医療情報ネットワークにおけるOpenFlowを用いた動的経路制御の一検討

岩崎 裕輔[1]　小野 悟[2]　猿渡 俊介[1]　渡辺 尚[1]

概要：本稿では OpenFlow を用いた医療情報ネットワークの動的経路制御について検討する．本研究では，高価かつ高性能なコアスイッチを，フルコネクトされた複数の安価な OpenFlow スイッチで構成されるコアネットワークで置き換える．このコアネットワークを用いることで，医療情報ネットワークにおける時間的なトラヒックの偏りに対して適応的に通信容量を割り当てることができる．リンクに障害が発生した場合でも，動的に経路を割り当てることで通信のロバスト性を提供する．シミュレーションによる評価の結果，本方式を用いることによって現状のコアスイッチに漸近する性能を発揮できることが確認できた．

1. はじめに

医療情報分野では，この四半世紀中で大きな変革があった．古くは汎用機を用いたレセコンと呼ばれる診療報酬請求明細書の計算と出力のために整備されていたコンピュータの活用は事務分野が中心であったが，ここ十数年の間に医療現場そのものまでに電子化が波及するようになった．平成 13 年度の e-Japan 構想に合わせて策定された厚生労働省の「保健医療分野の情報化にむけてのグランドデザイン」[1] では，その達成目標として，電子カルテに関しては平成 18 年度までに 400 床以上の病院 6 割以上に普及・全診療所の 6 割以上に普及を，レセプト電算処理システムに関しては平成 18 年度までに全国の病院 7 割以上に普及することを掲げていた．これらの目標を受けて，400 床以上の病院における電子カルテの普及率は平成 27 年現在で 70.1%を達成した [2]．電子カルテを導入することにより，ほぼすべての診療記録が電子化されることとなる．

このような電子化の流れの中で，医療情報ネットワークを流通するネットワークコンテンツは年々肥大化している．診療現場である外来ブースや，病棟のスタッフステーションには多くの端末が設置されている．病床数が千を超える特定機能病院では数千台規模の端末が常設されることも多い．検体検査，処方，X 線写真，生理検査等あらゆる医療行為は，絶えずこれらの端末から電子情報としてオーダされるため，受け手となる部門システムに確実に送信されなければならない．検体検査や X 線写真，生理検査等部門システムからの結果返信が必要な場合には，即座にこれらの情報を送り返す必要がある．特に外来ブースではこれらの送受信に係るレスポンスが患者の待ち時間や医師の業務効率に大きな影響を与える．

このような医療情報ネットワークにおいて，ネットワークシステムの高価格化が問題となっている．高価格化の要因は，医療情報ネットワークにおけるトラヒックの時変動性とロバスト性への要求の 2 つである．この 2 つの要因を，医療情報ネットワーク特有の「端末側のアプリケーションの変更を伴うアプローチが取れない」という制約下で対応しなければならない．これらの要因と制約に関しては 2 節で詳細に議論する．

このような観点から，本稿では，OpenFlow[3], [4], [5] を用いた医療情報ネットワークの動的経路制御について検討する．検討方式では，現在の医療情報ネットワークの高価なコアスイッチを複数の安価な OpenFlow スイッチで構成するフルメッシュのコアネットワークで代替する．時間帯によって特性の異なるトラヒックに対して経路を動的に割り当てることで各時間帯で通信の多いフローに通信容量を提供できる仕組みを実現する．また，リンクに障害が発生した場合でも障害の発生したリンクを迂回するように経路を割り当てることでロバスト性を確保する．

現在の医療情報ネットワークのトポロジを模した計算機シミュレーションによる評価では，検討方式は現状のコアスイッチに漸近する性能を発揮できることが分かった．

本稿の構成は以下の通りである．2 節では，現在の医療情報ネットワークの課題と制約について，トラヒックの特

[1] 大阪大学大学院情報科学研究科
Graduate School of Infomation Science and Technology, Ohsaka University
[2] 静岡大学創造科学技術大学院
Graduate School of Science and Technology, Shizuoka University

徴とロバスト性の要求の 2 つの側面から議論する．3 節では，提案手法の詳細について述べる．続く 4 節では，検討方式のスケーラビリティをシミュレーションによって評価する．最後に 5 節で本稿のまとめを行う．

2. 医療情報ネットワークの課題と制約

2.1 トラヒックの時変動性による高価格化

増加するスループットに対する要求の中で，医療情報ネットワークのトラヒックの時変動性がネットワークシステムの高価格化を招いている．医療情報ネットワークを流通するトラヒックはバースト的であり，ピーク時の性能が常時要求されることは少ない．図 1 に，H 大学病院で実際に稼働する医療情報ネットワークで収集したトラヒックを示す．Simple Network Management Protocol (SNMP) の RMON によるモニタは監視対象の装置に対する負荷が大きく，リアルタイム解析が困難であったため，Network Time Machine[6] を用いてトラヒック収集を行った．

Network Time Machine には，1 Gbps の Small Form-factor Pluggable (SFP) モジュールを最大で 4 つ搭載可能である．それぞれのモジュールから異なるセグメントのトラヒックを同時に収集することができる．3 TB のストレージを備えているので，長時間の連続した収集が可能である．今回の計測では，外来診療業務がピークとなる午前 10 時から午後 5 時までの時間帯で収集した．収集箇所は，電子カルテシステムのサーバが接続されている 4 台のサーバスイッチとコアスイッチがリンクされているポートである．コアスイッチには，院内すべての端末を収容する多くのエッジスイッチや，無線 AP が収容されている PoE スイッチが接続されているため，電子カルテを閲覧するためのトラヒックはこれらのポートを監視することですべて捕捉可能である．

図 1 より，医療情報ネットワークでは帯域を定常的に消費しているのではなく，偏差が大きく時間帯によって特性が異なるトラヒックであることがわかる．図 1 では，正午頃のピークと午後 4 時頃の 2 つのピークが確認できる．最大ピークは 2 Gbps を超えているが，ピーク外の時間帯は概ね 500 Mbps 未満のトラヒック量で推移している．

時間帯による特徴が分かりやすい例としては，バックアップのトラヒックが挙げられる．東日本大震災を契機に，災害下でも医療情報システムの継続性を保つための全国国立大学病院診療情報バックアップ事業（The Gemini Project）が平成 25 年度より開始されている [7]．各国立大学病院では，このバックアップの対象として，2 つのソースを広域ネットワークを介して保存する必要がある．1 つめは災害復旧時に利用するプロプライエタリなシステムファイルで 2 つめは事業に参加する全施設が災害時に共通して参照可能な標準化型ファイルである．これらのソースは一様に大容量であるため，ネットワークが高帯域に長時間専

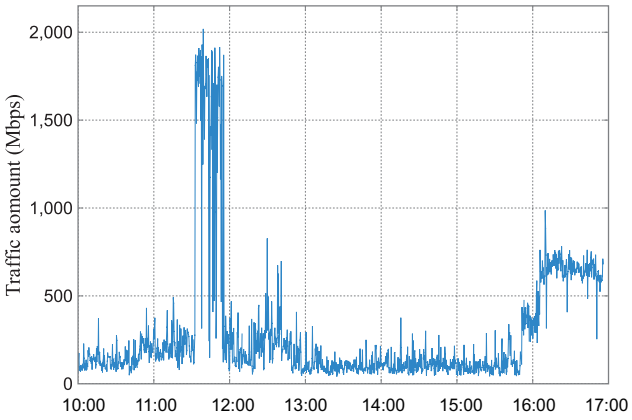

図 1 医療情報ネットワークのトラヒック推移

有されてしまう．

このようなピーク時の性能確保のために，現状ではネットワーク機器として高価格なものを用いなければならない．これまでの医療情報ネットワークでは，あらかじめ特定の宛先にトラヒックが集中することを予測し，その経路が含まれるリンクを Link AggreGation (LAG) などを用いて多重化していた．このような環境下では，トラヒックがバースト的である場合，多重化された帯域は常時利用されず，LAG 化されたポートは利用の有無に関わらず常時占有されてしまうという欠点がある．さらにバックアップタスクの存在はネットワーク機器の高価格化の一因でもある．バックアップタスクは綿密に実行のタイミングを計画しているものの，バックアップのソースは経年肥大するため，タスクスケジュールの定期的な見直し等，全体設計を含めてコスト増加の要因となっている．

2.2 ロバスト性の要求による高価格化

医療情報ネットワークでは，情報端末が多量に導入されるに従って相対的にネットワーク機器の故障率は低くなっているにも関わらず，ロバスト性への対応がネットワーク機器の高価格化を招いている．近年では，医療分野においてもスマートフォンやタブレットなど多様な端末装置が利用されるようになっている [8]．これらの端末装置は可搬性が高いため，ベッドサイドでの入力事象発生時の発生源入力をはじめ，各種医療行為に対する 3 点認証（患者，医療行為，施行者）への活用等に用いられていることから，トラヒックに対する高いロバスト性が求められている．電子カルテシステムでは，端末装置だけでなく多種の機器が稼働している．電子カルテシステムの主要構成要素の障害は人命に影響を及ぼす可能性があることからも，高いロバスト性が要求されている．

現状の医療情報ネットワークでは，システムへのロバスト性に対して高価な機器や多くのマネジメント費用で対応している．ロバスト性の向上を図るために，現状では主要構成要素を冗長化している．冗長化によって全体のロバス

ト性は向上するが，反面で高額な費用が必要となる．特に，ネットワークの中心となるコアスイッチは高性能化の一途を辿っており，シャシー型コアスイッチを用いるなど大きなコスト増加の要因となっている．

しかしながら，近年は構成機器の信頼性が向上したことで相対的にネットワーク機器の故障率が低下しているため，これら冗長構成が有効活用される場面は少ない．表 1 に平成 28 年度の H 大学病院の医療情報トータルシステムにおける機器別障害発生件数の比率を示す．ラップトップ型コンピュータの障害が最も多く，プリンタの障害件数がこれに次ぐ．これに対し，サーバやネットワーク等主要構成要素の故障が非常に少ないことがわかる．例えば，Virtual Router Redundancy Protocol (VRRP) 等を用いて冗長構成とした場合には，通常バックアップ側の機器は稼働しないため，マスタ側の機器に障害等が発生しない限りバックアップ側が活用されることはない．シャシー型スイッチで装置内冗長構成をとった場合，CISCO L3 スイッチでのスーパバイザ冗長化では一方のエンジンは停止している．両エンジンは同時に稼働しない．すなわち，現在の医療情報ネットワークでは，高価かつ高性能である機器の性能が十分に活用されていない．

表 1　機器別の障害発生件数の比率

デバイス名	比率
raptop	47 %
printer	22 %
desktop	18 %
PDA	10 %
storage	1 %
server	1 %
network	1 %

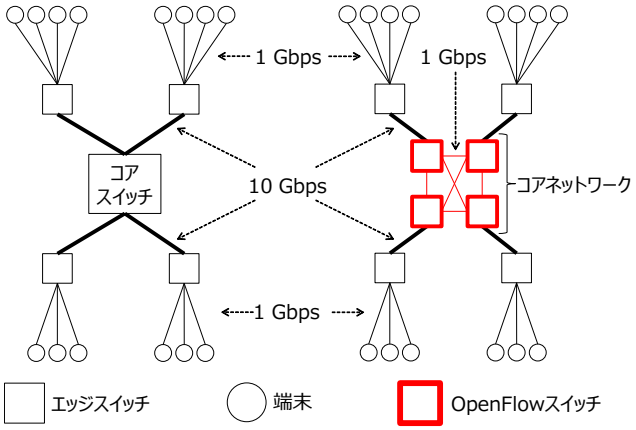

図 2　現在の医療情報ネットワーク (左) と検討方式 (右)

2.3　医療情報ネットワークの制約

2.1 節，2.2 節で述べたように，医療情報ネットワークの課題はトラヒックの時変動性とロバスト性の要求に対応するためにネットワーク機器が高価格になっていることである．このような問題に対して，e-science やデータセンタネットワーク等で検討されているような端末側に MPTCP の機能を持たせたりルートを選択可能な機能を持たせたりすることができれば解決できる可能性がある [9], [10].

しかしながら，医療情報ネットワークでは，端末側のシステムを変更するというアプローチを採ることが困難である．電子カルテシステムは，単独のベンダがすべての機能を網羅的に提供しているのではなく，多くのベンダが協調して諸機能を提供するマルチベンダ型のシステムであるからである．電子カルテを閲覧するための端末装置には，各ベンダから提供された多くのアプリケーションが相互干渉しないように検証された上でインストールされている．また，各ベンダから提供される固有の端末装置も多く存在する．例えば，これらの端末装置すべてに対して一律の構成変更を行う場合には，設定・検証作業のための多くの時間と費用が必要となる．さらに，これらの端末装置の中には，薬事法で定められた医療機器として承認されているものが存在する．医療機器承認番号を取得しているこれらの機器は，承認申請時の機器構成が原則として変更できない．

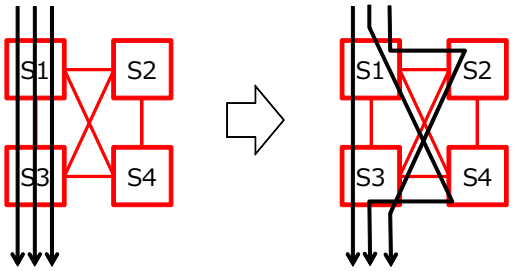

図 3　輻輳発生時の動作

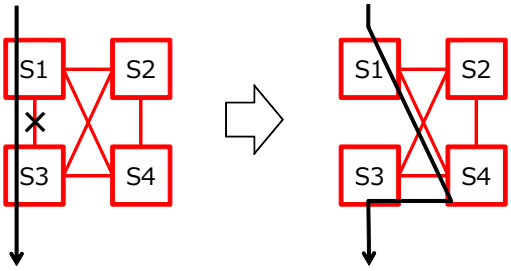

図 4　障害発生時の動作

3.　OpenFlow 医療情報ネットワーク

2 節での議論を元に，OpenFlow を用いた医療情報ネットワークの設計を行った．

OpenFlow の実装では，コントローラの制御アプリケーションが必要である．アプリケーション構築のためのさまざまなフレームワークが存在する．代表的なものに，NOX[11]，Beacon[12]，Maestro[13]，Floodlight[14]，Trema[15] などが挙げられる．

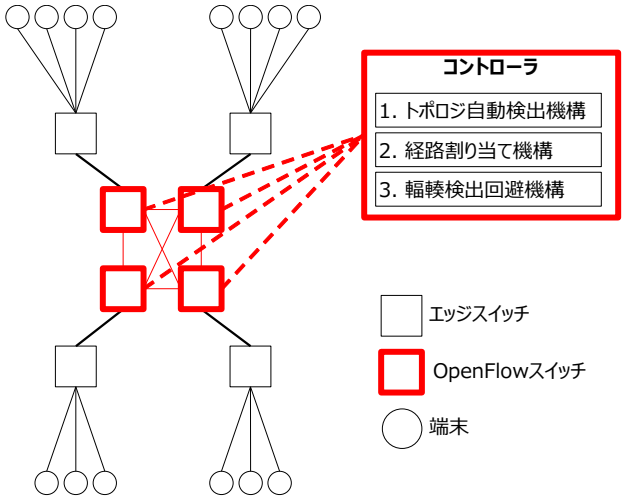

図 5　検討方式の全体像

表 2　検討方式のデータベース

ofswitchTable	OpenFlow スイッチ一覧
linkTable	OpenFlow スイッチ同士の対応関係
hostTable	端末と OpenFlow スイッチの対応関係
pathTable	コアネットワークの全ての経路
hopTable	各経路を構成するホップ
flowTable	フローと経路の対応関係
trafficTable	コアネットワークのトラヒック量

3.1　全体像

図 2 に本研究の基本的なアイディアを示す．検討方式では，図 2 左のように高価格化の要因であった従来の医療情報ネットワークにおけるコアスイッチを，図 2 右のようにフルメッシュのコアネットワークに置き換える．このコアネットワークは，安価な OpenFlow スイッチをフルコネクトして構築したネットワークである．

検討方式では，輻輳しているリンクを回避しながらフローを動的に経路に対して割り当てることで，各フローのスループットを向上する．図 3 に輻輳が発生した場合のコアネットワークの動作を示す．OpenFlow スイッチ S1〜S4 は相互に 1 Gbps のリンクで接続されている．図 3 左のように，S1 から S3 に流れる 1 Gbps のフローが 3 つあった場合，S1 から S3 のリンクが輻輳を起こして各フローは約 0.33 Gbps しかスループットがでなくなる．検討方式では，図 3 右のようにフローに対して輻輳を避けるように動的に経路を割り当てることで，総スループットを向上させることができる．

また，検討方式は障害発生時には障害が発生したリンクを避けるように動的に経路を割り当てることもできる．図 4 に障害発生時の動作を示す．図 4 では，S1 から S3 に対して 1 Gbps の 1 つのフローが存在する．S1 から直接 S3 に配送していた際に S1 と S3 を接続するリンクに障害が発生した場合，自動的に S1→S4→S3 と経由するように経路を切り替えることができる．

図 5 に検討方式の全体像を示す．各 OpenFlow スイッチはコントローラと接続されている．コントローラは，OpenFlow の機能で実現する

(1) トポロジ自動検出機構
(2) 経路割り当て機構
(3) 輻輳検出回避機構

の 3 つの要素から構成される．全ての要素は OpenFlow の機能のみで実現可能であるため，端末側の変更は不要である．

表 2 にコントローラが具備すべきテーブルの一覧を示す．

ofswitchTable はコアネットワークを構成する OpenFlow スイッチの情報を保持しているテーブルである．各 OpenFlow スイッチは datapath ID で一意に識別することができる．

linkTable は OpenFlow スイッチ同士の接続を保持しているテーブルである．接続の両端の OpenFlow スイッチの datapath ID，それぞれ接続しているポート番号を組として保持している．

pathTable はコアネットワークが持ちうる全ての経路を保持したテーブルである．各項目は経路を一意に識別する path ID，経路の先頭の OpenFlow スイッチ，経路の終端の OpenFlow スイッチ，経路の長さを保持している．

hopTable は pathTable に保持されている経路がどの OpenFlow スイッチを経由して構成されるかの情報を保持している．各項目は path ID，datapath ID，ポート番号で構成される．検討方式では，経路を経由する datapath ID と出力ポート番号の組の配列として表現している．ある path ID の経路でパケットを転送する場合に，ある datapath ID を持つ OpenFlow スイッチがどのポートで出力すれば良いかの情報を各項目として保持している．

flowTable はフローと経路の対応関係を保持したテーブルである．本稿の説明では，説明の簡単化のためにフローを送信元 MAC アドレスと宛先 MAC アドレスで表現している．検討方式はフローの識別に IP アドレスや TCP のポートを用いるように拡張可能である点に注意されたい．

trafficTable は OpenFlow スイッチ間を接続するリンクのトラフィック量を記録しているテーブルである．各項目はタイムスタンプ，datapath ID，ポート番号，累計送信データ量から構成されている．

3.2　トポロジ自動検出機構

トポロジ自動検出機構はコアネットワークのトポロジを把握するための機構である．検討方式では，OpenFlow の機能を利用して自動的にトポロジを把握することができるため，新しい OpenFlow スイッチを接続するだけでコアネットワークを容易に拡張することができる．トポロジ自動検出機構で取得したトポロジ情報を元に経路の割り当てや輻輳の回避する経路を決定する．経路を算出するため

には，
(1) OpenFlow スイッチ同士の相互接続状況
(2) 端末の接続状況
(3) リンクの切断
を把握する必要がある．

(1) OpenFlow スイッチ同士の相互接続状況は，OpenFlow の features メッセージと Link Layer Discovery Protocol (LLDP) パケットで把握する．検討方式では，OpenFlow スイッチが接続される時に発行される hello メッセージを受け取ると，features メッセージを用いて接続された OpenFlow スイッチの情報を取得する．features メッセージでは，OpenFlow スイッチ自体の MAC アドレスと，その OpenFlow スイッチの各ポート番号を取得することができる．取得した情報を元にして，コントローラは各ポートから LLDP パケットを送信するように通知する．OpenFlow スイッチが LLDP パケットを受け取ると packet-in メッセージが発行される．packet-in メッセージにはどの OpenFlow スイッチからどのポートを経由して LLDP パケットを受け取ったかの情報が含まれている．

取得できる各 OpenFlow スイッチの MAC アドレスと LLDP パケットで取得できる情報を組み合わせることで OpenFlow スイッチ同士がどのように接続されているかを把握することができる．features メッセージで取得した情報は ofswitchTable に，LLDP パケットで取得した情報は linkTable に記録される．

(2) 端末の接続状況は，OpenFlow の packet-in メッセージを使用して把握する．packet-in メッセージとは，パケットが OpenFlow スイッチに届いた場合に，フローテーブルにそのパケットと適合するエントリが存在しなかった場合にコントローラに対して発行されるメッセージである．packet-in メッセージがコントローラに届いた際に，packet-in メッセージを送信した OpenFlow スイッチの識別子とポート番号，packet-in メッセージに含まれる送信元 MAC アドレスを取得する．送信元 MAC アドレスがコントローラに初めて届いた MAC アドレスであった場合に，取得した OpenFlow スイッチ識別子とポート番号の方向に端末が存在することを hostTable に記録する．

(3) リンクの切断は，OpenFlow スイッチからの port-status メッセージと各 OpenFlow スイッチ間で定期的に LLDP パケットを交換することで検出する．port-status メッセージはポートの状態が変化したときに発行されるメッセージである．各 OpenFlow スイッチからの LLDP パケットの送信はコントローラから制御する．LLDP パケットに該当するフローエントリは作成しないように設計しているため，各 OpenFlow スイッチが LLDP パケットを受け取るとコントローラに必ず packet-in メッセージが送られる．コントローラにおいて，port-status メッセージでリンクダウンを受け取るか，各ポートから LLDP パケットを一定期間受け取らなかったと判定した場合には，そのリンクはダウンしたとしてトポロジ情報に反映する．

Algorithm 1 packet-in イベントハンドラ

```
1: Input m: packet-in message
2: d_now ← getDatapathId(m)
3: a_src ← getSourceMacAddress(m)
4: a_dst ← getDestinationMacAddress(m)
5: p ← flowTable[(a_src, a_dst)]
6: if p = NULL then
7:     d_src ← hostTable[a_src]
8:     d_dst ← hostTalbe[a_dst]
9:     p ← getShortestFromPathTable(d_src, d_dst)
10:    flowTable[(a_src, a_dst)] ← p
11: end if
12: q ← getPortFromHopTable(d_now, p)
13: addFlowEntry(d_now, q, m)
14: sendPacketOut(d_now, q, m)
```

3.3 経路割り当て機構

経路割り当て機構はコアネットワークにおいて端末間のパケットを転送するための経路を決定するための機構である．経路の決定のための計算コストが大きいと通信性能を損なってしまうため，少ない計算コストで実現できることが望ましい．例えば，packet-in メッセージが来たタイミングで経路探索を行うとフローの種類の数だけ経路探索コストが生じてしまう．経路割り当てに伴う計算コストを少なくすることを目的として検討方式における経路割り当て機構では

- 経路と端末を分離
- OpenFlow スイッチ間の経路のみ事前計算で全て列挙

の 2 つの工夫を行っている．

経路と端末の分離は hostTable，pathTable，flowTable によって実現される．hostTable は端末の接続先情報を MAC アドレスから datapath ID に変換する仕組みである．flowTable はフローと経路を対応付けする仕組みである．検討方式では hostTable と flowTable の存在により経路を datapath ID とポート番号だけで表現できるため，計算コストも OpenFlow スイッチの数だけ考慮すれば良い．

また，経路は事前計算によってあらかじめ全て列挙されて pathTable と hopTable に記録されている．3.2 節で述べたトポロジ自動検出機構で取得した情報を元に，各 OpenFlow スイッチ間のパスを幅優先探索で経路を発見する．コアネットワークで用いる OpenFlow スイッチは多くても 10 台程度を想定しているため，幅優先探索で経路を全て列挙しても計算量は問題にならない．3.3 節で述べる経路割り当ての際に最短経路を選択できるように，各経路はホップ数と一緒に記録している．

Algorithm 1 に packet-in イベントハンドラのアルゴリズムを示す．コントローラに packet-in メッセージが到着すると，packet-in メッセージに含まれる送信元 MAC ア

Algorithm 2 port statistics reply イベントハンドラ

```
1: Input m: port statistics reply message
2: d ← getDatapathId(m)
3: p ← getPortNumber(m)
4: b ← getTxBytes(m)
5: t ← now()
6: (t_pre, b_pre) ← getPreviousFromTrafficTable(d, p)
7: r ← (b − b_pre)/(t − t_pre)
8: if r > α then
9:     call congested_event_handler(d, p)
10: end if
11: insertToTrafficTable(d, p, t, b)
```

ドレスと宛先 MAC アドレスを抽出する．抽出した送信元 MAC アドレスと宛先 MAC アドレスの組をフローとして，既にその MAC アドレスの組に経路が割り当てられているかを調べる．もし既に経路が割り当てられていた場合には次にどの OpenFlow スイッチに送るべきかを取得してフローエントリに登録すると共にパケットを送信する．

もしまだ経路が割り当てられていなかった場合には，抽出した宛先 MAC アドレスから hostTable の情報を元に最終的にどの OpenFlow スイッチに送るべきかを取得する．次に，packet-in メッセージを受け取った OpenFlow スイッチの datapath ID を先頭，取得した OpenFlow スイッチの datapath ID を終端とする経路の中で最短の経路を pathTable から取得する．最後に，取得した経路を元に次にどの OpenFlow スイッチに送るべきかの情報を取得してフローエントリと flowTable に登録すると共にパケットを送信する．

OpenFlow スイッチやリンクに障害が発生された場合には，まず，関連する情報を ofswitchTable，linkTable，pathTable，hopTable から削除する．通常であれば，次に行うべきは OpenFlow スイッチから関連するフローエントリを削除することである．しかしながら，OpenFlow スイッチから関連するフローエントリを消すと再度 packet-in メッセージが発生してオーバヘッドになってしまう．オーバヘッドを削減するために，検討方式では，代替ルートを先に OpenFlow スイッチにフローエントリとして登録してから OpenFlow スイッチから障害が発生したリンクに関連するフローエントリを削除する．

3.4 輻輳検出回避機構

輻輳検出回避機構は，コアネットワーク内のリンクで発生している輻輳を検出し，輻輳が発生しているリンクを回避するように経路を再設定する機構である．フローの数を F，ノード数を N とすると，フローに対して割り当て可能な経路のバリエーションは $O(N^F)$ となる．医療情報ネットワークではフローは数が膨大かつ時々刻々と変化するため，可能な限り軽量な仕組みで輻輳の検出と回避ができることが望ましい．

Algorithm 3 congested_event_handler 関数

```
1: Input d: datapath ID, p: port number
2: F ← getAllocatedFlowSet(d, p)
3: if |F| = 1 then
4:     return
5: end if
6: (a_src, a_dst) ← choose a flow randomly from F
7: Q ← getPathSetFromPathTable(a_src, a_dst)
8: while |Q| > 0 do
9:     q_new ← get a path randomly from Q
10:    if q_new does not include (d, p) then
11:        q_old ← flowTable[(a_src, a_dst)]
12:        flowTable[(a_src, a_dst)] ← q_new
13:        add flow entries related to q_new
14:        delete flow entries related to q_old
15:        return
16:    end if
17:    Q = Q − q
18: end while
```

検討方式では，輻輳の検出を OpenFlow の機能を用いて各ポートの単位時間あたりの通信量を取得して閾値基準で輻輳の判定を行う．まず，コントローラは各 OpenFlow スイッチに対して multipart メッセージを用いて port statistics request を定期的に発行する．この時，OpenFlow スイッチ同士を接続しているポートのみの情報を要求することでトラヒック情報を取得する際の負荷を削減している．各 OpenFlow スイッチから port statistics reply が返ってくると port statistics reply イベントハンドラを実行する．

Algorithm 2 に port statistics reply イベントハンドラの動作を示す．port statistics reply が返ってきたらそのメッセージからそのメッセージを送った OpenFlow スイッチの datapath ID，ポート番号，送信バイト数を取得する．OpenFlow の port statistics reply に含まれている送信バイト数はその時点までの累計値である．現段階でのそのポートのトラヒック量を算出するために，その datapath ID, ポート番号において前回取得した送信バイト数 b_{pre} と取得した時刻 t_{pre} を getPreviousTrafficRecord 関数を用いて取得する．最新の送信バイト数と前回取得した送信バイト数の差分，現在の時刻と前回取得した時刻の差分から現在のそのポートにおけるトラヒック量を算出する．トラヒック量が閾値 α を超えていた場合には輻輳が発生していると判定して congested_event_handler 関数を実行する．congested_event_handler 関数に関しては後述する．閾値 α は現在の実装ではワイヤレートの 90 %を使用している．例えばワイヤレートが 1 Gbps のリンクの場合には，α は 900 Mbps となる．最後に取得した送信バイト数を insertTrafficStats 関数でデータベースに記録する．

congested_event_handler 関数は輻輳しているリンクを回避するように経路を再割り当てする関数である．関数内では経路の計算は行わず， 3.3 節で述べたあらかじめ算出済

みの経路をランダムに選択するだけの処理で実現している．ランダム性を導入しているのも軽量な処理で輻輳を回避するためである．ランダムに経路を選択する処理はほとんど計算コストがかからない．経路選択後に再び輻輳が発生する場合には，再度 congested_event_handler 関数が呼ばれるため，最終的に適切な経路割り当てが行われるか，輻輳を起こすトラヒックがなくなるかするまで経路の再割り当てが行われ続ける．

Algorithm 3 に congested_event_handler 関数の動作を示す．congested_event_handler 関数は入力としてどの datapath ID のどのポートで輻輳が起こっているかの情報が与えられる．まず，引数として与えられた datapath ID d とポート番号 p を用いて，既に経路の割り当てられたフローの中で (d, p) を使用しているフローを全て取得する．もし取得したフローが 1 つしかなかった場合，経路の変更は必要ないので関数の処理を終了する．もし (d, p) を含む経路に割り当てられたフローが 2 個以上存在した場合には，その中から 1 つを経路変更候補としてランダムに選択する．7 行目では，経路変更候補のフローに割り当て可能な経路を全て取得する．8 行目から 17 行目では，集合 Q の中から 1 つずつ経路を取り出し，現在輻輳が起きているリンクを含まない場合にはその経路をフロー $(a_{\mathrm{src}}, a_{\mathrm{dst}})$ に割り当てる新しい経路として利用する．

3.2 節で述べた障害発生時と同様に，経路変更時のオーバヘッドを削減する仕組みを輻輳発生時にも導入している．具体的には，Algorithm 3 の 13 行目において先に新しい経路のフローエントリを関連する OpenFlow スイッチに登録してから 14 行目において古い経路のフローエントリーを削除することでシームレスな経路変更を実現可能とした．

4. シミュレーションによる評価

本節では，計算機シミュレーションを用いて検討方式の評価を行う．

4.1 評価環境

図 6 に本評価で用いる実際の医療情報ネットワークを基準にしたトポロジを示す．各ネットワークは OpenFlow スイッチ，エッジスイッチ，端末で構成される．OpenFlow スイッチの数は OpenFlow スイッチの数が増えた場合にスケールアウトできているかどうかを確認するために 4 台から 10 台と変化させた．各 OpenFlow スイッチは相互に各 1 Gbps でフルコネクトで接続してコアネットワークを構築している．エッジスイッチの数は実際の医療情報ネットワークと揃えることを目的として 4 台の固定とした．端末は各フロー毎に送信元と宛先が 1 台ずつ独立に存在する．各フローは 1 Gbps のトラヒックとした．本評価の元となった医療情報ネットワークではエッジスイッチと OpenFlow スイッチは 10 Gbps のリンクで接続されている．評価結果

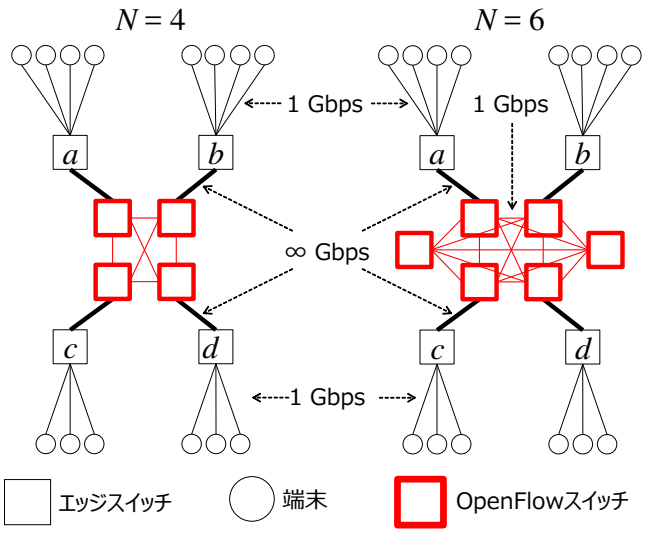

図 6　評価トポロジ．$N = 4$ の場合 (左) と $N = 6$ の場合 (右)

の分かりやすさを考慮して，本評価ではエッジスイッチと OpenFlow スイッチは無限の容量を持つリンクで接続されていることとした．比較対象として現在の医療情報ネットワークシステムで用いられているコアスイッチ (グラフ中の core switch) を用いた．コアスイッチの性能は無限大としている．

4.2 偏ったフローに対する評価

偏ったフローが発生した場合の性能を計算機シミュレーションによって評価した．偏ったフローとは，コアネットワークに対してある時間帯に特定のエッジスイッチ方面からの流入と，ある特定のエッジスイッチ方面への流出が集中した状態を模擬している．図 6 におけるエッジスイッチ a からエッジスイッチ b へのフローが複数発生している状態である．偏ったフローの例としては，深夜帯に発生するバックアップタスクによって生じるフローが挙げられる．

図 7 に偏ったフローでの評価結果を示す．横軸はエッジスイッチ a からエッジスイッチ b へのフロー数を，縦軸は

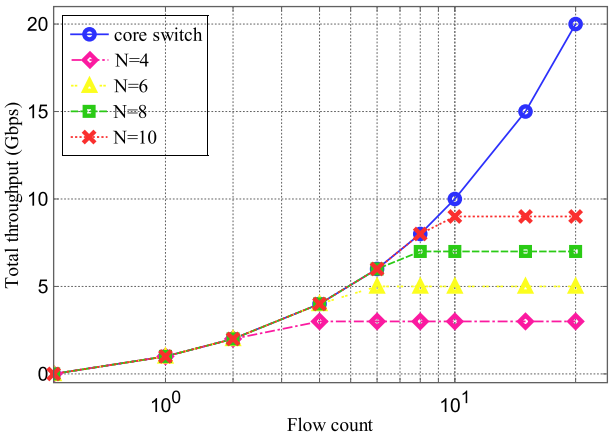

図 7　偏ったフローによる評価

端末間での総スループットを示している．コアネットワークを構成する OpenFlow スイッチの台数 N を変化させた場合の各フロー数における総スループットと，既存のコアスイッチを用いた場合 (core switch) をそれぞれプロットしている．既存のコアスイッチでは，与えられたフロー数に比例してスループットが増加していることが確認できる．一方で，検討方式では OpenFlow スイッチの数に応じて上限が存在しているものの，OpenFlow スイッチの数を増やすと上限がスイッチ数に比例して増加していることが分かる．また，上限の増加は既存のコアスイッチに漸近していることも分かる．複数のフローが流入元のエッジスイッチと流出先のエッジスイッチを共有しているため，各フローを 1 Gbps とすると上限は $N-1$ [Gbps] となっている．

4.3 ランダムなフローに対する評価

4.2 節の評価では，偏ったフローにおける評価であった．実際の医療情報ネットワークでは，多様なトラヒックが多様な流入元，流出先の組み合わせで発生する時間帯も存在すると考えられる．このような観点から，流入エッジスイッチと流出エッジスイッチをランダムに選択してフローを発生させた場合の評価を行った．

図 8 にランダムなフローにおける評価結果を示す．横軸はフロー数，縦軸はスループットを示している．4.2 節と同様に，コアネットワークを構成するスイッチ台数 N が変化した場合の各フロー数におけるスループットをプロットしている．検討方式では OpenFlow スイッチの数に応じて上限が存在しているものの，OpenFlow スイッチの数を増やすと上限がスイッチ数に比例して増加していることが分かる．また，上限の増加は既存のコアスイッチに漸近していることも分かる．さらに，各 OpenFlow スイッチ数での性能はそれぞれ 4.2 節の総スループットより大きいことも分かる．検討方式の仕組みにより，流入元スイッチ，流出先スイッチが多様な場合にはフローが偏っている場合に比べてより多様なリンクの容量を有効活用できているからだと考えられる．

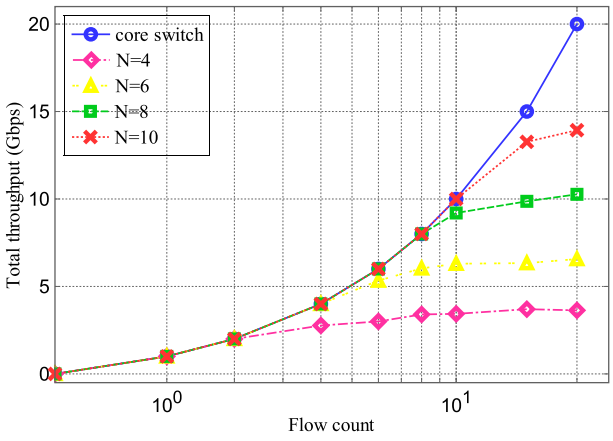

図 8　ランダムなフローによる評価

5. おわりに

本稿では，現在の医療情報ネットワークの高価格化を解決する手法として，OpenFlow を用いた医療情報ネットワークの動的経路制御手法を検討した．検討方式では，従来の医療情報ネットワークの高価なコアスイッチをフルメッシュコアネットワークに置き換えることで高価格化の問題を解決する．このコアネットワークは，フルコネクトされた複数の安価な OpenFlow スイッチで構成される．フローに応じて動的に経路を変更することで，時々刻々と変化するトラヒックの変動やリンク断などの障害にも対応できる．複数台のノードで構成されるコアネットワークを用いたシミュレーションでは，検討方式が現在の医療情報ネットワークで利用されているコアスイッチに漸近する性能を有することが確認できた．

謝辞

本研究は JSPS 科研費 JP16H01718 等の助成を受けて行った．

参考文献

[1] 厚生労働省：保健医療分野の情報化にむけてのグランドデザインの策定について，http://www.mhlw.go.jp/shingi/0112/s1226-1.html.

[2] 一般社団法人保健医療福祉情報システム工業会：オーダリング・電子カルテシステム病院導入状況調査報告書，https://www.jahis.jp.

[3] Mckeown, N., Shenker, S., Anderson, T., Peterson, L., Turner, J., Balakrishnan, H. and Rexford, J.: OpenFlow: Enabling Innovation in Campus Networks, *ACM SIGCOMM Computer Communication Review*, Vol. 38, pp. 69–74 (2008).

[4] Kolasani, A. and Ramamurthy, B.: Network Innovation using OpenFlow: A Survey, *IEEE Communications Surveys and Tutorials*, Vol. 16, pp. 493–512 (2014).

[5] Hu, F., Hao, Q. and Bao, K.: A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation, *IEEE Communications Surveys and Tutorials*, Vol. 16, pp. 2181–2206 (2014).

[6] Fluke Networks: Network Time Machine, http://www.toyo.co.jp/ict/products/detail/clearsight.html.

[7] 田中勝弥：SINET L2VPN を用いた国立大学病院災害対策医療情報システムにおける遠隔バックアップシステムの構築，http://w4a.sinet.ad.jp/storage/advnet2014-04.pdf (2014).

[8] 荒木賢二，中武豊伸：IT System Innovation Review Android 対応システム「WATATUMI」のパフォーマンスを検証する，月刊医療，Vol. 38, pp. 60–62 (2011).

[9] Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D. and H, M.: Improving Datacenter Performance and Robustness with Multipath TCP, *ACM SIGCOMM Computer Communication Review*, Vol. 41, pp. 266–277 (2011).

[10] Cao, Y., Xu, M., Fu, X. and Dong, E.: Explicit Multipath Congestion Control for Data Center Networks, *Proceedings of the ACM 9th Conference on Emerg-*

ing Networking Experiments and Technologies (ACM CoNEXT'13), pp. 73–84 (2013).

[11] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., Mckeown, N. and Shenker, S.: NOX: Towards an Operating System for Networks, *ACM SIGCOMM Computer Communication Review*, Vol. 38, pp. 105–110 (2008).

[12] Erickson, D.: Beacon Home, `https://openflow.stanford.edu/display/Beacon/Home/`.

[13] Cai, Z., Cox, A. L. and Ng, T. S. E.: Maestro: A System for Scalable OpenFlow Control, `http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf`.

[14] Project Floodlight: Floodlight OpenFlow Controller, `http://www.projectfloodlight.org/floodlight/`.

[15] Trema: Full-Stack OpenFlow Framework in Ruby and C, `http://trema.github.com/`.