リンク・ステート型経路制御における トポロジ・ブロードキャスト制御

今泉英明[†] 中村修[††] 村井純[††]

本論文では，リンク属性情報が頻繁に更新されるリンク・ステート型経路制御（LSR）技術においてブロードキャストされる制御トラフィックを制御する手法を提案し，それを適用したトポロジ情報ブロードキャスト・プロトコル *Graph Configurable Topology Broadcast Protocol*（GCTBP）の詳細について述べる．従来の LSR 技術では，特定リンクへの制御トラフィック流入回避や，同一ノード間の多重リンクとなる制御トラフィック専用リンクの利用はできない．本論文では，データトラフィックと制御トラフィックが流れるネットワークを区別し，*Topology Broadcast Reverse Path Forwarding*（TBRPF）を拡張したプロトコル GCTBP を適用することにより，この問題を解決した．同一ノード間に多重リンクを付設できない場合を想定し，2 段階の優先度による特定リンクへの制御トラフィックの流入回避をシミュレーションによって評価した．その結果高優先に設定するリンクに依存し，全体収束速度を変えずにリンク属性情報の変化による制御トラフィックの流入を回避できた．

A Topology Broadcast Control on Link State Routing

Hideaki Imaizumi,[†] Osamu Nakamura[††] and Jun Murai[††]

This paper proposes a technique for controlling a flow of control traffic of Link State Routing (LSR) Technology, and describes detail of a topology broadcast protocol called *Graph Configurable Topology Broadcast Protocol* (GCTBP). Due to the fact that existing LSR technology can not control a flow of control traffic, it is impossible for a network manager to prevent the flow of control traffic on a specific link or to use an exclusive link for control traffic. This paper approaches such problem by using GCTBP, which is an extended protocol of *Topology Broadcast Reverse Path Forwarding* (TBRPF). Simulations on random topologies with two-level priority showed that the technique was effective on link-attribute change event in terms of communication cost and convergence time.

1. はじめに

リンク・ステート型経路制御（Link State Routing, 以後 LSR）技術[1)~3)] は，その可用性の高さからトラフィック・エンジニアリング技術[4),5)] や QoS 経路制御技術[3),6)] 等の新しい技術に応用されている．LSR 技術は対象ネットワークをノードとリンクからなるグラフ情報としてとらえ，重み等の属性情報をノードやリンクに付属させネットワーク全体にブロードキャストし，このグラフ情報を基に経路計算を行う．これらの新しい技術は，比較的更新頻度の少ない情報を属性情報としてブロードキャストし実現される．この機能は一般に**トポロジ情報ブロードキャスト**（以後，ブロードキャスト）と呼ばれる．

一方でルータの出力キュー長が，ある閾値を超える際に，そのイベントをリンクの属性情報としてブロードキャストし，それを基に経路制御を行う技術が研究されている[7)]．この情報は非常に更新頻度が高く，最適化なしのシミュレーションではデータトラフィックの 25%が制御トラフィックとなった[7)]．最適化の結果，この制御トラフィックは 2〜4%程度に減少しているが，実際のネットワーク環境で見られる複雑なトラフィックパターンに対して同様の結果が得られるかは明らかではない．

この例のように，更新頻度の高い情報を用いて経路制御を行う場合は制御トラフィックがデータトラフィックに与える影響が問題となる．特に，キューイング技術により優先制御されるデータトラフィックに対して，元来優先されるべき性質を持つ制御トラフィックが干

† 慶應義塾大学大学院政策・メディア研究科
Graduate School of Media and Governance, Keio University
†† 慶應義塾大学環境情報学部
Faculty of Environmental Information, Keio University

渉する可能性がある．

この問題の解決策として，(1) 問題となる特定の 2 ノード間にデータトラフィック用リンクとは異なる制御トラフィック用リンクを設け両者を完全に分離する，(2) 代替経路が存在する場合は特定のリンク上に制御トラフィックを流さない，という 2 つがあげられる．制御トラフィックの伝達速度という面で (1) が望ましいが，そのような設備がない場合は (2) を使うことも考えられる．

しかし現状の LSR 技術においてこれらは実現できない．一般的に Flooding[8] をブロードキャスト・プロトコルとして利用するが，Flooding では制御トラフィックはすべてのリンク上に流れる．また，別のブロードキャスト・プロトコルとして *Topology Broadcast based on Reverse Path Forwarding*（TBRPF）[9] がある．これは主に無線アドホック・ネットワークを対象として考案され，制御トラフィックを最短ホップから計算された配送木に沿って転送する．両者ともに，データトラフィックと制御トラフィックを流す対象のネットワークが同一であり，(1) のような多重リンクを扱うことはできない．また (2) のように明示的に制御トラフィックを操作することは考えられていない．

本研究では，更新頻度の高い情報をリンク属性情報として用いる新しい技術の支援を目的に，従来の LSR 技術では考えられなかった制御トラフィックのブロードキャストを制御する手法を提案する．そのために，まずリンクの役割を 3 つに分類し，それらに基づいたブロードキャスト制御の手法について説明する．そして TBRPF を拡張し，本手法を適用したトポロジ情報ブロードキャスト・プロトコル *Graph Configurable Topology Broadcast Protocol*（GCTBP）を設計し，その詳細を示す．また，シミュレーションによって本プロトコルの有用性の評価を行う．

本論文に示すプロトコルによって，ネットワーク管理者は制御トラフィックを，ボトルネックとなるリンクや衛星等の高遅延リンクを避けて，新たに設けた制御トラフィック専用リンクに流したり代替経路に流したりすることが可能となる．

2 章では，LSR 技術におけるブロードキャスト制御に関して説明し，LSR 技術を成立させるための条件を示す．3 章では，本手法を適用したプロトコル GCTBP の詳細について示し，4 章でその複雑度の解析，5 章でシミュレーションの結果と評価を示す．以上の議論を基に 6 章で考察を行い，7 章で関連技術をあげ，8 章にまとめと今後の課題を示す．

2. LSR 技術におけるブロードキャスト制御

本章では，LSR 技術においてブロードキャストされる制御トラフィックが流れるネットワークと，データトラフィックが流れるネットワークを別のものと考え，分類・モデル化を行う．そのうえで，制御トラフィックが流れるネットワークにおいて必要な操作について論じる．

2.1 ネットワークの分類

LSR 技術における制御トラフィックとは，本質的に各ノードの現在のリンク状態情報を他のすべてのノードに送信するメッセージの集合からなる．

このメッセージを配信する LSR 技術のブロードキャスト機能は**信頼性のあるブロードキャスト**である．これは**通信ネットワーク上のあるノードが送信したメッセージは，有限時間内に他のすべてのノードにエラーなく同じ順序で到達する**，と定義される[10]．

対象ネットワークを無向グラフ $G = (V, E)$ にモデル化する．E は式 (1) で表され，n は同一ノード間の多重リンクを識別する．このとき，V の要素はルータを示し，E の要素はルータ間を接続するデータリンクを示す．

$$E \subset \{(i, j, n) | i, j \in V, n = 0, 1, \ldots\} \quad (1)$$

このグラフはその目的からブロードキャストグラフ $G_B = (V_B, E_B)$ とルーティンググラフ $G_R = (V_R, E_R)$ に分類できる．これらの関係は，$G = (V_B \cup V_R, E_B \cup E_R)$ となる．

ブロードキャストグラフとは，制御メッセージがブロードキャストされるネットワークを示す．一方ルーティンググラフは，データトラフィックの転送に使われるネットワークを示す．また同時に，ルーティンググラフは制御メッセージによって属性情報とともにグラフ情報として配信される対象である．従来の LSR 技術では $G_R \equiv G_B$ である．

G_R は，多重リンクを扱う経路決定アルゴリズムを適用する場合には多重グラフとなりうる．データトラフィックを転送するノード集合 V_R に属するノードは，経路制御を行ううえで必ず属性情報が付属したグラフ G_R の情報が必要になる．この情報は制御トラフィックとしてブロードキャストされるため，V_R に属するノードは必ずブロードキャストグラフのノード集合 V_B にも属さなければならない．また，V_R に属するすべてのノードへ等価な情報が送信されることを保証するため，G_B の連結成分 $w(G_B)$ は 1 でなくてはならない．

したがって，グラフ G において LSR 技術による経

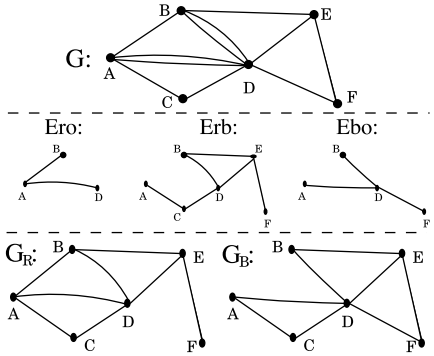

図 1 リンク種別と 2 つのグラフ G_R と G_B
Fig. 1 Relationship between link-type and two graphs.

路制御を成立させるためには式 (2) を満たさなければならない．実際には，G_B の連結成分となる G_Bi のノード集合 $V(G_Bi)$ が V_R を部分集合とすればよいが，本論文では上述のように扱う．

$$V_R \subseteq V_B, w(G_B) = 1 \quad (2)$$

2.2 リンクの利用目的による分類

管理者によって各リンクは，その利用目的により次の 3 つに設定される．

ROLink（**Routing-Only**）：データトラフィック専用
RBLink（**Routing-Broadcast**）：兼用
BOLink（**Broadcast-Only**）：制御トラフィック専用

それぞれのリンクの集合を E_{ro}，E_{rb}，E_{bo} とすると，各グラフとの関係は次のようになる．

$$E = E_{ro} \cup E_{rb} \cup E_{bo} \quad (3)$$
$$G_R = (V_R, E_{ro} \cup E_{rb}) \quad (4)$$
$$G_B = (V_B, E_{bo} \cup E_{sub}) \quad (5)$$
$$E_{sub} = \{(i,j,n) \in E_{rb} | E_{bo}(i,j) = \phi\} \quad (6)$$

式 (6) で用いた $E_{bo}(i,j)$ は，ノード i，j を結ぶ *BOLink* の集合を示す．G_R は，E_{ro} と E_{rb} の和集合をリンク集合 E_R として持つ．一方 E_B は，データトラフィックがリンク帯域を最大限利用できるように，同一ノード間に *RBLink* と *BoLink* が同時に存在している場合は *RBLink* を含まない．

図 1 に全体的な関係図を示す．

一般的に *RBLink* 数 $|E_{rb}|$ が 0 に近いグラフでは，リンク障害あるいはノード障害によってブロードキャストグラフ G_B が分断され，条件式 (2) を満たせなくなる可能性が高い．したがってこのようなグラフで表されるネットワークは，障害に弱いといえる．

2.3 ブロードキャスト制御

前節に示したようにリンクを分類し，ブロードキャストグラフを分離して扱うことによって，制御トラフィック専用のリンクを制御できる．

特定リンクだけを回避してブロードキャストを行うためには，その方針を重みに反映させ，その重みが付けられたグラフから配送木を作る必要がある．

ブロードキャストグラフに属するリンクは *BoLink* あるいは *RBLink* である．*BoLink* は制御トラフィック専用であるため，重みはつねに最小と考えられる．一方で *RBLink* 間ではそれぞれのリンクの重要度やボトルネックの度合いを考慮してネットワーク設計時に重みが決められる．

ブロードキャストの配送木を構築するには，共通のスパニングツリーを用いる手法[11),12)] や *Extended Reverse Path Forwarding*（ERPF）アルゴリズム[13)] が利用できる．前者の場合，無向リンクの重みに対して逐次アルゴリズム *Kruskal*[14)] を基にした分散アルゴリズムを用いて配送木を構築する．すべての制御トラフィックは，この配送木を通る．後者の場合は，各ノードが有向リンクの重みに対して最小コスト木を構成する．この配送木は，送信ノードによって異なる．また後者の場合は有向リンクごとに異なる重みを設定できるため，方向によって遅延の大きさが異なるリンク[15)] 等に対応できる．

3. GCTBP 詳細

Graph Configurable Topology Broadcast Protocol（GCTBP）は，TBRPF を拡張した信頼性のあるトポロジ情報ブロードキャスト・プロトコルであり，LSR 技術に応用可能である．TBRPF 同様 ERPF を用いて配送木を構成し，シーケンス番号を用いて信頼性を確保する．

3.1 対象ネットワークの定義

GCTBP では対象となる通信ネットワークを，有向グラフ $G = (V, E)$ として表現する．このとき V はルータやスイッチ等のノードの集合，E はノード間のリンクの集合を表す．各ノードはネットワーク内で一意な識別子を持つ．各リンクは 1 対 1 の双方向通信路であり，(i,j,n) と (j,i,n) は区別される．

2.2 節で説明したとおり，グラフ G の各リンクは方向に関係なく *ROLink*，*RBLink*，*BOLink* のいずれかとなる．ここで，それぞれのリンクの集合を，E_{ro}，E_{rb}，E_{bo} とする．

2 つのノード u，v 間では，必ず次のいずれかの関係になることを前提とする．

(a) リンクが存在しない．
(b) どれか 1 種類のリンクが 1 つ存在する．
(c) *ROLink* と *RBLink* のいずれか 1 つと，*BOLink*

が 1 つ存在する．

経路制御の対象となるグラフ G_R は式 (4) と同様の操作によって求められる．本論文の主眼はトポロジ情報ブロードキャスト・プロトコルであるため，ルーティンググラフが多重グラフとなる場合は考えない．

一方ブロードキャストを行うグラフ G_B は式 (5), (6) と同様の操作から求める．グラフ G_B はグラフ G に変化があるたびに，新たに式 (5), (6) によって再計算される．

結果として求められる 2 つのグラフ G_R，G_B はそれぞれ多重辺を持たない単純な有向グラフとなり，各リンクは (i,j) と表現される．また同様にリンクの集合 E_{ro}，E_{rb}，E_{bo} も多重辺を持たないため，以後各リンクは (i,j) と表現する．

3.2 通信リンクの特性

GCTBP が対象とするネットワークの通信リンクは，以下を満たすことを前提とする．

(a) リンク (u,v) が使用可能状態（up 状態）であるとき，かつそのときに限り，リンク (v,u) も up 状態である．

(b) あるリンクがあるノードで使用不可能状態（down 状態）になった場合，有限時間内に，そのリンクの接続先のノードでも down 状態になる．

(c) あるリンクを通して送られたメッセージは，有限時間内にエラーなく同じ順序で接続先のノードに届く．

(d) リンクの種類は両端で同じであり，片方向だけ異なることはない．また動的には変わらない．

(e) リンクの種類が *RBLink* である場合は，ブロードキャスト・コスト情報が方向に関係なく付属する．*BOLink* の場合は最小コストが設定され，*ROLink* の場合は何も設定されない．

(f) リンクの種類が *ROLink* または *RBLink*，経路制御用属性情報が方向それぞれに独立して付属する．*BOLink* の場合は何も設定されない．

(b)，(d)，(e) は Hello プロトコル等で保証されると仮定する．また，(c) はデータリンク層等の下位層で実現されるものとする．

3.3 GCTBP に必要な操作

グラフ G_R に属する各リンク (u,v) には，あらかじめ管理者によって設定された経路制御用の重みが設定されている．この重みは $cost(u,v)$ として参照され，前節で述べたとおり $cost(u,v)$ と $cost(v,u)$ は同じとは限らない．

一方で，リンク集合 E_{RB} に属する各リンク (u,v) にはブロードキャスト用の重みが設定されている．この重みは $bcost(u,v)$ と参照される．この重みには方向はなく，つねに $bcost(u,v) = bcost(v,u)$ となる．

グラフ G_R，G_B に属するすべてのリンク (u,v) のリンク状態情報は，(u,v,t,atr,sn) という組でブロードキャストされ更新される．ここで t はリンクの種類を示し，*ROLink*，*RBLink*，*BOLink* のいずれかである．atr は重みを含む属性情報の集合であり，経路制御用の重みは $atr.rc$，ブロードキャスト用の重みは $atr.bc$ と表記する．一方 sn はシーケンス番号を示す．(u,v,t,atr,sn) のリンク状態更新は，ノード u が始点となりブロードキャストを行う．

グラフ G_B に属するノード i は，(1) リンク状態表，(2) 隣接ノードの集合，(3) ブロードキャスト配送に関する情報を保持する．

(1) のリンク状態表は，ノード i が保持するリンク状態の表である．*ROLink* は RLT_i，*RBLink* は $RBLT_i$，*BOLink* は BLT_i にそれぞれ最新のリンク状態が格納される．リンク (u,v) の情報は，$RLT_i(u,v)$ と表記され，$RLT_i(u,v).atr$ や $RLT_i(u,v).sn$ のようにそれぞれ参照される．

(2) は，ノード i が保持する i の隣接ノードの集合であり，BN_i と表記する．

(3) は，ブロードキャスト配送に関連する情報であり，ノード $src(src \neq i)$ に関してそれぞれ保持される．

(a) **親ノード：**ノード i は BLT_i と $RBLT_i$ の情報を基に，i を根とした最小コスト木を構成する．この木において i から src へ到達するパスの第 1 番目のノード（すなわち次ホップノード）は，src からのブロードキャスト・メッセージを i へ届ける親ノードの役割を持つ．この親ノードを $p_i(src)$ と表記する．

(b) **子ノード集合：**ノード i が，あるノード src からのブロードキャスト・メッセージを転送する必要のある子ノードの集合．$children_i(src)$ と表記する．

(c) **シーケンス番号：**ノード i が受信したノード src のリンク状態更新の中で最新のシーケンス番号．$sn_i(src)$ で表記する．本論文中では，TBRPF と同様にシーケンス番号にはタイムスタンプを利用する．

3.4 GCTBP メッセージ

GCTBP は，*NewParent*，*CancelParent*，*Update* の 3 つのメッセージから成り立つ．メッセージの送信ノード，受信ノードをそれぞれ n，m とする．

NewParent：ノード z と $sn_n(z)$ の組を m に送信し，ノード z から送信されたメッセージを自分

に転送するように要求する．メッセージを受信した m は，$children_m(z)$ に n を追加する．また RLT_m，$RBLT_m$，BLT_m に属する，z から送信されてきたリンク状態の中で，$sn_n(z)$ よりも新しいものをすべて n に *Update* メッセージとして送る．

CancelParent：指定するノード z から送信されたメッセージをこれ以上転送しないように要求する．メッセージを受信した m は，$children_m(z)$ から n を削除する．

Update：新しいリンク状態を送信する．受信したノード m は，リンクの種類によって RLT_m，$RBLT_m$，BLT_m の中で対応する表を検査する．表にないか保持しているものよりも新しければ，リンクの種類に対応した表に格納し，$children_m$（リンク状態を広告したノード）に属する隣接ノードに *Update* メッセージとして送信する．

3.5 プロトコル詳細

GCTBP を仮想コードによって表す．

ノード i は初期状態において，隣接ノードとのリンクを 1 つも持たない．したがってノード i の初期状態では，$RLT_i = RBLT_i = BLT_i = BN_i = \phi$ となり，すべてのノード s について $p_i(s) = NULL$，$children_i(s) = \phi$，$sn_i(s) = 0$ となる．

この初期状態から始まり，すべてのノードは隣接ノードとのリンクが確立した時点で *Link_Up* を実行することで，GCTBP が始動する．

```
 1: Link_Up (i, j, t) { /* link(i,j),t=type */
 2:   ts ← current_timestamp;
 3:   if (t = ROLink) {
 4:     RLT_i(i, j).atr.rc ← cost(i, j);
 5:     RLT_i(i, j).sn ← ts;
 6:     atr ← RLT_i(i, j).atr;
 7:   } else if (t = RBLink) {
 8:     RBLT_i(i, j).atr.rc ← cost(i, j);
 9:     RBLT_i(i, j).atr.bc ← bcost(i, j);
10:     RBLT_i(i, j).sn ← ts;
11:     atr ← RBLT_i(i, j).atr;
12:     if (j ∉ BN_i)
13:       BN_i ← BN_i ∪ {j}; Update_Parents(i);
14:   } else if (t = BOLink) {
15:     BLT_i(i, j).atr.bc ← 1; /* constant */
16:     BLT_i(i, j).sn ← ts;
17:     atr ← BLT_i(i, j).atr;
18:     BN_i ← BN_i ∪ {j}; Update_Parents(i);
19:   }
20:   update_msg ← (i, j, t, atr, ts);
21:   for each (node k ∈ children_i(i))
22:     Send update_msg to node k;
23: }
```

```
 1: Link_Down (i, j, t) { /* link(i,j),t=type */
 2:   ts ← current_timestamp;
 3:   atr.rc ← atr.bc ← ∞;
 4:   if (t = ROLink) {
 5:     RLT_i(i, j).atr ← atr; RLT_i(i, j).sn ← ts;
 6:   } else if (t = RBLink) {
 7:     RBLT_i(i, j).atr ← atr; RBLT_i(i, j).sn ← ts;
 8:     if ((i, j, at, sn) ∉ BLT_i ∨ at.bc = ∞)
 9:       BN_i ← BN_i − {j}; Update_Parents(i);
10:   } else if (t = BOLink) {
11:     BLT_i(i, j).atr ← atr; BLT_i(i, j).sn ← ts;
12:     if ((i, j, at, sn) ∉ RBLT_i ∨ at.bc = ∞)
13:       BN_i ← BN_i − {j};
14:     Update_Parents(i);
15:   }
16:   if (j ∉ BN_i)
17:     for each (node src ∈ V(BLT_i ∪ RBLT_i))
18:       children_i(src) ← children_i(src) − {j};
19:   update_msg ← (i, j, t, atr, ts);
20:   for each (node k ∈ children_i(i))
21:     Send update_msg to node k;
22: }
```

```
 1: Link_Change (i, j) { /* link(i,j) */
 2:   ts ← current_timestamp;
 3:   if ((i, j, atr, sn) ∈ RLT_i) {
 4:     RLT_i(i, j).atr.rc ← cost(i, j);
 5:     RLT_i(i, j).sn ← ts;
 6:     atr ← RLT_i(i, j).atr;
 7:     update_msg ← (i, j, ROLink, atr, ts);
 8:   else if ((i, j, atr, sn) ∈ RBLT_i)
 9:     RBLT_i(i, j).atr.rc ← cost(i, j);
10:     RBLT_i(i, j).sn ← ts;
11:     atr ← RBLT_i(i, j).atr;
12:     update_msg ← (i, j, RBLink, atr, ts);
13:   }
14:   for each (node k ∈ children_i(i))
15:     Send update_msg to node k;
16: }
```

これらの *Link_Up*，*Link_Down*，*Link_Change* ルーチンは，それぞれリンク状態が down から up へ，up から down へ，経路制御用属性情報の変更が起きた場合に呼び出される．*Link_Up*，*Link_Down* のみ，引数としてリンクタイプが追加的に渡される．これらのルーチンは共通して，リンク状態情報を作成し，自分が送信ノードとなるときの子ノードへそれを送信している．また *Link_Up*，*Link_Down* では，G_B 内での隣接ノード集合 BN_i を管理し，変更があった場合は *Update_Parent* ルーチンを呼び出して配送木を更新する．*Link_Down* の 17 行目の操作 $V(BLT_i \cup RBLT_i)$ は，2 つのリンク状態表に属するすべてのノードの集合を示す．

```
 1: Process_New_Parent (i, nbr, srclist, snlist) {
 2:   update_list ← φ;
 3:   for each (node src ∈ srclist) {
 4:     children_i(src) ← children_i(src) ∪ {nbr};
 5:     dr ← {(k, l, ROLink, a, s) | (k, l, a, s) ∈ RLT_i s.t.
        k = src ∧ s > snlist(src)};
 6:     drb ← {(k, l, RBLink, a, s) | (k, l, a, s) ∈ RBLT_i s.t.
        k = src ∧ s > snlist(src)};
 7:     db ← {(k, l, BOLink, a, s) | (k, l, a, s) ∈ BLT_i s.t.
        k = src ∧ s > snlist(src)};
 8:     diff ← dr ∪ drb ∪ db;
 9:     update_list ← update_list ∪ diff;
10:   }
11:   Send update_list to nbr;
12: }
```

```
 1: Process_Cancel_Parent (i, nbr, srclist) {
 2:   for each (node src ∈ srclist)
 3:     children_i(src) ← children_i(src) − {nbr};
 4: }
```

```
 1: Process_Update (i, nbr, inmsg) {
 2:   Update_Table(i, nbr, inmsg, updates);
 3:   Update_Parent(i);
 4:   if ((RLT_i ∪ RBLT_i) has been changed and Hop-by-Hop
      routing is applied)
 5:     Compute_New_Routes(i);
 6:   for each (node k ∈ BN_i)
```

```
 7:     outmsg(k) ← φ;
 8:   for each (node src ∈ V(BLT_i ∪ RBLT_i) s.t. src ≠ i)
      {
 9:     upd(src) ← {(k, l, t, a, sn) ∈ updates s.t. k = src};
10:     for each (node k ∈ children_i(src))
11:       outmsg(k) ← outmsg(k) ∪ upd(src);
12:   }
13:   for each (node k ∈ BN_i s.t. outmsg(k) ≠ φ)
14:     Send outmsg(k) to node k;
15: }
```

Process_New_Parent，*Process_Cancel_Parent*，*Process_Update* ルーチンは，対応するメッセージを受信した際に呼び出される．基本的な動作は 3.4 節に示したとおりである．*Process_Update* が 1 行目に呼び出している *Update_Table* は，受け取ったリンク状態の集合から新しいものを *updates* に入れて返すルーチンである．また 4，5 行目では，経路制御用のリンク状態に変化があり，かつ経路制御方式としてホップバイホップ方式が採用されている場合は，経路表を再構築するために *Computer_New_Route* を呼び出している．これらのルーチンは付録に示す．

```
 1: Update_Parent (i) {
 2:   Compute_New_Parent(i);
 3:   for each (node k ∈ BN_i) {
 4:     cancel_list(k) ← φ;
 5:     src_list(k) ← φ;
 6:     sn_list(k) ← φ;
 7:   }
 8:   for each (node src ∈ V(BLT_i ∪ RBLT_i) s.t. src ≠ i)
      {
 9:     if (new_p_i(src) ≠ p_i(src)) {
10:       if (p_i(src) ≠ NULL ∧ p_i(src) ∈ BN_i) {
11:         k ← p_i(src);
12:         cancel_list(k) ← cancel_list(k) ∪ {src};
13:       }
14:       if (new_p_i(src) ≠ NULL) {
15:         k ← new_p_i(src);
16:         src_list(k) ← src_list(k) ∪ {src};
17:         sn_list(k) ← sn_list(k) ∪ {sn_i(src)};
18:       }
19:       p_i(src) ← new_p_i(src);
20:     }
21:   }
22:   for each (node k ∈ BN_i) {
23:     if (src_list(k) ≠ φ)
24:       Send NEW_PARENT (src_list(k), sn_list(k)) to
        node k;
25:     if (cancel_list(k) ≠ φ)
26:       Send CANCEL_PARENT(cancel_list(k)) to node k;
27:   }
28: }
```

```
 1: Compute_New_Parents (i) {
 2:   BG ← {(i, j, a.bc)|(i, j, a, s) ∈ BLT_i, a.bc ≠ ∞}
 3:   BLT_sub ← {(i, j, a, s) | (i, j, a, s) ∈ RBLT_i, a.bc ≠ ∞,
      (i, j, x) ∉ BG}
 4:   BG ← BG ∪ {(i, j, a.bc)|(i, j, a, s) ∈ BLT_sub}
 5:   for each (node src ∈ V(BG) s.t. src ≠ i)
 6:     new_p_i(src) ← NULL;
 7:   /* calculates SPF tree rooted at node i. */
 8:   Dijkstra(i, BG);
 9:   for each (node src ∈ BG s.t. src ≠ i)
10:     new_p_i(src) ← nexthop-node to src;
11: }
```

Update_Parent と *Compute_New_Parents* はサブルーチン的に呼び出され，ブロードキャスト配送木を計算し，必要であれば *NewParent*，*CancelParent* メッセージを隣接ノードへ送信する．

Compute_New_Parents の 2 から 4 行目では，式 (5)，(6) に従って G_B を求め，8 行目で最小コストパス木を計算し，9，10 行目で各送信ノード *src* に対する新しい親ノードを $new\_p_i(src)$ に記録する．

表 1 各プロトコルのメッセージ複雑度
Table 1 Message complexity of each protocol.

イベント	Flooding	TBRPF	GCTBP
リンク属性変化	$O(\|E\|)$	$O(\|V\|)$	$O(\|V\|)$
リンク障害	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\|^2)$
リンク復旧 A	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\|^2)$
リンク復旧 B	$O(\|E\|^2)$	$O(\|E\|\|V\|)$	$O(\|E\|\|V\|)$

4. 複雑度の解析

通信複雑度（Communication Complexity）と時間複雑度（Time Complexity）に関して，GCTBP は TBRPF とほぼ同じである．それぞれ異なる点に関して以降補足する．

4.1 通信複雑度

各プロトコルの通信複雑度として，各イベント時に収束するまでに必要なメッセージ数の最悪値を**表 1** に示す．リンク復旧 A とリンク復旧 B との違いは連結性であり，リンク復旧 A はすでに連結しているグラフ内でさらにリンクが復旧することを，リンク復旧 B は分断してしまったグラフを連結させるリンクが復旧したことを意味する．

リンク障害とリンク復旧 A に関して，TBRPF/GCTBP は Flooding と比較して必要メッセージ数のオーダが高く見えるが，平均のメッセージ数は少ない．

TBRPF と GCTBP は最悪値は同じであるが，木の構成が異なるため *RBLink* のブロードキャスト用の重み次第でメッセージ数が変わる．基本的には，TBRPF はホップ数を基に木を構成するため，新たなリンクが増えても *NewParent* と *CancelParent* のメッセージが発生しにくい．一方 GCTBP では，ブロードキャスト用の重み次第で，隣接ノード間で親子関係にあったものがキャンセルされ，複数ホップの経路が選択されるため，新たなリンクが増えた場合のメッセージ数は多くなる．しかし，リンク障害時は TBRPF の場合必ず木の構成が影響されるが，GCTBP の場合は木に含まれないリンクが障害を起こした場合は影響されない．したがってリンク障害時では，GCTBP の方が TBRPF よりもメッセージ数が少なくなる可能性がある．

4.2 時間複雑度

各プロトコルの時間複雑度として，各イベント時に

表 2 各プロトコルの時間複雑度
Table 2 Time complexity of each protocol.

イベント	Flooding	TBRPF	GCTBP
リンク属性変化	$O(D)$	$O(D)$	$O(Dpt)$
リンク障害			
リンク復旧 A,B			

収束するまでに必要な時間の最悪値を**表 2** に示す．ここで，D はグラフの直径を示し，Dpt はブロードキャスト配送木の最大の深さを示す．

Flooding/TBRPF と GCTBP で大きく異なることは，前者がグラフの直径に依存して収束時間が変化するのに対し，GCTBP は配送木の深さに依存して変化することである．これは収束時間に大きく影響を与えるためネットワーク設計時に熟考が必要となる．

5. 評　　価

本章では GCTBP の評価をシミュレーション実験によって行う．

1 章で説明したとおり，最も情報伝達が速い方法は既設のリンク以外に制御トラフィック専用リンクを設けて利用することである．しかし，そのような設備がなく各リンクをデータ・制御トラフィック兼用で使わざるをえない場合，GCTBP を用いた特定リンクへの制御トラフィックの流入回避がどの程度有効かを評価する．

まず GCTBP の特性を調べるために他のプロトコルとともにシミュレーションを行い結果を比較する．対象となるプロトコルは Flooding[8]，最適化された Flooding，TBRPF である．Flooding と最適化された Flooding の相違は，リンクが確立した際の処理である．Flooding はリンク状態表をすべて送信するが，最適化された Flooding は両者の持つノード識別子とシーケンス番号のすべての組を交換してから差分を相互に送信する．

その後，ブロードキャスト用コストとして二値的な設定を行った GCTBP の性能についてシミュレーションによって評価を行う．

シミュレーションでは様々なイベントが発生した後から収束するまでに要した制御トラフィック量と時間を計測する．ここで収束とは，「すべてのノードが同じリンク状態情報の集合を保持し，制御トラフィックがこれ以上発生しない状態」を指す．

発生するイベントとして，(1) 初期収束，(2) リンク属性変化，(3) リンク障害，(4) リンク復旧，(5) ノード障害，(6) ノード復旧がある．(1) はすべてのリンクが down 状態から up 状態へ移行したことを指す．(2) は全リンクの属性情報が同時に変化したことを指す．GCTBP では経路制御用コストが変更されたことを意味するが，キューの長さを属性情報として利用する場合はその変化も含む．(3)(4)，(5)(6) はリンク・ノードが down 状態から up 状態へ，あるいは up 状態から down 状態へ移行したことを指す．ノードが up 状態から down 状態となる場合はそのノードに接続しているすべてのリンクが down 状態へ移行することを意味する．逆にノードが up 状態へ移行する場合はノードに接続するすべてのリンクが up 状態へ移行することを意味する．

各識別子のビット長は，ノード識別子を 32 ビット，経路制御用とブロードキャスト用コストを 16 ビット，リンクタイプを 2 ビット，シーケンス番号を 32 ビットとした．ただしブロードキャスト用コストとリンクタイプは GCTBP でしか使われない．これらはプロトコルオーバヘッドとなり，*Update* メッセージに含まれる各リンク状態情報を他のプロトコルのリンク状態情報よりも約 16%増加させる．

また，対象グラフの各リンクは *RBLink* として扱い，リンク遅延はすべて 1 ms とした．

5.1 ランダムグラフでの比較

ランダムに生成された 20 ノードのグラフに対して各プロトコルをシミュレートし，比較した．ランダムグラフは，BRITE[16] を利用し様々なリンク数のグラフを合計 34 個生成した．生成アルゴリズムとしては Barabasi[17] と Waxman[18]（*parameters*: $\alpha = 0.15$，$\beta = 0.2$）を用いた．各リンクは *RBLink* としブロードキャスト用コストは 1 から 50 の間のランダムな値とした．Waxman のグラフに関しては，Barabasi と同様の傾向が見られたため，Waxman の結果は省略する．

まず，**図 2** に，初期収束するまでネットワーク全体に流れた総トラフィック量（左）と時間（右）を示す．最もトラフィックが多いのは，最適化された Flooding であった．これはリンク確立時のシーケンス番号交換を行った際，共通して保持しているリンク状態情報がないためにオーバヘッドになったと考えられる．総トラフィック量では TBRPF が最も少なかった．総トラフィック量では，最適化された Flooding と比較し，TBRPF が最大で 95.3%（Link 数=190）少なかったのに対して GCTBP は約 89.4%（Link 数=190）少なかった．一方収束時間に関しては，最も短かったのは Flooding であり，GCTBP は最長で Flooding の 3.3 倍（Link 数=85，184）であった．TBRPF は最長で Flooding の 1.5 倍（Link 数=37，54，70）であった．

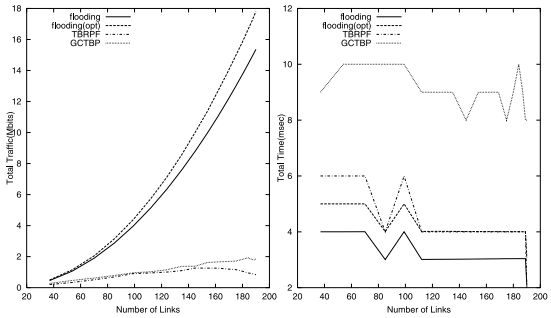

図 2 初期収束までの総トラフィック量（左）・時間（右）
Fig. 2 Comparative evaluation of total traffic/time on initial convergence.

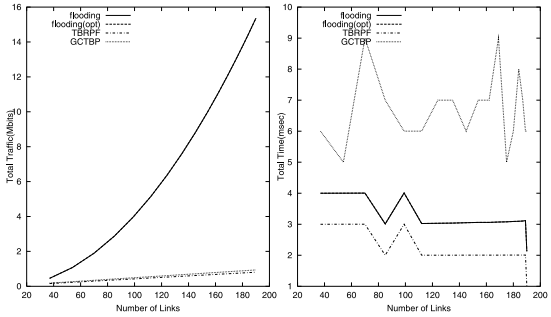

図 3 リンク属性変化イベントでの総トラフィック量（左）・最大時間（右）
Fig. 3 Comparative evaluation of total traffic/time on link-attribute change.

次に**図 3** に，すべてのリンク属性情報を変えた後に収束するまでにネットワーク全体に流れた総トラフィック量と最大時間を示す．トラフィック量は Flooding 系は $O(|E|)$ に従って増加しているが，TBRPF/GCTBP はそれほどの増加は見られない．重要な点は，この収束時間は TBRPF にとってはグラフの直径 D となり，GCTBP にとっては配送木の最大の深さ Dpt となることである．Flooding 系は，収束するために $D+1$ 単位時間必要であることが分かる．

一方，個々のリンク，ノードに対するイベントにおいて，そのイベントが発生してから収束するまでに必要なトラフィック量と時間のイベントごとの平均や標準偏差を求めるために，次のようにシミュレーションを行った．リンク障害イベントでは，すべてのリンクが up 状態で収束している状態から，リンクを 1 つ down 状態へ移行し収束するまでを計測，再度そのリンクを up 状態へ移行させ収束させてから，次のリンクを同様に計測し，平均発生トラフィック量や平均収束時間を求めた．逆にリンク復旧イベントでは，あるリンク 1 つだけが down 状態で収束している状態から，そのリンクを up 状態へ移行させ収束するまでを

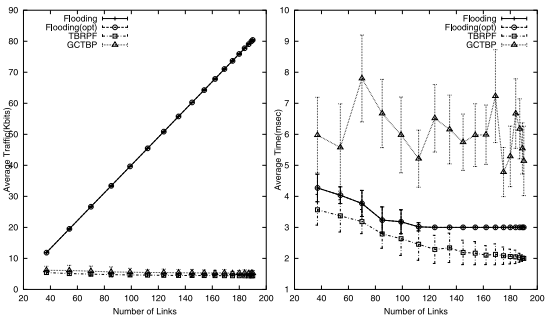

図 4 リンク障害から収束までの平均トラフィック量（左）・時間（右）の比較
Fig. 4 Comparative evaluation of average traffic/time on link failure.

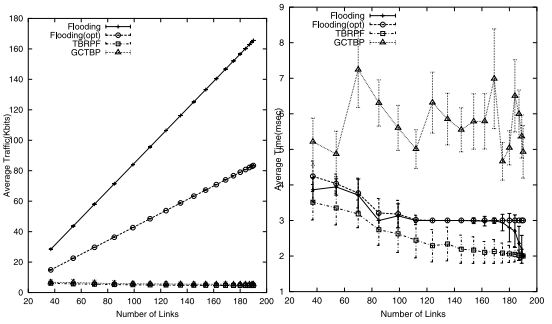

図 5 リンク復旧から収束までの平均トラフィック量（左）・時間（右）の比較
Fig. 5 Comparative evaluation of average traffic/time on link recovery.

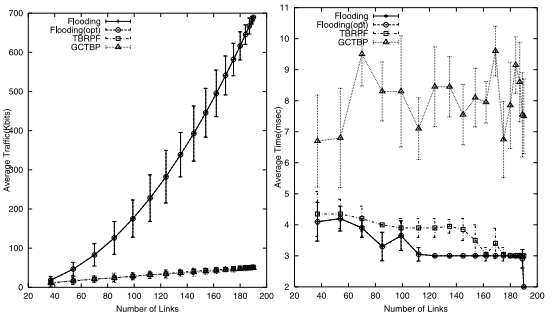

図 6 ノード障害から収束までの平均トラフィック量（左）・時間（右）の比較
Fig. 6 Comparative evaluation of average traffic/time on node failure.

計測し，すべてのリンクに対して同様に計測を行った．ノード障害，ノード復旧では，各ノードに対して上記と同様の方法を用いてそれぞれのノードに対して計測を行った．

図 4，**図 5**，**図 6**，**図 7** にそれぞれ，各イベントごとの測定結果を示す．

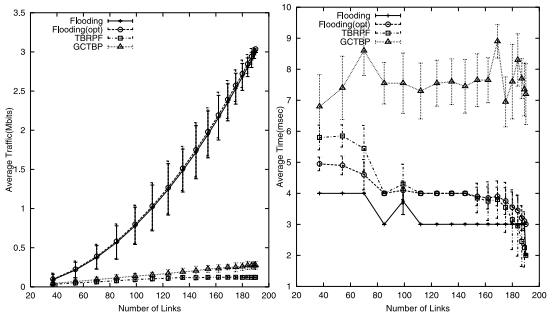

図 7 ノード復旧から収束までの平均トラフィック量（左）・時間（右）の比較
Fig. 7 Comparative evaluation of average traffic/time on node recovery.

平均トラフィック量に関してはすべてのイベントにわたって，Flooding 系は非常に多く，TBRPF/GCTBP はそれに比べると大幅に少なかった．TBRPF と GCTBP の差は，Flooding との差と比べ非常に小さい．リンク復旧に関しては，Flooding の最適化が働いていることが分かる．ノード障害において，GCTBP の方がプロトコルオーバヘッドがあるにもかかわらず，多くの場合 TBRPF よりも少なかった．GCTBP は TBRPF よりも，最大で 15.6%（Link 数=134）減少させ，少なかった例すべてでは平均約 5.8%減少させた．TBRPF の方が GCTBP より少なかったは 34 例中リンク数 85 以下の 9 例であり，比較的疎なネットワークであった．9 例中 8 例は TBRPF に対して 4%未満の増加，最大は 8.3%の増加であり，すべての例がプロトコルオーバヘッド 16%未満であった．

収束時間に関しては，どの結果も図 3 の結果に影響されている．すなわち直径，あるいは配送木の最大の深さの影響が大きい．GCTBP は，他のプロトコルに比べてばらつきが大きいが，これは特定のリンク・ノードによって必要とされる収束時間が大きく異なるためと考えられる．

5.2 配送木の構成変更の頻度

前節でのシミュレーションにおいて，各イベント発生から収束までに TBRPF/GCTBP が必要とした配送木の再構成の頻度について評価を行う．

評価方法は，前節での初期収束，リンク障害・復旧，ノード障害・復旧の各イベントのシミュレーションにおいて，「配送木の親ノードが up 状態にもかかわらず，親ノードを別のノードに変更した回数」の合計回数を計測した．その結果を**図 8** に示す．

初期収束では GCTBP は単調に増加しているが，TBRPF はリンク数 100 付近までは増加し，160 くらいから下降している．リンク数が増加し密なネットワークになると，TBRPF ではほとんどのノードと初めから隣接関係になり配送木が固定されるためである．一方 GCTBP はあるノード間が隣接関係であってもブロードキャスト用コストによっては配送木の最小コストパスは異なるため，ネットワークが密になっても配送木の変化は依然として発生する．

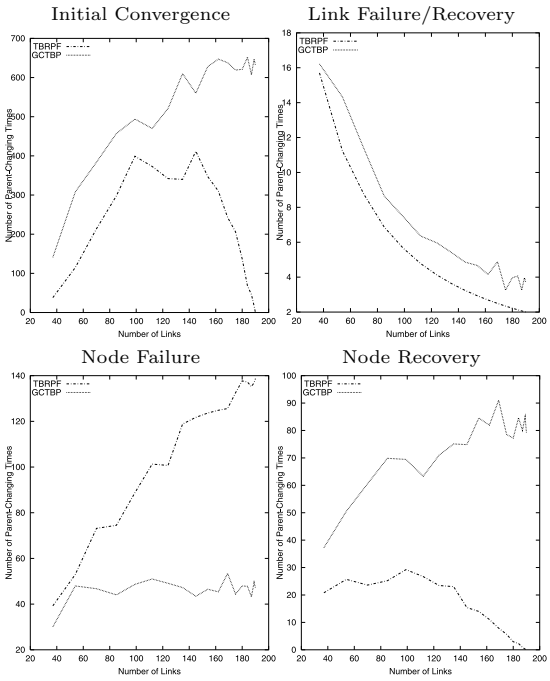

図 8 各イベントにおける親ノード変更の総回数
Fig. 8 Total number of parent-changing times on each event.

リンク障害・復旧は，前節で説明したとおりの手順で計測したため同一の配送木の変化が発生し，まったく同じ結果となった．全リンクでの合計であるため，リンクの平均にした場合は両者の差は非常に小さい．

ノード障害とノード復旧では対照的な結果となった．ノード障害では，GCTBP はリンク数が増加しても配送木の変化が大きく変わらないのに対して，TBRPF は単調に増加する．これは TBRPF は最短ホップ数で最小コスト木を作るのに対して，ランダムコストの設定の GCTBP では最小コストパスとして使われるリンクに偏りがあるためである．同様の理由でノード復旧の際は，まったく逆の結果となっている．これは TBRPF の場合，同一ノード間に同じホップ数で到達する複数パスがある可能性が高く，そのために配送木を変更する必要がないことが多いためと考えられる．一方で GCTBP の場合，不要な配送木変化を経てノード障害前の状態へ戻っていることが分かる．

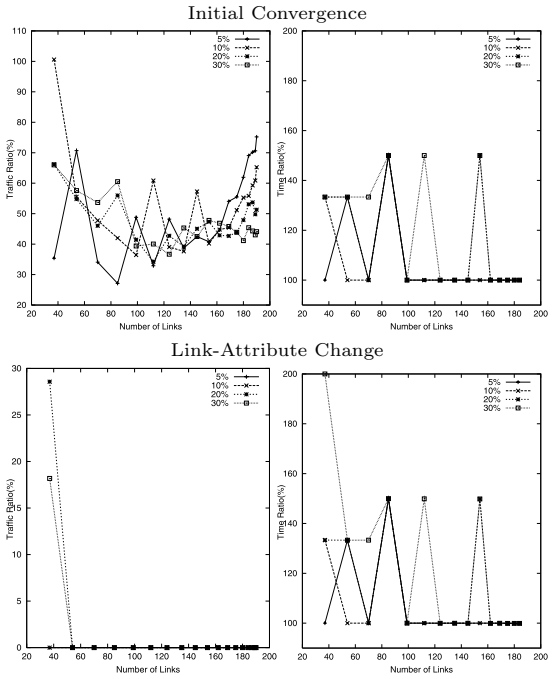

図 9 二値的な設定の際の平均トラフィック量（左）・時間（右）の割合
Fig. 9 Traffic/time ratio of two-criteria control.

5.3 二値的コスト設定の性能

GCTBP の具体的な適用手法の 1 つとして，2 段階の優先度をブロードキャスト用コストに反映した設定を行い，その性能を評価する．

ブロードキャスト用コストとして 2 段階の優先度を用意して，前節までに利用したランダムグラフの全リンク数のうち，ランダムに選出した 5，10，20，30%のリンクを高優先度に，他のリンクは常に低優先度とした．低優先度の場合はブロードキャスト用コストを 1 とし，高優先度の場合は 1000 とした．

この設定で，初期収束時に高優先度リンクを流れた平均トラフィック量と収束時間，リンク属性変化イベント時の高優先度リンクを流れた平均トラフィック量と収束時間を測定した．またすべてのリンクを低優先にした場合に関して各リンクに流れた平均トラフィックと収束時間を測定した．この場合の収束時間は TBRPF と等価である．

図 9 に，すべてのリンクを低優先度にした場合の平均トラフィックと収束時間に対する，高優先度の設定を行ったリンクと収束時間の結果を割合で示す．

初期収束の場合，高優先リンクではトラフィック量は設定を行わなかった場合よりも減少している．収束時間に関しては 1.5 倍になる場合があった．

一方リンク属性変化時は，リンク数が 37 のネットワーク上で 20，30%の高優先度の設定を行ったときのみ制御トラフィックが流れたが，それ以外の場合ではすべて回避できた．この時属性情報がすべてのノードに到達するために要した時間は，高優先度の設定を行わなかった場合の 1 から 2 倍程度であった．低優先度の場合と収束時間が変化しなかった例は，高優先度の設定を行っても配送木の深さが変わらなかったためである．

6. 考　　察

本論文では，更新頻度の高い情報をリンク属性情報として用いる新しい LSR 技術を前提としている．したがって最も回避するべき制御トラフィックはリンク属性情報の変化によって生成されるものとなる．

前章の二値的なブロードキャスト用コスト設定の性能結果に示されたとおり，GCTBP によるブロードキャスト用コストを変更することによって特定リンクへのリンク属性変化イベントによる制御トラフィックの流入を回避できた．

しかし，問題として収束速度があげられる．これに関しては図 9 に見られるように，配送木の深さ Dpt がネットワークの直径 D よりも大きくならないようにすることで，収束時間を変化させずに特定リンクへの制御トラフィックの流入回避を実現できる．これを実現するには適切にブロードキャスト用コストを設定する必要があるが，この計算はネットワークが大きくなるにつれ困難となるため，何らかのツールを用いてシミュレーションを行う必要があると考えられる．コスト変更の対象としているリンクによってはつねに Dpt が D よりも大きくなるが，この場合は新しいブロードキャスト用リンクを当該ノード間に用意することによって収束速度の増加は回避できる．

GCTBP のその他の特性として，各イベントにおいて全体トラフィック量はフラッディングに比べ非常に少ないが，収束時間が他と比べ長く標準偏差も大きい．特に初期収束・ノード復旧イベントではブロードキャスト用コストの設定次第では，図 8 に示されるとおり配送木の再構成が非常に多くなる可能性があり，収束に時間がかかる．GCTBP の各イベントにおける最小の収束速度はすべてのブロードキャスト用コストを同一にしたときであり，それは TBRPF の各イベントにおける収束速度と等しい．TBRPF の収束速度は，ノード障害・ノード復旧を除いて Flooding の性能よりも高い．したがって上述のような注意深い設定によって，リンク属性変化以外のイベントにおいても

収束速度のオーバヘッドを小さくできる．

またGCTBPの付加的な性質として，ネットワーク上に他のノードと比べ著しく性能が劣るノードがある等の場合そのノードのリンクのブロードキャスト用コストを大きくすることで，そのノードに制御トラフィックを転送させることを回避し，さらにそのノードに障害が発生した場合に配送木の再構成が少なくなりネットワークへの影響が少なくできると考えられる．

7. 関連研究

関連研究として他のトポロジ・ブロードキャストプロトコルと，トラフィックを削減するために考えられたいくつかのLSRプロトコルをあげる．

SPTA（Shortest Path Topology Algorithm）[19]は，トポロジ・ブロードキャストプロトコルの1つであり，シーケンス番号等の付属的な情報や定期的な再送なしに信頼性を保証する．制御トラフィックの発生パターンはFloodingと等しい．これに対しGCTBPは，制御トラフィックの流れを制御することを目的としたプロトコルである．

また，制御トラフィックを抑えるLSRプロトコルとしてLVA（Link Vector Algorithm）[20]，ALP（Adaptive Link-state Protocol）[21]，OLSR（Optimized Link State Routing）[22]をあげる．これらは部分グラフ伝搬型であり，必要最小限の情報だけを隣接ノードに対して送信する．これに対しGCTBPは，全グラフ伝搬型であり，冗長な情報を伝搬する．これによって，トラフィックエンジニアリング技術やQoS経路制御技術等，新しい技術に応用可能となる．

8. おわりに

本論文では，リンク属性情報として更新頻度の高い情報を用いる新しいLSR技術を対象として，ブロードキャストされる制御トラフィックを操作する手法として，(1) 特定の2ノード間にデータトラフィック用リンクとは異なる制御トラフィック用リンクを設け両者を完全に分離する，(2) 代替経路が存在する場合は特定のリンク上に制御トラフィックを流さない，という2つをあげた．また，これらを実現するトポロジ情報ブロードキャスト・プロトコルGCTBPを提案した．

制御トラフィックの伝達速度という面で (1) が望ましいが，そのような設備がない場合は (2) を使うことも考えられる．その際のGCTBPの特性，有効性を評価するためにシミュレーション測定を行った．

ランダムなブロードキャスト用コスト設定のシミュレーションの結果，様々なイベントにおいても収束時間が配送木の深さに大きく影響を受けることが分かった．

また，2段階の優先度を反映したGCTBPのシミュレーションの結果，最も制御トラフィックを発生させるリンク属性変化イベントにおいて，高優先度のリンクに制御トラフィックの流入を回避できた．また，高優先度に設定するリンクによっては収束速度に影響なしに制御トラフィックを操作できることが分かった．

謝辞 有益な助言を下さった慶應義塾大学環境情報学部の南政樹氏に感謝する．また，本質的な問題点を指摘していただいた査読者の方々に感謝する．

参考文献

1) Moy, J.: OSPF version 2, RFC 2328, Internet Engineering Task Force (Apr. 1998).
2) ISO: Intermediate system to intermediate system intra-domain routeing exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (iso 8473). Technical Report 10589, ISO (1990).
3) ATM Forum: ATM private network-network interface specification v1.0. Technical report, ATM Forum (1996).
4) Katz, D., Kompella, K. and Yeung, D.: Traffic engineering (TE) extensions to OSPF version 2, RFC 3630, Internet Engineering Task Force (Sep. 2003).
5) Li, T. and Smit, H.: Is-is extensions for traffic engineering, RFC 3784, Internet Engineering Task Force (June 2004).
6) Apostolopoulos, G., Kama, S., Williams, D., Guerin, R., Orda, A. and Przygienda, T.: QoS routing mechanisms and OSPF extensions, RFC 2676, Internet Engineering Task Force (Aug. 1999).
7) Basu, A., Lin, A. and Ramanathan, S.: Routing using potentials: A dayanamic traffic-aware routing algorithm, *ACM SIGCOMM'03* (2003).
8) Vishkin, U.: An efficient distributed orientation algorithm, *IEEE Trans. Info. Theory* (1983).
9) Bellur, B. and Ogier, R.G.: A reliable, efficient topology broadcast protocol for dynamic networks, *Proc. IEEE Infocom'99* (Mar. 1999).
10) Awerbuch, B. and Even, S.: A reliable broadcast protocol in unreliable networks, *Networks* (1986).
11) Gallager, R.G., Humblet, P.A. and Spira, P.M.: A distributed algorithm for minimum weight spanning trees, *ACM Trans. Program. Lang. Syst.* (1983).
12) Cheng, C., Cimet, I. and Kumar, S.: A protocol to maintain a minimum spanning tree in a

dynamic topology, *ACM SIGCOMM Computer Communication Review* (1988).
13) Dalal, Y. and Metclafe, R.: Reverse path forwarding of broacast packets, *Comm. ACM* (1978).
14) Kruskal, J.B.: On the shortest spanning subtree on a graph and the traveling salesman problem, *Proc. American Maht. Society 2*, pp.48–50 (1956).
15) Duros, E., Dabbous, W., Izumiyama, H., Fujii, N. and Zhang, Y.: A link-layer tunneling mechanism for unidirectional links, RFC 3077, Internet Engineering Task Force (Mar. 2001).
16) Medina, A., Lakhina, A., Matta, I. and Byers, J.: BRITE: An approach to universal topology generation, *Proc. International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (2001).
17) Barabási, A.L. and Albert, R.: Emergence of scaling in random networks, *Science* (1999).
18) Waxman, B.: Routing of multipoint connections, *IEEE Journal on Selected Areas in Communications* (1988).
19) Spinelli, J. and Gallager, R.G.: Event driven topology broadcast without sequence number, *IEEE Trans. Comm.* (1989).
20) Garcia-Luna-Aceves, J.J. and Behrens, J.: Distributed, scalable routing based on vectors of link states, *IEEE Journal on Selected Areas in Communications* (1995).
21) Garcia-Luna-Aceves, J.J. and Spahn, M.: Scalable link-state internet routing, *Proc. IEEE ICNP'98* (Mar. 1998).
22) Clausen, T., Hansen, G., Christensen, L. and Behrmann, G.: The optimized link state routing protocol, evaluation through experiments and simulation, *IEEE Symposium on Wireless Personal Mobile Communications* (2001).

付　　録

Compute_New_Route を呼び出すルーチン

```
1: Update_Table (i, nbr, in, out) {
2:   out ← φ;
3:   for each ((u, v, t, atr, sn) ∈ in)  {
4:     if (p_i(u) ≠ nbr)  {
5:       ignore; /* nbr isn't parent */
6:     } else {
7:       if (t = ROLink)
8:         LT ← RLT_i;
9:       else if (t = RBLink)
10:        LT ← RBLT_i;
11:      else
12:        LT ← BLT_i;
13:      if ((u, v, a, s) ∈ LT ∧ sn > s)  {
14:        out ← out ∪ (u, v, t, atr, sn);
15:        LT(u, v).atr ← atr;
16:        LT(u, v).sn ← sn;
17:        if (sn > sn_i(u))
18:          sn_i(u) ← sn;
19:      } else if ((u, v, a, s) ∉ LT)  {
20:        LT ← LT ∪ (u, v, atr, sn);
21:        out ← out ∪ (u, v, t, atr, sn);
22:        if (sn > sn_i(u))
23:          sn_i(u) ← sn;
24:      }
25:    }
26:  }
27: }
```

```
1: Compute_New_Routes (i) {
2:   RG ← {(i, j, a.rc)| (i, j, a, s) ∈ (RLT_i ∪ RBLT_i), a.rc
     ≠ ∞}
3:   /* calculates SPF tree rooted at node i. */
4:   Dijkstra(i, RG);
5:   Update Routing Table;
6: }
```

(平成 16 年 7 月 5 日受付)
(平成 17 年 2 月 1 日採録)

今泉　英明（学生会員）
修士（政策・メディア）．1976 年 8 月 13 日生．1999 年慶應義塾大学総合政策学部卒業．2001 年同大学大学院政策・メディア研究科修士課程修了．同大学院政策・メディア研究科博士課程在籍中．2005 年 3 月博士（政策・メディア）取得見込．

中村　　修（正会員）
博士（工学）．1959 年 12 月 1 日生．1982 年慶應義塾大学工学部数理工学科卒業．1984 年同大学大学院理工学部研究科修士課程数理工学専攻修了．1993 年 1 月博士（工学）．1993 年 9 月より同大学環境情報学部助手として就任．2000 年 4 月より同大学環境情報学部助教授．

村井　　純（正会員）
博士（工学）．1955 年 3 月 29 日生．1979 年慶應義塾大学工学部数理工学科卒業．1981 年同大学大学院理工学研究科修士課程数理工学専攻修了．1987 年 1 月博士（工学）．同大学環境情報学部教授．