The University of Osaka
Institutional Knowledge Archive

Title	ネットワークコーディングを用いたゴシップスタイルブロードキャスト
Author(s)	徳山, 瞬; 土屋, 達弘; 菊野, 亨
Citation	平成23年度 情報処理学会関西支部 支部大会 講演論文集. 2011, 2011
Version Type	VoR
URL	https://hdl.handle.net/11094/50235
rights	ここに掲載した著作物の利用に関する注意 本著作物の著作権は情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

ネットワークコーディングを用いたゴシップスタイルブロードキャスト
Gossip-Style Broadcast Using Network Coding

徳山 瞬* 土屋 達弘* 菊野 亨*
Shun Tokuyama Tatsuhiro Tsuchiya Tohru Kikuno

1 まえがき

近年, 大規模な分散システム中で効率的に情報をブロードキャストする手法として, ゴシッププロトコルが注目を集めている [1, 2, 3]. ゴシッププロトコルではメッセージを送信する際に, 隣接ノード全てにメッセージを送信するのではなく, ランダムに選んだいくつかのノードにのみメッセージを送信する. これを繰り返すことでメッセージはシステム全体に広まっていく. ゴシッププロトコルは余剰なメッセージを削減しかつ高いスケーラビリティと信頼性を持ったブロードキャストを行うことが出来る. そのため複製データベース [4, 5], 故障検知 [6, 7], 分散情報管理 [8, 9], ライブストリーミング [10] 等のアプリケーションで利用されている.

ゴシッププロトコルは, メッセージが全てのノードに正しく伝わることを保証するものではないので, その性能評価はノードがメッセージを受け取る確率で行われる. メッセージを受信する確率とトラフィックはトレードオフの関係にあり, 一つのノードがメッセージを送信する隣接ノードの数を増やすことで, システム中のノードがメッセージを受信する確率は上がるが, トラフィックも同時に増加してしまう.

本論文ではネットワークコーディング [11] を利用した新たなゴシッププロトコルを提案する. 提案手法は, 従来のゴシッププロトコルと比較して, 少ないトラフィックで高いメッセージ到達率 (信頼性) を実現する. ネットワークコーディングとは, 送信ノードや受信ノードだけでなく, 中継ノードでもメッセージの符号化, 復号化を行うような通信手法を意味する. 提案手法ではブロードキャストメッセージはそのまま転送されるのではなく, 幾つかの固まりに分割され, そのランダム結合がメッセージとして送信される. 各ノードは複数のメッセージを受信し, それらを復号することでオリジナルのメッセージを得ることができる.

ネットワークコーディングを利用したゴシッププロトコルの研究は既になされているが, 本研究とは異なる環境を想定している. 例えば, [12] では各々のノードが別のもう一つのノードと同期式のラウンドを使い通信を行う. 一方, 提案手法では, 実際の多くのアプリケーションで見られるように, 一つのノードが幾つかのノードとユニキャストを使って通信する. また [12] では情報の伝達速度について着目しており, 信頼性の問題は考慮されていない. 対して本研究では, シミュレーションを行い故障ノードが存在する時のコミュニケーションコストと信頼性について評価している. 文献 [13] では [12] と同様の環境を仮定しており, 故障ノードに関しても考慮されていない.

本論文の構成は以下の通りである. 2 節では従来のゴシッププロトコルについて説明する. 3 節ではネットワークコーディングを利用したゴシッププロトコルの説明を行う. 4 節では最適化と, それを用いた新たなアルゴリズムを説明する. 5 節ではシミュレーションの結果を示す. 6 節では本論文の結論を述べる.

2 従来手法

本節では従来のゴシッププロトコルについて説明する. 図 1 は従来のゴシッププロトコルのアルゴリズムを示している. ノードがメッセージの送信を開始する際には, ランダムに f 個の隣接ノードを選び送信する. メッセージを受け取った際, それが初めてであれば開始時と同様にランダムに f 個の隣接ノードを選びそのメッセージを転送する. ここで, f はファンアウトと呼ばれる.

この手法ではランダムにメッセージを送信するノードを選ぶため, 多くのノードが同じメッセージを受け取る一

*大阪大学, Osaka University

```
Initiation of broadcast of M:
  Send M to f randomly chosen nodes;

When a node receives a message M:
  If (M is received for the first time)
    Send M to f uniformly randomly chosen nodes;
```

図 1: 従来のゴシッププロトコル

表 1: 正常なノードが同じメッセージを受け取った回数 ($n = 1000$, 故障率：10%)

	0	1	2	3	4	≥ 5
$f = 4$	2.9%	10.9%	18.5%	21.6 %	18.8 %	27.3%
$f = 5$	1.2%	5.0%	11.7%	17.0%	19.2%	45.6%
$f = 6$	0.6%	2.5%	6.6%	11.8%	16.2%	62.3%
$f = 7$	0.3%	1.1%	3.6%	7.6%	12.1%	75.1%

方, メッセージが一度も届かないノードも存在する. 表 1 はノード数 $n = 1000$, 故障率 10% の環境下での単純なシミュレーションの結果である. この表は f の値を変化させた時に一つのノードが何度同じメッセージを受け取ったかを示している. 各々の値はシミュレーションを 100 回行った平均値である.

この表から $f = 4$ で到達率が 97% の時に 30% 近いノードが 5 回以上同じメッセージを受け取っていることが分かる. また $f = 7$ で到達率が 99.7% の時には 75% ものノードが 5 回以上同じメッセージを受け取っている.

3 基本提案手法

本節ではネットワークコーディングを利用したゴシッププロトコルについて説明する. これによって, 同じブロードキャストメッセージを何度も受け取ることを減らすことが出来る. 図 2 にアルゴリズムを示す.

3.1 ブロードキャスト開始時

ブロードキャストを開始するノードは, ブロードキャストメッセージを k 個のブロックに分割する. ここで $F_1, F_2, \cdots, F_k$ が分割されたブロックであるとする. 元のメッセージを l ビットであるとし, $b = \lceil l/k \rceil$ とする. 分割されたブロックは全て, 大きさ q のガロア体 $GF(q)$ 上の $\lceil b/\log_2(q) \rceil$ 次のベクトルである.

```
Initiation of broadcast of M:
  Divide M into k fragments F_1, ··· , F_k;
  Choose a set RN of f_init random nodes;
  For p ∈ RN
    Create a message msg from F_1, ··· , F_k
      with random linear encoding;
    Send msg to node p;

When a node receives a message m:
  { Step 1 }
  If (msg is informative)
    Add msg to Received; { Received is initially empty. }
    { Step 2 }
    Choose a set RN of f random nodes;
    For p ∈ RN
      Create a message msg from the messages in Received
        with random linear encoding;
      Send msg to node p;
    { Step 3 }
      If (|Received| = k)
        Decode the broadcast message from Received;
```

図 2: ネットワークコーディングを利用したゴシッププロトコル

次に, 開始ノードはそれらのブロックからランダム線形コーディングを使って送信するメッセージを作り出す. 各々のブロックに対し, $GF(q)$ 上からランダムにある数を選んで係数とし, それらの線形結合を作る. それぞれのブロックの係数の集合とこの線形結合を併せて, 送信メッセージとする. ここで係数は 0 を含まないものとする. なお, 全てのブロックは $GF(q)$ 上のベクトルであり, 加算や乗算は $GF(q)$ 上の演算とする.

メッセージを作成する度, 開始ノードはそれを f_{init} 個の隣接ノードに対して送信する. 初期ファンアウト f_{init} は通常のファンアウト f よりも大きな値とすべきである. これは開始ノードとその他のノードの負荷を等しくするためである. その他のノードは多くて $k * f$ のメッセージを送信するので, f_{init} を $k * f$ に設定する.

また, それぞれのメッセージはヘッダに ID を持っているとする. ID は元のブロードキャストメッセージ毎に定められる値で, これによりノードが複数のメッセージのブロードキャストを処理することができるようにする. 以降ではアルゴリズムがどのように動作するかを単一のブロードキャストメッセージに対して説明していく.

3.2 メッセージの送信

従来の手法と違い, 本手法では受信ノードは受け取ったメッセージそのまま送信するのでなく, 既に受信しているメッセージに符号化を施し新たなメッセージを作り出してそれを送信する.

ノードはバッファを持っており, 受信したメッセージをそこに格納することができるものとする. 図 2 では *Received* がバッファを示している. ノードがメッセージ m を受信したとき, 以下の 3 つの段階の処理がなされる.

- **Step 1:** m が意味のあるメッセージであるならばバッファに格納し, *Step* 2 に移動する. そうでなければ m を破棄する.
- **Step 2:** バッファから f 個の新たなメッセージを作り出し, ランダムに選んだノードに送信する. *Step* 3 に移動.
- **Step 3:** もしバッファ内のメッセージ数が k に達したら, オリジナルのブロードキャストメッセージを復号する.

*Step*1 : メッセージ m を受信したときバッファに既に $m_1, \cdots, m_{s-1}$ が格納されているとする. $m_1, \cdots, m_{s-1}$ と m は全て 2 つの部分から成る. 一つは $F_1, \cdots, F_k$ の線形結合から成るペイロード, もう一つは係数ベクトルである. $m_1, \cdots, m_{s-1}$ の係数ベクトルは線形独立である.

メッセージ m の係数ベクトルが $m_1, \cdots, m_{s-1}$ の全ての係数ベクトルと線形独立である時, m を意味のあるメッセージという. *Step* 3 で述べるように, オリジナルのメッセージは互いに線形独立な係数ベクトルを持つ k 個の任意なメッセージからのみ復号できるからである.

Step 2 : どのように新たなメッセージが作られるかを示す. メッセージは前述の通り二つの部分から成る. ペイロード部分は, バッファに蓄えられている $m_1, \cdots, m_s$ の線形結合から成る. ここで m_s は *Step* 1 で受信した新規メッセージである. 具体的には以下の式で計算される.

$$\sum_{i=1}^{s} a_i m_i$$

ここで a_i は $GF(q)$ 上からランダムに選ばれた係数である.

係数ベクトルの部分は, 以下の式で計算される.

$$\sum_{i=1}^{s} a_i e_i$$

ここで e_i は m_i の係数ベクトルを示す.

Step 3 : バッファ内に格納されているメッセージの数が k に達したら, オリジナルのブロードキャストメッセージの復号が可能になる. この時 k 個のメッセージの係数ベクトルは全て線形独立となる. 復号は k 元連立方程式を解くことによって行われる.

3.3 例

表 2: $GF(8)$ 上の加算表, 乗算表

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

·	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

本手法の簡単な例を示す. $GF(8)$ 上で演算を行い, $k = 3$ であるとする. 表 2 は $GF(8)$ 上での加算, 乗算を示す表である. オリジナルのブロードキャストメッセージは 36 ビットであり, 8 進表示では 12 桁の 111113612532 であるとする. この場合, k 個に分割されたメッセージは $F_1 = (1\ 1\ 1\ 1)$, $F_2 = (1\ 3\ 6\ 1)$, $F_3 = (2\ 5\ 3\ 2)$ となる.

あるノードが A と B の係数ベクトルが線形独立なメッセージを受信した時を考える. それぞれの係数ベクトルは (1 2 3) と (2 5 3) であるとする. それぞれのペイロードは以下の通りである.

$$A = F_1 + 2F_2 + 3F_3 = (5\ 3\ 3\ 5)$$
$$B = 2F_1 + 5F_2 + 3F_3 = (1\ 2\ 4\ 1)$$

このノードが, 係数ベクトルが (3 7 0) でペイロードが (4 1 7 4) である新たなメッセージを受け取ったとする. その場合, このメッセージの係数ベクトルは A と B の係数ベクトルと線形従属であるためバッファには保存されない. ((1 2 3) + (2 5 3) = (3 7 0))

一方, 係数ベクトルが (1 5 2) でペイロードが C = (0 4 4 0) であるメッセージを受け取ったとする. このメッセージの係数ベクトルは A と B の係数ベクトルと線形独立であるため, バッファに保存される.

ここで, 受信したメッセージの数が $k = 3$ と等しくなったため, このノードはオリジナルのブロードキャストメッセージを得ることが出来る. 具体的には以下の連立方程式式を解くことで F_1, F_2, F_3 を得る.

$$A = F_1 + 2F_2 + 3F_3$$
$$B = 2F_1 + 5F_2 + 3F_3$$
$$C = F_1 + 5F_2 + 2F_3$$

4 最適化

本節では前節の手法の改良を提案する. これは, 予備実験により, 提案手法の性能を調べた結果得られるパフォーマンスの向上がわずかであることが判明したためである. この原因として, 2 つの問題が基本提案手法にあることが分かった. これらを解決するため, 最適化を施した新たなアルゴリズムを図 3 に示す.

4.1 無意味なメッセージの拡散防止

問題の一つは, 極初期の段階ではメッセージの拡散が非効率的になる, ということである. あるノードが受信したメッセージが 1 つだけである時, 新たな送信メッセージを作成しようとするのなら, 全てのメッセージが線形従属となってしまう. このため無意味なメッセージが多くなり, オリジナルのメッセージを復号するためにより多くのメッセージを受け取らなくてはならないことになる.

これを防ぐため以下の対策をとる.

1. 初めてメッセージを送るノードに対しては 2 つのメッセージを送信する.
2. 2 つ以上メッセージを受信している場合のみ, 新たなメッセージを作り, 別のノードに送信する.

1 の対策は, メッセージを交換したことのあるノードを記憶しておくことで実現できる. *Contacts* というバッファがこのために使われる. これは最初は空であり, ノード p からメッセージを受信したり (Step 1), ノード p にメッセージを送信することによって (Step 2), 対象ノード名が格納されていく. もしノード p が *Contacts* の中にあれば, そのノードは既にメッセージを複数個受け取っていることが保証される. なければ p にメッセージを送信する際には 2 つのメッセージを作り出して送信する.

2 の対策は, 受信したメッセージ数が 1 の時には送信を行わないようにすることで容易に実現できる.

4.2 動的ファンアウト

もう一つの問題は, メッセージが十分に広まった段階では新たなメッセージの送信が無駄になることが多いことである. ノードが k 番目のメッセージを受け取った時, 多くの他のノードは k やそれに近い個数のメッセージを受け取っている. これは, メッセージの伝搬が全てのノードでほぼ等しく広まっていくためである. このような場合, 新たなメッセージを送信することは無駄なトラフィックを増やす要因となる.

これを解決するため, 動的ファンアウトを導入する. これは, メッセージを受信した数に伴いファンアウトを動的に減らすというものである. ファンアウトの値は関数 $f(|Received|)$ で与えられるものとする. ここで *Received* は受信した意味あるメッセージを格納しているバッファを示しており, $|Received|$ はそのメッセージ数である.

ネットワークの構造や k の値等の要素によって効率的な関数 $f(|Received|)$ は決定される. 本研究の現段階では事前にシミュレーションを行い $f(|Received|)$ を決定している.

5 シミュレーション結果

本節では, シミュレーションの結果を示す. 各々のノードはシステム中から, 等しくランダムにノードを選択できるものとする. これは *peer sampling service* [14] を使うことで実現できる. ノード数は 500 とする.

表 3: ファンアウト $f(|Received|)$

$\|Received\|$	1	2	3	4	5	6	7
$k = 4$	–	f_d	0	–	–	–	–
$k = 6$	–	f_d	2	0	0	–	–
$k = 8$	–	f_d	f_d	1	0	0	0

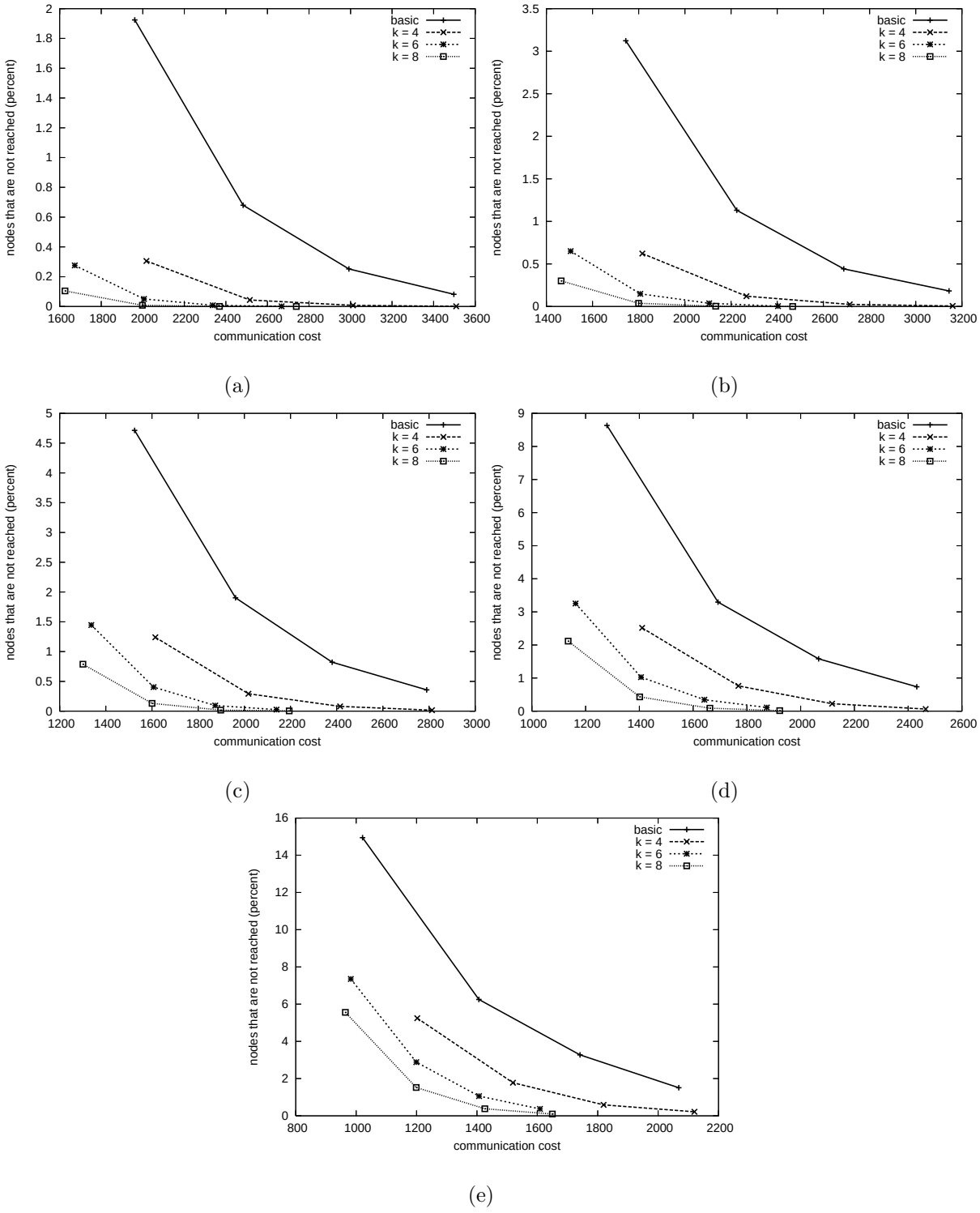

図 4: ノード数 500 の時のコミュニケーションコストとメッセージ未着のノード割合
(a) 0%, (b) 10%, (c) 20%, (d) 30%, (d) 40% の故障ノード

```
Initiation of broadcast of M:
  Divide M into k fragments F_1, ..., F_k;
  Choose a set RN of f_init random nodes;
  For p ∈ RN
    Create two messages msg, msg' from F_1, ..., F_k
      with random linear encoding;
    Send msg, msg' to node p;

When a node receives a message m:
  { Step 1 }
  If (msg is informative)
    Add msg to Received; { Received is initially empty. }
    Add p to Contacts; { Contacts is initially empty. }
    { Step 2 }
    If (|Received| ≥ 2)
      Choose a set RN of f(|Received|) random nodes;
      For p ∈ RN
        Create a message msg and send it to p;
        If (p ∉ Contacts)
          Create another message msg' and send it to p;
          Add p to Contacts;
    { Step 3 }
    If (|Received| = k)
      Decode the broadcast message from Received;
```

図 3: 最適化された提案手法アルゴリズム

演算は $GF(2^8)$ 上で行うものとする. 分割数 k は 4,6,8 と変化させる. 動的ファンアウトは表 3 の通りとする. f_d はデフォルトのファンアウトの値とする. この値は 4 から 7 とする. ブロードキャスト開始ノードのファンアウト f_{init} は $k * f_d$ と定める.

故障ノードの割合は 0%から 40%まで 10%ずつ変化させる. 故障ノードに送られたメッセージはすぐに破棄されるとする. ノードから送られるメッセージは指数分布に従う遅延時間後に受信ノードに到達する. ノードでの符号化によるオーバーヘッドは無視する. その代わり, メッセージの遅延時間に含まれるものと仮定する. また, メッセージの係数ベクトルはそのペイロードと比べて無視できる程度に小さいものとする. これは, ブロードキャストメッセージを十分に大きくすることで実際の環境でも成り立つ.

シミュレーションの結果を図 4 に示す. これらの図は信頼性と通信コストの関係を示している. 横軸は通信コストを, 縦軸はブロードキャストメッセージを復号できなかった正常なノードの割合を表している. 通信コストは, ブロードキャストメッセージをそのまま送信するときはメッセージ数, 符号化したメッセージを送信するときはメッセージ数の $1/k$ 倍とする.

図中の各点は f_d を $4, \cdots, 7$ まで一つずつ変化させた場合を示している. それぞれの点の値は,1000 回シミュレーションを行った平均値である. 4 つの線は, 従来のゴシッププロトコルと提案手法の $k = 4, k = 6, k = 8$ に対応している.

図から, 提案手法の方がより少ない通信コストで高い信頼性を達成していることが分かる. 例えば故障ノード割合 10%の図 4(b) では, $k = 8$, $f_d = 4$ の時に 0.3%の未到着のノードが存在するが, 通信コストは 1500 以下である. 従来のゴシッププロトコルではこの信頼性を達成するにはその 2 倍の通信コストが必要である.

他に分かることとして k の値が大きくなるほどにブロードキャストメッセージが届かないノードの割合が少なくなっていくことがある. これは実験を行った全ての場合で見られる. その理由として以下の 2 つが挙げられる. 一つは k を大きくするほど係数ベクトルの長さが大きくなり, 線形独立な係数ベクトルを作り出す可能性が減るからである. もう一つの理由は, メッセージが小さいほど伝搬するメッセージ量をコントロールしやすいためである. 例えば k が大きいほど, より効率的な動的ファンアウトの変化が可能となる.

6 まとめ

本論文ではネットワークコーディングを利用した新たなゴシッププロトコルを提案した. 提案手法では元のブロードキャストメッセージに符号化を施し, サイズの小さなメッセージを作り出してそれを転送する. 各ノードは元のメッセージの分割数 k に等しい, 線形独立な係数ベクトルを持つメッセージを受信することで復号が可能となる. これにより従来のゴシッププロトコルとは違い, 意味のあるメッセージを何回も受け取ることが可能になる.

シミュレーションにより, 提案手法が信頼性と通信コストのトレードオフを改善できることを示した. また, k の値が大きくなるほどより効果的となることが分かった.

しかし, 提案手法があらゆる環境で優れた結果を出すことができると断言することはできない. これは, シミュレーションで理想的な設定を仮定しているためである. 例えば, 符号化や復号化にかかるオーバーヘッドは今回考慮していない. 将来的にはより現実に即した設定で実験を行う予定である. また, 動的ファンアウトにおける関数 $f(|Received|)$ の定め方も問題として残っている. 本論文では予備実験で効果的であった値を使用したが, より具体的な定め方が課題として挙げられる.

参考文献

[1] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Transactions on Computer Systems*, vol. 17, no. 2, pp. 41–88, May 1999.

[2] P. T. Eugster, R. Guerraoui, S. Handurukande, A.-M. Kermarrec, and P. Kouznetsov, "Lightweight probabilistic broadcast," in *Proceedings of the 2001 International Conference on Dependable Systems and Networks (DSN '01)*, Jul. 2001, pp. 443–452.

[3] Q. Sun and D. Sturman, "A gossip-based reliable multicast for large-scale high-throughput applications," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2000)*, June 2000, pp. 347–358.

[4] D. Agrawal, A. El Abbadi, and R. Steinke, "Epidemic algorithms in replicated databases," in *Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems*, 1997, pp. 161–172.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the Sixth Ann. ACM Symp. Principles of Distributed Computing (PODC)*, Aug. 1987, pp. 1–12.

[6] R. van Renesse, Y. Minsky, and M. Hayden, "A gossip-style failure detection service," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Sept. 1998, pp. 55–70.

[7] A. Lakshman and P. Malik, "Cassandra - a decentralized structured storage system," in *3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS 09)*, Oct. 2009.

[8] A. Montresor, M. Jelasity, and O. Babaoglu, "Robust aggregation protocols for large-scale overlay networks," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*. IEEE Computer Society, 2004, pp. 19–28.

[9] R. van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 164–206, 2003.

[10] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs," in *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '08. New York, NY, USA: ACM, 2008, pp. 325–336. [Online]. Available: http://doi.acm.org/10.1145/1375457.1375494

[11] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[12] S. Deb, M. Médard, and C. Choute, "Algebraic gossip: A network coding approach to optimal multiple rumor mongering," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2486–2507, Jun. 2006.

[13] D. Mosk-Aoyama and D. Shah, "Information dissemination via network coding," in *Proc. ISIT*, Jul. 2006, pp. 1748–1752.

[14] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems*, vol. 25, August 2007. [Online]. Available: http://doi.acm.org/10.1145/1275517.1275520