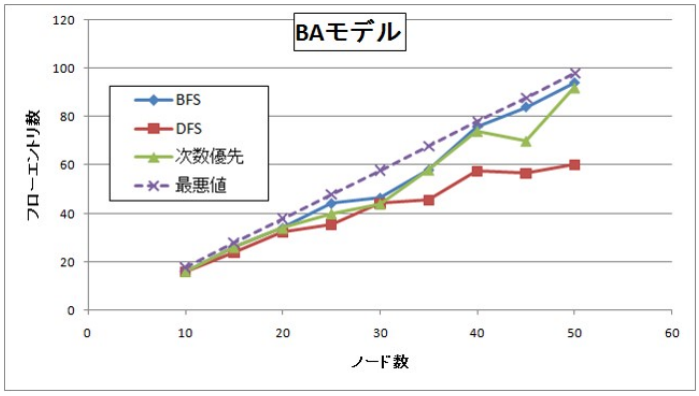


図 6.2: フローエントリ数の最悪値

が分かっている．より大規模なネットワークでの特性を調べるため，フローエントリ数の上限を求めることが必要となる．

6.1.2 彩色アルゴリズムを応用したタイセット ID の節約手法の提案

基本タイセット系を用いた障害復旧制御では，パケットヘッダの特定のフィールドにタイセット ID を書き込んでいる．必要なタイセット ID 数は，ネットワークの零度が高くなると増加し，ヘッダに書き込むことが出来る上限数を超えてしまう可能性がある．そこで，グラフの彩色問題を応用したタイセット ID の節約方法が必要となる．この方法では，リンクを共有しないタイセットには同じ ID を付与可能であることを利用し，ID 数を削減可能としているが，リンクの追加などのトポロジに変化があった場合には，彩色数が増大してしまう可能性がある．そこで，トポロジの変化に彩色を求める必要がある．

6.1.3 グループテーブルを使用した高速障害復旧方式の提案

SDN において，キャリアで要求されるような 50ms 以内の障害復旧を実現するためには，コントローラとスイッチ間の通信遅延を必要としない障害復旧処理が必要であることが指摘されている [11]．本論文では，OpenFlow1.0 の仕様を用いて動作可能な実装方法を提案したが，OpenFlow1.3 の仕様にはポートの先に障害が起きたときに，転送アクションを切り替える事が出来るグループテーブル（Fast failover）がある．これを利用することで，発生した障害をコントローラへ通知することなく障害を回避することが可能となった．そこで，現用経路を設定する際に，この経路に含まれる任意のリンク障害に対応可能なグループテーブルの作成方法を考える必要がある．この手法では，タイセットに基づく障害復旧を利用しており，事前にタイセットを回転させるためのフローエントリを設定しておき，障害時には，グループテーブルがヘッダ情報を変更することによりタイセットによる予備経路へ転送する．

6.1.4 クラスタリングにより削減可能な共有情報量の下限の解析

サイクルクラスタリングでは，情報量を最小化するクラスタリングを求めることは保証されていない．本研究では，より良いクラスタリングを求める手法をシミュレーションを用いて評価してきたが，より大規模なネットワークでの有効性を検証するためには，理論的な解析が必要となる．最適なクラスタリングを求めるアルゴリズムが存在するかの解析や共有情報の削減量の理論限界などを求めることが課題としてあげられる．

6.1.5 共有情報の削減によるコントローラ負荷の低減効果の計測

本研究では，コントローラの負荷を削減するため，共有情報量を削減するサイクルクラスタリングを提案してきた．シミュレーションにより，共有グラフのサイズを削減できていることは分かったが，実際にどの程度の負荷を軽減できているかは計測できていない．そこで，OpenDayLight，Ryu や Trema などの公開されているコントローラ作成のためのフレームワークを用いて，提案したクラスタリング手法を実装し，共有情報の削減がどの程度コントローラの負荷を減らすことが出来るのかを評価する必要がある．

付 録A　業績一覧

論文誌

1. 長野純一, 福田純一, 篠宮紀彦,“OpenFlow によるサイクル構造に着目した障害復旧方式の実装と評価,”電子情報通信学会論文誌. D, 情報・システム, vol. 96, no. 10, pp. 2340-2350, Oct. 2013.
2. J. Nagano and N. Shinomiya,“Efficient switch clustering for distributed controllers of OpenFlow network with bi-connectivity,”Computer Networks, The International Journal of Computer and Telecommunications Networking, ELSEVIER, Vol.9x, Pages xx-xx, DOI:10.1016/j.comnet.2015.10.017, Feb. 2016. (in press)

国際学会

1. J. Nagano and N. Shinomiya,“A Failure Recovery Method Based on Cycle Structure and Its Verification by OpenFlow,”in 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), 2013, pp. 298-303.
2. J. Nagano and N. Shinomiya,“Efficient information sharing among distributed controllers of OpenFlow network with bi-connectivity,”in 2015 International Conference on Computing, Networking and Communications (ICNC), 2015, pp. 320-324.

国内学会

1. 長野純一, 福田純一, 篠宮紀彦,“サイクル構造に着目した障害復旧方式の OpenFlow による有効性検証,”電子情報通信学会技術研究報告. NS, ネットワークシステム, vol. 111, no. 468, pp. 289-294, Mar. 2012.
2. 長野純一 篠宮紀彦,“サイクル構造に着目した障害復旧方式における OpenFlow の分散コントローラによる実装,”電子情報通信学会技術研究報告. NS, ネットワークシステム, vol. 113, no. 292, pp. 77-80, Nov. 2013.
3. 長野純一 篠宮紀彦,“Efficient information sharing among distributed controllers of OpenFlow network with bi-connectivity,”電子情報通信学会技術研究報告. NS, ネットワークシステム, vol. 114, no. 477, pp. 481-486, Mar. 2015.

参考文献

[1] 総務省, 平成 *27* 年版情報通信白書, 2015. [Online]. Available: http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h27/index.html

[2] 日経コンピュータ, すべてわかる *IoT* 大全モノのインターネット活用の最新事例と技術. 日経 BP 社, 2014.

[3] J.-P. Vasseur, M. Pickavet, and P. Demeester, *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Morgan Kaufmann, 2004.

[4] N. A. Lynch, "Distributed Algorithms," jan 1996.

[5] C. Hedrick, "Routing Information Protocol," *RFC 1058*, 1988. [Online]. Available: https://tools.ietf.org/html/rfc1058

[6] S. Hares, Y. Rekhter, and T. Li, "A Border Gateway Protocol 4 (BGP-4)," *RFC 4271*, 2006. [Online]. Available: https://tools.ietf.org/html/rfc4271

[7] IEEE802.1D, "Spanning Tree Protocol (STP)."

[8] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, oct 2005.

[9] I. Gahsinsky, "Warehouse-scale datacenters: the case for a new approach to networking," in *Open Networking Summit*, CA, USA, 2011. [Online]. Available: http://opennetsummit.org/archives/apr12/site/talks/gahinsky-tue.pdf

[10] M. Kobayashi, S. Seetharaman, G. Parulkar, G. Appenzeller, J. Little, J. van Reijendam, P. Weissmann, and N. McKeown, "Maturing of OpenFlow and Software-defined Networking through deployments," *Computer Networks*, vol. 61, pp. 151–175, mar 2014.

[11] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *2011 18th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN)*, oct 2011, pp. 1–6.

[12] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks." in *OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation*, vol. 10, 2010, pp. 1–6.

[13] S. Jain, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, and J. Zhou, "B4: Experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM '13*, vol. 43, no. 4. New York, New York, USA: ACM Press, aug 2013, pp. 3–14.

[14] I. Guis, "Enterprise data cener networks," in *Open Networking Summit*, Santa Clara, CA, USA, 2012. [Online]. Available: http://opennetsummit.org/archives/apr12/guis-mon-enterprise.pdf

[15] G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, and G. Hjálmtýsson, "Routing design in operational networks: a look from the inside," in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '04*, vol. 34, no. 4. New York, USA: ACM Press, aug 2004, pp. 27–40.

[16] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Computer Communications*, vol. 67, pp. 1–10, jun 2015.

[17] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, "Open signaling for ATM, internet and mobile networks (OPENSIG'98)," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 1, p. 97, jan 1999.

[18] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, jan 1997.

[19] J. Moore, M. Hicks, and S. Nettles, "Practical programmable packets," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society*, vol. 1. IEEE, 2001, pp. 41–50.

[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, mar 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1355734.1355746

[21] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, p. 105, jul 2008.

[22] Z. Cai, A. Cox, and T. Ng, "Maestro: A system for scalable openflow control," *Structure*, 2010.

[23] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, jun 2014, pp. 1–6.

[24] T. development team, "Trema." [Online]. Available: https://github.com/trema

[25] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 351–362, oct 2011.

[26] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies - CoNEXT '13*. New York, New York, USA: ACM Press, dec 2013, pp. 13–24.

[27] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, feb 2013.

[28] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in software defined networks," *Computer Networks*, vol. 71, pp. 1–30, oct 2014.

[29] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, jul 2014.

[30] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*. New York, USA: ACM Press, aug 2012, pp. 7–12.

[31] M. Kiaei, C. Assi, and B. Jaumard, "A Survey on the p-Cycle Protection Method," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 3, pp. 53–70, 2009.

[32] T. Koide, H. Kubo, and H. Watanabe, "A study on the tie-set graph theory and network flow optimization problems," *International Journal of Circuit Theory and Applications*, vol. 32, no. 6, pp. 447–470, nov 2004.

[33] K. Nakayama, N. Shinomiya, and H. Watanabe, "An Autonomous Distributed Control Method for Link Failure Based on Tie-Set Graph Theory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 11, pp. 2727–2737, nov 2012.

[34] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements," *Computer Communications*, vol. 36, no. 6, pp. 656–665, mar 2013.

[35] K. Kadena, K. Nakayama, and N. Shinomiya, "Network Failure Recovery with Tie-Sets," in *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*. IEEE, mar 2011, pp. 467–472.

[36] O. VSwitch, "Open vSwitch." [Online]. Available: http://openvswitch.org/

[37] S. H. Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*. New York, USA: ACM Press, aug 2012, pp. 19–24.

[38] U. Brandes and T. Erlebach, *Network Analysis*. Springer Berlin Heidelberg, 2005.

[39] E. Hartuv and R. Shamir, "A clustering algorithm based on graph connectivity," *Information Processing Letters*, vol. 76, pp. 175–181, 2000.

[40] M. Newman and D. Watts, "Renormalization group analysis of the small-world network model," *Physics Letters A*, vol. 263, no. 4-6, pp. 341–346, dec 1999.

[41] N. Shinomiya, T. Hoshida, Y. Akiyama, H. Nakashima, and T. Terahara, "Hybrid Link/Path-Based Design for Translucent Photonic Network Dimensioning," *Journal of Lightwave Technology*, vol. 25, no. 10, pp. 2931–2941, 2007.

[42] T. Sakano, Y. Tsukishima, H. Hasegawa, T. Tsuritani, Y. Hirota, S. Arakawa, and H. Tode, "A Study on a Photonic Network Model based on the Regional Characteristics of Japan (in Japanese)," in *IEICE Technical Report of Photonic Network*, vol. 113, no. 91, Fukushima, Japan, 2013, pp. 1–6.