

高速LANのモデルとその上での更新アルゴリズム

A network model of high-speed LANs and an information updating algorithm

齊藤 明紀⁺ 都志 武史⁺⁺ 辻野 嘉宏⁺ 都倉 信樹⁺

Akinori SAITOH

Takeshi TSUSHI

Yoshihiro TSUJINO

Nobiki TOXURA

⁺大阪大学

⁺⁺久保田鉄工

⁺Osaka University

⁺⁺KUBOTA limited

あらまし これまでに、多くの分散アルゴリズムが提案され、研究されている。分散アルゴリズムで一般的に用いられているネットワークモデルでは、通信が高価で遅いことを前提にしている。一方、イーサネットなどの高速LANは安価で遅延時間の短い通信を実現している。そのため、既に提案されている分散アルゴリズムは高速LANにはうまく適用できない。

そこで各ノードの停止/回復を含めた高速LANのモデル化を行い、ネットワークの結合性についても定義した。また、情報の更新を行う問題を解くアルゴリズムをこのモデル上で作成した。

Abstract Local area networks (LANs) such as Ethernet provide cheap and high-speed communication. We present a new network model of high-speed LANs. This model includes a concept of the connectivity on the LANs whose node-processors halt and/or recover asynchronously. We also propose an algorithm for an information update problem on this model. The correctness of the algorithm is proven.

1. 前書き

最近イーサネット、トークンリングなどの高速LANが普及してきた。その結果高速LANによる分散環境下で複数の計算機が協調して動作するようなソフトウェアに対する要求が高まっている。しかしながら、実際に使用されているソフトウェアは、ほとんどがホスト計算機（あるいはサーバ）においてそこで全体を管理する集中管理方式で実現されている。集中管理方式をとると、全体としての動作の一貫性が保ちやすく、プログラムの作成が比較的容易となる。しかし、サーバに負荷が集中することと、サーバの故障が全体に影響を与えるという欠点があり、特定のホスト計算機を仮定せず、各計算機が対等に動作する方式の方が望ましいと考えられる場合もある。

ある問題を解くために必要な情報がネットワーク上の各計算機に分散している状況で、メッセージのやりとりで情報を交換して、その問題を解くアルゴリズムは分散アルゴリズムと呼ばれる。これまでに、各種の分散アルゴリズムが提案され、研究されている。しかし、これまで提案された分散アルゴリズムで一般に用いられているネットワークモデルや評価基準は、通信遅延時間が大きく通信コストが高い広域ネットワークにうまく適合するもので、高速LANにはうまく適用できない。

そこで本研究では、高速LANの性質を反映したネットワークモデルを作成し、その上での情報更新問題を解くアルゴリズムを作成し、正当性を証明した。本稿では、分散アルゴリズムで一般的なものと異なるネットワークモデルを用いていることを陽に示すため、高速LANのモデル上での分散アルゴリズムをLANアルゴリズムと呼ぶことにする。

2. 高速LANのモデル

本節では、イーサネットなどの高速LAN（ローカルエリアネットワーク）に対応するネットワークのモデルについて述べる。

2.1 高速LANの性質

高速LANは一般に以下に示すような性質を持つ。特に性質（1）、（2）のため、一般に用いられている分散アルゴリズムのネットワークモデルの適用が困難である。

- （1）通信遅延時間が小さい。また、途中で長時間の蓄積をとまなう中継が行われない。このため、送信されたメッセージはある一定時間内に相手に届くことを期待してよい。いいかえると、ある一定時間以内に通信が成功しない場合にはタイムアウトで通信失敗と

みなすようなプロトコルを採用できる。また送信が相手
がメッセージを受信したことをもって送信の成功とする
場合もある（送受信の同時性）。

・通信遅延時間が小さいため、アルゴリズムの時間評価
を行うとき、計算処理時間が無視できない。

（２）通信のコストが小さい。

一般的な分散アルゴリズムのネットワークモデルでは、
通信が従量課金でまた計算処理コストよりも高価である
ことを想定し、通信量に注目してアルゴリズムの評価を
行なっている。一方高速LANでは、ケーブルを敷設し
てしまうと以後の通信路の使用にかかるコストは非常に
小さい。また、最近では通信プロトコルのかなりの層が
ハードウェアで実現されるようになってきており、ソフト
ウェアへの負担が軽くなってきている。このためアル
ゴリズムの実行にかかるコストを考えると、各計算機
での通信以外の処理に要するコストを無視すべきでない
と考えられる。

（３）通信路の故障は非常に少ない。

（４）ある２つの計算機は、他の計算機の状態に関係な
く、双方が動作中であれば通信できる。

（５）各計算機は、ユニークな識別子を持ち、ネットワ
ーク上の計算機は、そのネットワークに属する全計算機
の識別子を知っている。

（６）計算機の停止や回復が日常的に起きる。

利用者の誤操作による一時的なシステムダウンや、夜
間の停止など、偶発的な故障以外の停止や回復がかなり
頻繁に起きる。

2.2 モデルの設定

2.1節の考察にもとづいて作成した高速LANのネット
ワークモデルを以下に示す。

【定義１】ネットワークとノード

ネットワークを構成する各計算機をノードと呼ぶ。ネ
ットワーク N はノード $n_1, n_2, \dots, n_m$ の集合として定義
する。すなわち、 $N = \{ n_1, n_2, \dots, n_m \}$

ノード間の共有メモリは存在せず、通信はメッセージ
の送受信によってのみ行われる。リンクに関する定義が
ないことに注目されたい。これは性質（３）、（４）を反
映したものである。

【定義２】メッセージは、ノード間の通信でやりとりさ
れるデータである。メッセージ長は、最大メッセージ長
（定数）以下とする。

【定義３】各ノードは、それぞれ異なる識別子をあたえ
られている。

【定義４】各ノードは、あらかじめネットワーク内の全
ノードの識別子についての情報を与えられている。

【仮定１】各ノードはRAM（random access machine）
である。また、マルチプロセス方式をとり、複数のプロ
セスが並行して処理を行なうことができるものとする。
プロセスとは処理中のプログラムである。同一のプログ

ラムが複数並行して実行される時、それぞれは個別のプロ
セスとして扱う。プロセスには、実行中、実行待ち、
イベント待ちなどの状態があるものと考えられるが、詳
細については考えない。複数のプロセスの存在が許され、
なんらかのスケジューリング機能があって、それぞれの
プロセスが見かけ上、並行して動作することだけを仮定
する。

【定義５】ノードの状態には、run, stop の二つがあり、
ノードが

(a) 状態が stop であるとき停止中であるという。

(b) 状態が run であるとき動作中であるという。

(c) 状態が run から stop になることを停止という。

(d) 状態が stop から run になることを回復という。

(e) 回復時にあるプロセスを自動的に開始させることがで
きるものとする。

またある時刻 t におけるノード n の状態を $S(t, n)$ で表わ
す。

【定義６】各ノードは、非同期に停止、回復し、それは
予想不可能である。

定義５、６と仮定１は、性質（６）を反映している。

【定義７】ノードは通信プリミティブsendとreceiveを用
いて通信する。

・送信 send(in mes, dst) : (suc, fail)

ノード dst にメッセージ mes を送る。ネットワーク
上のノード数によって定まる一定時間 T_w 以内に終了し、
成功のとき suc を、失敗のとき fail を返す。

なお、相手ノードが停止しているとき、および相手ノ
ードにreceive 待ちのプロセスが現われないときに失敗
するものとする。この T_w を最大送信待ち時間と呼ぶ。
sendプリミティブがsucを返して終了したときにはすでに
相手先ノードにメッセージが到着しているものとする。

・受信 receive(out mes, sender)

メッセージを受けとる（mes, senderとともに出力引き数で
ある）。

どこかのノードからメッセージが送られて来るまで待ち、
mesに送られてきたメッセージを、senderに送信元ノード
の識別子をいれて返す。

（注）これらのプリミティブは性質（１）を反映して決
めたもので、それらの内部では、タイムアウト、再送、
アクノレッジの返送などの処理が行なわれていることを
想定している。 T_w は通信プロトコルによって規定されて
いるタイムアウトの時間である。実際のLANではノード
数に上限があるので T_w は定数であるが、ここではネット
ワークのノード数に依存し、 $O(\text{ノード数})$ だと考える。

【仮定２】各ノードは不揮発性記憶を持つ。

不揮発性記憶はディスクファイルを想定したものであ
る。ノードが停止した場合各プロセスの持つ情報（主記
憶上の変数）は失われる。

（注）ただし、不揮発性記憶は格納並びに参照等に要す

るコストは変数の参照、変更に比べて大きいものと考えられる。

〔定義8〕実行時間について定義する。

各プロセスは、処理を行ったり待機状態になったりする。そこで、実際に経過時間だけでなくその内の処理を行っていた時間についても定義する。

実行時間とは、経過した時間の事であり、待ち時間と処理時間の和である。

処理時間とは、実行時間から待ち時間を除いたもので、通信処理時間と計算処理時間の和である。処理時間はプログラムが計算機にける負荷を表わす。

通信処理時間とはsend, receiveの実行に要した時間から待ち時間を除いた時間のことである。送信待ち時間が最大 T_w であることを除けば、通信待ち時間は不定である。一方、通信処理時間は一般には、メッセージ長とプリミティブ内部で起きる再送の回数の積で決まると考えられる。ところがここではプリミティブの内部で起きる再送は性質(2)で述べたようにハードウェアによる下位プロトコルでソフトウェアにほとんど負担をかけずに行われると考えている。そこで、通信処理時間はメッセージの長さにも依存して決まるものとする。

2.3 ネットワークの結合性

一般の分散アルゴリズムでは、ある2ノードが直接通信できるかどうかはその間にリンクがあるかどうかでできる。よってネットワークの結合度はノードを頂点、リンクを辺としたグラフの連結度で評価される。しかし、このモデルでは任意の2ノードが通信できるがそれは双方が同時に動作しているときに限る。よってLANアルゴリズムに置けるネットワークの結合度とは、ノードが同時に動作している時間帯がどのように分布するかによって評価すべきである。

ネットワークとその上のノードについての結合性を記述する述語として次の四つを定義する。

〔定義9〕ノード n_1, n_2 についてconnect(τ, k, n_1, n_2)

iff

$\forall t (\tau < t < \tau + k \cdot T_w) :$

$(S(t, n_1) = \text{run and } S(t, n_2) = \text{run})$

{時刻 τ から少なくとも k 回連続して通信できる時間、両方のノードが動作中である。}

〔定義10〕

ノード n_1, n_2 について node_connect(t, k, n_1, n_2)

iff

$\forall T, \exists \tau (T < \tau < T+t) : \text{connect}(\tau, k, n_1, n_2)$

{任意時刻 T から時間 t の間に、connectである時刻が少なくとも一回はある。}

〔定義11〕

ネットワーク N について strong_connect(t, k, N)

iff

$\forall n_1, n_2 \in N : \text{node_connect}(t, k, n_1, n_2)$

{ネットワーク上の任意のノード n_1, n_2 のあいだに node_connect の関係が成り立つ。}

〔定義12〕

ノード n_1, n_2 について weak_connect(t, k, n_1, n_2)

iff

node_connect(t, k, n_1, n_2)

or

$(\forall T, \exists M_1, M_2, \dots, M_p, \tau_1, \tau_2, \dots, \tau_{p+1}$

$(T < \tau_1 < \tau_1 + k \cdot T_w < \tau_2 < \dots$

$< \tau_i < \tau_i + k \cdot T_w < \tau_{i+1} < \dots$

$< \tau_{p+1} < \tau_{p+1} + k \cdot T_w < T+t) :$

$(\text{connect}(\tau_1, k, n_1, M_1) \text{ and }$

$\text{connect}(\tau_2, k, M_1, M_2) \text{ and }$

$\dots \text{and } \text{connect}(\tau_{p+1}, k, M_p, n_2))$

{任意時刻 T から、直接またはいくつかのノードを経由して n_1 から n_2 へメッセージを送る経路及び機会があり、時間 t 以内に n_2 へメッセージが到着する。}

〔定義13〕

ネットワーク N について weak_connect(t, k, N)

iff

$\forall n_1, n_2 \in N \text{ weak_connect}(t, k, n_1, n_2)$

{ネットワーク上の任意のノード n_1, n_2 のあいだに weak_connect の関係が成り立つ。}

例1 strong_connect

全ノードが24時間運転しているようなLANを考える。故障による長期間の停止はなく、誤操作などによる一時的なシステムダウンはあるものとする。また各ノードはシステムダウンの後、速やかに再起動し、回復するものとする。

システムダウンから再起動に必要な時間が15分であるとする。連続稼働時間が15分、 $k \cdot T_w$ どちらと比べても十分に長ければ、このネットワーク N は、strong_connect(30分, k, N)であるといえる。

図3.1のように、通信要求が起きてから最悪でも30分待てば通信が成功する。また、二つのノードの停止期間の重なりを考えないときは、strong_connect(15分, k, N)であるといえる。

例2 weak_connect

必要な時だけ稼働させ、そのほかは停止させておくような運用形式をとるノードからなるLANを考える。昼間だけ動作しているノードや、逆に夜間のみ動作するノードが存在し得る。この様な状況では、直接の通信が行えないノードが存在し得る。

各ノードは少なくとも1日1回は動作する。すなわち、停止中のノードは最大48時間以内に回復する(たとえばある日の0時0分1秒に停止し、次の日の23時59分59秒まで回復しない場合)。また常に少なくとも2ノードが動作しているものとする、このネットワーク

は $\text{weak_connect}(48\text{時間}, k, N)$ である。

一般の分散アルゴリズムでは、ネットワークが連結であることが仮定されている。連結であるということは、いくつかのノードで中継を行えば、任意のノードへメッセージを届けることができるということである。LANアルゴリズムでは、 weak_connect がこれに対応する。 weak_connect が満たされないということは、他のノードによる中継をおこなってもメッセージのやりとりができないノードが存在するということであるから、一つのネットワークとはいえない。そこで、LANアルゴリズムの対象であるネットワークでは、次の仮定が成り立つものとする。

【仮定3】ネットワークNはある t と k について、 $\text{weak_connect}(t, k, N)$ を満たす。

3. 問題の設定と考察

この節においては、前節で構築したLANモデルの上での、情報更新問題を提示する。情報更新問題の一例としてネットワーク環境でのパスワード管理問題がある。これは、全計算機を同じパスワードで利用する事と、任意の計算機からパスワードを変更することを実現する問題である。またどの計算機でパスワードを変更しても、最新のパスワードが全計算機に伝達される必要がある。パスワードの場合は、利用者の数だけ項目があるが、ここで考える情報更新問題では1項目だけをあつかうが一般性は失われない。

3.1 情報更新問題

目的:

ネットワーク上の全ノードが同じ情報をもっていなければならない項目がある。その項目を配布対象情報と呼ぶ。ネットワーク上の全ノードで、非同期に配布対象情報の更新要求が起き得る。配布対象情報が更新されたとき、ネットワーク上の全ノードにその配布対象情報を届けて更新させ、全ノードが同じ最新の配布対象情報をもつようにする。

情報更新問題とは上記の目的を実現するアルゴリズムを2節で示したモデルの上で、以下に示す(1)~(4)の条件のもとで作成する問題である。

条件:

(1) あるノードでの情報更新要求の発生間隔は T_u 以上であるとする。

発生間隔に何も制限を置かない場合には、情報更新問題を解くアルゴリズムが存在しないことは自明であり、なんらかの制限を置く必要がある。

(2) ネットワークNに対してある t が決まり、 $\text{weak_connect}(t, k, N)$ を満たす。strong_connectは仮定しない。

(3) 情報発生直後に計算機が停止した場合、その情報

が失われることは許す。

このモデルでは、ノードの停止はいつでも起き得る。よって発生した情報を不揮発性記憶に格納する前に停止した場合には、その情報は失われるし、これを回避する方法はない。よってこの条件をおく。

(4) 異なるノードで発生した更新要求の時間順序を判定する方法はあるものとする。また、配布対象情報は発生したノードと発生時間順序によって区別され、内容が同じであっても、異なる時刻あるいは異なるノードで発生した情報は区別されるものとする。

配布対象情報の新旧判定は情報更新問題の一部であるが、ここでは独立した別の問題として扱う。各ノードが同期した時計を持っていればタイムスタンプで実現できる。また同期した時計を持たなくてもある程度の誤差を許せば判定可能である[3]。

(5) 停止したノードは少なくとも T_u の間は回復しない。
注) (2)での t は与えられた条件であるが、 k はこの問題を解くアルゴリズムによって定まるものである。全ノードが同じ情報を持つのに必要な時間が短いほど、また k の下限が小さいほどよい解であるといえる。

3.2 考察

この問題の重要な点を考察する。

- ・ weak_connect では、時間 t 以内に任意のノードへメッセージを伝達するパスの存在を保証しているが、そのパスはあらかじめ分かる事は保証していない。そこで、このパスを確実に利用することが問題となる。
- ・停止はいつでも起き得るので、どこで停止してもよいようにアルゴリズムを構成する必要がある。

4. 更新問題を解くLANアルゴリズム

この節では、前節で示した問題を解決するLANアルゴリズムを示す。

4.1 アルゴリズムのアイデア

この問題では、各ノードは、途中の履歴に関係なくいちばん新しい情報のみを持っていればよい。またこの問題では、 weak_connect を仮定しているだけであり、停止しないか、あるいは他のノードと比べて停止しにくい特定のノードの存在は仮定できない。そこで特定のノードに特殊な役割を与えるのではなく、全ノードが対等に働くアルゴリズムが望ましいと考えられる。

各ノードでの処理をイベント駆動形式で書くと、以下のようなになる。

- 1) 配布対象情報の更新要求が起きたら、自分の持つ配布対象情報を更新する。
- 2) 配布対象情報が更新されたら、全ノードに配布対象情報を送信する。
- 3) 新しい配布対象情報を受信したら、自分の持つ配布対象情報を更新する。

4)自分が持つと同じ配布対象情報を受信した場合には無視する。

5)古い配布対象情報を受信したら、その発信元ノードに自分の持つ配布対象情報を送信する。発信元ノードとはメッセージの送り元であり、メッセージに含まれる情報が発生したノードとはかぎらない。

6)回復したときには、全ノードに配布対象情報を送信する。

4.2 アルゴリズム

4.1で示したアイデアに基づいて作成したアルゴリズムを以下に示す。

4.2.1 アルゴリズムに関する定義と仮定

この節では、このアルゴリズムに適用する定義を行ない、仮定を示す。2節で示したLANのモデルについての定義はそのまま使用する。

・アルゴリズムはPASCAL風のプログラムを用いて記述する。

・ノードの識別子の型は、NODE_IDとする。

・各ノードは自分の識別子とネットワーク上の全ノードの識別子を変数myname、netとして参照できるものとする。型はそれぞれ、

```
myname:NODE_ID;  
net:set of NODE_ID;
```

とする。

〔仮定4〕配布する必要がある情報を配布対象情報と呼ぶが、それには新旧判断に必要な付加的なデータも含まれているものとする。

(1) データ型

各データの型と型名を図4.1のように決める。

(a) 配布対象情報は、以下の要素をもつレコード型である。

発生ノードの識別子、
情報、

(b) あとで述べるようにこのアルゴリズムでは1つだけ、待ち行列を使用する。待ち行列に入れるデータは以下の要素をもつレコード型である。

配布対象情報の種類（受信によるか、発生によるか）

送信してきたノードの識別子、

配布対象情報、

(c) メッセージの種類はただ1種類で、発信元ノードのもつ配布対象情報を含むものである。よって通信に関するプリミティブの引数と返す値の型を以下のように決める。

```
・function send(mes:MESSAGE;dst:NODE_ID);  
    (suc, fail)  
・procedure receive(var mes:MESSAGE;  
    sender:NODE_ID)
```

また、アルゴリズムが用いるプリミティブについて以下の仮定をおく。

〔仮定5〕待ち行列についての仮定

このアルゴリズムでは待ち行列を使用するが、以下のようなプリミティブで操作されるものとする。待ち行列は共有変数であり、プロセス間の通信に利用できるものとする。待ち行列は一つだけなので、待ち行列を識別するための引き数は省略する。

(a) procedure enqueue(data:QUE_ENTRY)

待ち行列の中にデータの一つ入れる。また、複数のプロセスによるenqueue 操作が同時にあるいは重なって行われた場合でも、問題なく待ち行列への入力が行われるものとする。

また、enqueueの処理時間は、Twよりも十分に短いものとする。

(b) procedure dequeue(var data:QUE_ENTRY)

待ち行列の先頭からデータの一つ取り出す。待ち行列の中にデータがない場合には、データが入って来るまで待つものとする。

(c) function empty_queue(): (true, false)

待ち行列の中にデータがないときには true、あるときには falseを返す。

(d) procedure clear_queue()

待ち行列を初期化する。

〔仮定6〕プロセスを起動させるプリミティブとして次のものが利用できるものとする。

・ procedure start(proc:STRING)

プログラム名（文字列） proc のプロセス（プログラム）を起動する。なお、これ以外に、定義5より、ノード回復時に自動的にプロセスを起動させることもできるものとする。

〔仮定7〕 配布対象情報が発生したときに、getプリミティブでその配布対象情報を取り入れるものとする。

・ procedure get(var newinf:INFO)

配布対象情報が発生して来るまで待ち、newinf に入れる。

〔仮定8〕不揮発性記憶の操作

各ノードは、配布対象情報を不揮発性記憶に格納するが、次の2種のプリミティブが利用できるものとする。

procedure store_info(inf:MESSAGE);

procedure store_newinfo(inf:MESSAGE);

store_newinfo は、新しく発生した情報を格納するときに使用する。タイムスタンプ等、情報の新旧比較に必要な付加情報の設定も併せて行うものとする。また双方とも、格納した情報を、プログラムから参照できる変数 myinfo にも格納するものとする。よって、不揮発性記憶の参照は常にそれと同じ内容を保持する揮発性記憶 myinfo に対して行われる。

これらのプリミティブの実行途中でノードが停止した場合でも、不揮発性記憶は完全に書き変わっているか、あるいは古い情報のまままったく変化していないかのいずれかであるとする。

4.2.2 アルゴリズムの概略および全体の構成

この問題では配布対象情報の発生及びメッセージの受信は非同期に起こる。そこで主な処理を行うプロセスのほかに、それぞれの入力イベントを専門に扱うプロセス2つと待ち行列を用いて2種類のイベントを直列化する。つまり、受信するだけのプロセス、発生したものを取り入れるだけのプロセス、比較、配布などの処理を行なうプロセスとあわせて三個のプロセスを用いてこのアルゴリズムを実現する。これらのプロセスの関係を図4.2に示す。

- ・回復時には、回復時起動プログラムが実行され、これによって、待ち行列の初期化、不揮発性記憶の読みだしなどの初期化処理が行われ、受信処理プログラム、発生処理プログラム、配布処理プログラムを起動される。
- ・受信処理プログラムは、receiveを実行して受信した配布対象情報を待ち行列に格納して、再びreceiveを実行して受信待ちとなる。

4.2.1で述べたように、待ち行列に入れるデータには、実際に受け取った配布対象情報、その送り元ノードの識別子、及び受信したデータであることを示すフラグを含む。

- ・発生処理プログラムは、getを実行して発生待ちの状態となり、発生時は発生した配布対象情報を待ち行列に格納して、再びgetを実行して発生待ちとなる。

前述のように、待ち行列に入れるデータには発生した配布対象情報及びこのデータが発生によるものであることを示すフラグを含む。

図4.3、4.4、4.5に回復時処理、受信処理、発生処理プログラムを示す。

4.2.3 配布処理プログラム

このプログラムが、このアルゴリズムの中心である。本プログラムを図4.6に示す。

以下、アルゴリズムについて説明する。

(1) メインルーチン。

配布処理プログラム起動時、自分の配布対象を全ノードに設定する(変数range)。そして、次の二つを繰り返して行う。

1) 指定の範囲に情報を送信する。

送信中に、待ち行列への入力があれば、その処理をさきに行う。その結果、更新が起きていれば、処理を中断し、新たに設定された配布範囲への情報送信を始める。

2) 待ち行列への入力を待ち、それを処理する。処理の結果、送信が不必要なら繰り返し待つ。そうでなければ1)へ。

(2) msgprocessは、待ち行列に入っているデータの処理を行う。待ち行列が空になるまで繰り返し処理する。ただし、最初から待ち行列が空ならば、入力されるまで待つ。

待ち行列から取り出した各データについて以下の処理をおこなう。

A) データが発生した配布対象情報ならば、不揮発性記憶に書き込む。また、配布対象を自分を除く全ノードに設定する。

データが受信したメッセージならば、それに含まれる配布対象情報と自分のもつ配布対象情報を比べる。

B) 新しいとき：不揮発性記憶に書き込み、配布範囲を自分と送り元ノードと発生ノードを除く全ノードに設定する

C) 古いとき：送信元ノードを配布対象に追加する。ただし、配付対象情報の更新がすでに起きている場合には配付対象には全ノードが設定されているはずなので何もしない。

D) 同じとき：無視

待ち行列が空になったら終了する。処理の過程で情報の更新が行われていたら、UPDATEDを返す。更新が行われなかったが配付対象の追加があった場合には、OLDINFOを返す。待ち行列の内容がすべて自分の持つ配付対象情報と同じだったら、NOTUPDATEDを返す。

5. アルゴリズムの評価と検討

5.1 アルゴリズムの検討

このアルゴリズムは以下のようなアイデアで作成されている。

全ノードが同じ情報を持つために、まず、

(1) 配布対象情報が発生したノードは他の全ノードに送信する。

しかし、これだけではまだ不十分であり、以下のような問題が起きる。

1) あるノードが送信してもそのときに停止していたノードはその情報を受け取れない。

2) 情報が発生したノードが他のノードへの送信を終了しないうちに停止した場合、情報が受け取れないノードが有り得る。

3) あるノードで配布対象情報が発生し、更新した後、送信が一度も成功しないうちにそのノードが停止した場合、他のどのノードへも新しい情報が伝わらない。

よって、情報の発生源のノード以外のノードもなんらかの形で配布を行うべきである。また回復した時には、3)のような場合に対処する動作をしなければならない。そこで、

(2) 発生したノードだけでなく、情報を受信して更新したノードも他の全ノードの情報を送信する。

(3) 回復したノードは、他のすべてのノードと通信し、どちらの持つ情報が新しいか調べる。古い情報を持つノードは更新する。

ということを言えばよい。

このアルゴリズムの正当性を証明するにはまずネットワーク上の全ノードが、同じ最新の配布対象情報を持つ

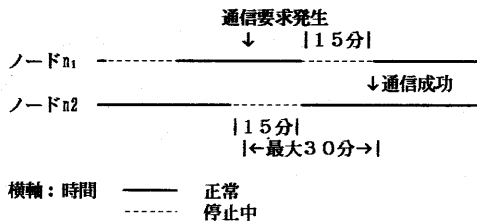


図3.1 strong_connect の例

```

type
  NODE_ID=(モデルにより規定)
  INFO={通信する配布対象情報の型}
  MESSAGE=record
    src_node:NODE_ID;
    data:INFO;
  end;
  QUE_ENTRY=record {待ち行列のデータの型}
    generated:boolean; {配布対象情報の種別}
    sender:NODE_ID; {配布対象情報の送信元}
    mes:MESSAGE; {配布対象情報}
  end
  COMPTYP=(NEWER, OLDER, SAME); {情報比較 compareの型}

```

図4.1 データの型

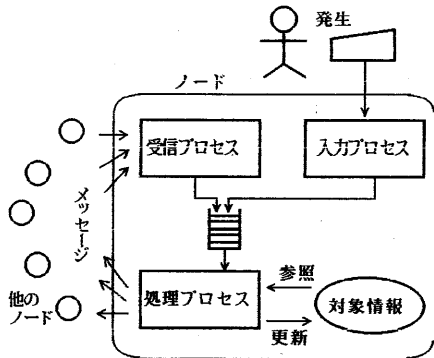


図4.2 プロセス関係図

```

program kaifuku;
  procedure clear_queue;extern;
begin
  clear_queue;
  start('receipt');
  start('local_request');
  start('management');
end.

```

図4.3 回復時起動プログラム

```

program receipt; {受信したデータを格納する。}
var entry_data:QUE_ENTRY; {queueに格納するデータ}
begin
  while (true) do begin {常に受信待ち}
    receive(entry_data.mes, entry_data.sender);
    entry_data.generated:= false; {受信によることを示す}
    enqueue(entry_data); {データを格納する}
  end
end.

```

図4.4 受信処理プログラム

```

program local_request;
var entry_data:QUE_ENTRY; {queueに格納するデータ}
begin
  while (true) do begin {常に発生待ち}
    get(entry_data.mes.data);
    entry_data.generated:=true; {発生によることを示す}
    enqueue(entry_data); {発生したデータを格納する}
  end
end.

```

図4.5 発生処理プログラム

```

program management
label 10;
type RESULT=(IGNORED, OLDINFO, UPDATED);
var {なし, 返送, 新配布}
  dst: NODE_ID;
  range, r : set of NODE_ID;
  {配布対象情報を配布すべき範囲}
  myinf:MESSAGE; {自分自身の情報}
function msgprocess:RESULT;
var qtop:ENTRY_DATA;
  returnval:COMPTYP;
begin
  returnval := NOTUPDATED;
  with qtop do
    repeat
      dequeue(qtop);
      if generated then begin {発生時}
        store_newinfo(mes);
        range:= net - [myname];
        returnval:=UPDATED
      end else case compare(mes, myinfo) of
        NEWER : begin {受信時}
          store_info(mes);
          range:=net-[myname, sender, mes.src_node];
          returnval:=UPDATED end;
        OLDER :
          if returnval <> UPDATED then begin
            returnval = OLDINFO;
            range := range+[sender];
          end
        SAME : {ignore}
      end; {esac}
    until empty_queue;
    msgprocess := returnval;
  end;
end;

```

```

begin {メインプログラム}
  range:= net - [myname];
  {回復時, 自分の配布対象情報の配布を指定}
  while (true) do begin
    10:
    while(range <> []) do begin
      r:=range;
      range:=[];
      foreach dst in r do begin
        if not empty_queue then
          if msgprocess = UPDATED then
            goto 10;
          send(myinf, dst);
        end;
      end;
      while msgprocess = IGNORED do
        ;
      end
    end.
  end.

```

図4.6 配布処理プログラム

ようになる(アルゴリズムの完全性)ことを示す必要がある。またこのアルゴリズムでは発生源以外のノードも配布対象情報を更新する毎に情報の配布を行うので、ネットワーク上の全ノードが同じ配布対象情報を持てば、発生及び回復がない状態でネットワークが定常状態(定義15参照)になる(アルゴリズムの停止性)ことも示す必要がある。

5.2 評価のための諸定義

アルゴリズムの時間評価は、一般にはアルゴリズムが停止するまでに要する時間で行う。しかし更新問題のアルゴリズムは停止しない。そこで、ノードの休止状態を定義し、各ノードが休止に至るまでに必要な時間でアルゴリズムの時間評価を行う。

【定義14】ノードが動作中で、ノードの全プロセスが・receiveプリミティブによる受信待ちであるか、あるいは

・dequeueプリミティブによるキューのデータ入力待ちであるとき、

そのノードは休止中であると言う。sendプリミティブでの通信待ち状態は、休止状態ではない。

【定義15】ネットワークが定常状態とは、ネットワーク上の全てのノードが停止中かまたは休止中である状態をいう。

アルゴリズムの評価を行なうために、いくつかの定義を行なう。まず記法について定義する。

【定義16】

(1) ネットワーク上の全ノード数: N (一定)

(2) 時間:

・ T_r : 不揮発性記憶への格納に要する時間

・ T_c : メッセージを受信したときに要する最大計算処理時間から T_r を引いた残り(補題2参照)。不揮発性記憶の参照は揮発性記憶上にあるコピーに対して起きるので、実際に不揮発性記憶にアクセスするのは書き込むときのみ。

【仮定9】 $T_w > NT_c$, $T_w > T_r$ と仮定する。

定義7の注より $T_w > NT_c$ は妥当な仮定である。

また $T_w > T_r$ を仮定しないと、ローカルに起きる更新要求にすら対応できなくなる(発生間隔 $> T_w$)。

5.3 アルゴリズムの完全性

まず証明に必要な補題と系を示す。

【補題1】休止状態のノードは、回復したときに保持していた配布対象情報と回復以後受信した情報と回復以後そのノードで発生した情報のうち、最新のものを保持している。□

(証明)受信した情報と発生した情報は待ち行列に格納され、失われることはない。また、配布処理プログラムは待ち行列から取り出した情報が新しいときはかならず

更新を行う。休止状態のノードは、待ち行列が空であり、発生及び受信した情報はすべて配布処理プログラムによって処理されている。よって補題1は成り立つ。□

【補題2】計算処理時間は、情報の発生回数と送受信したメッセージ数の和に比例する。□

(証明)

アルゴリズムの作り方から、発生、受信処理プロセスでの計算処理時間がそれぞれ発生、受信回数に比例する。

配布処理プロセスのメインルーチンでの計算処理時間は、関数msgprocess内での処理を除けば、送信回数に比例している。

msgprocessでの計算処理時間は、待ち行列の中のデータ数に比例している。待ち行列に入れられるデータの数は、メッセージの受信と情報発生回数の和である。

よって、計算処理時間は、情報の発生回数と送受信したメッセージ数の和に比例する。□

定義16で、メッセージを一つ受信したときに必要な計算処理時間の最大から不揮発性記憶への格納に要する時間を除いたものを T_c と定義した。

次にノードの回復と配布対象情報の送信に関する補題を示す。

【補題3】ネットワーク上で時間 $N \cdot T_w$ のあいだにノードが回復する回数は、 N^2 以下である。□

(証明)問題の条件(5)より、各ノードは一度停止すると T_w 以上の時間回復しないから、 $N \cdot T_w$ 時間のあいだに回復するのは、高々 N 回である。

よって、ネットワーク上の全ノードについて、任意の $N \cdot T_w$ 時間のあいだに停止し、そして回復してくる回数は高々 N^2 回である。□

【補題4】新情報発生がない長さ $m \cdot T_w$ の期間に、あるノード n にあてた送信は高々 $(2m+N)(N-1)$ 回しか起きない。発生も回復もなければ $N(N-1)$ 回である。またその間正常であり続けるノード n への送信は、高々 $(m+N)(N-1)$ 回である。□

(証明)

(i) この期間内には情報の発生による配付は起きない。よって他のノードで情報の発生により行われる n への送信は0回である。

(ii)回復したノードはすべてのノードにメッセージを送信する。ノードは時間間隔 T_w に高々1回しか回復しない。よって T_w ごとにすべてのノードが停止、回復したとしても回復による配布は mT_w の間に高々 mN 回しか起きない。しかし、自分自身には送信しないので、ノード n への送信は mT_w あたりたかだか $m(N-1)$ 回である。

(iii)回復時に行う配布処理の結果、回復したノードより新しい情報を持つノードは、その情報を回復したノードに送信する。他のすべてのノードが自分より新しい情報を持っていたとしても、この送信は1回の回復あたり高々 $N-1$ 回しか起きない。よって mT_w のあいだに、 n の回復時処理の結果ノード n に送られてくるメッセージの数はた

かだか $m(N-1)$ である。

(iv) 各ノードが持つ配布対象情報の種類は高々 N 個である。

新しい配付対象情報を受信する回数は、すべてのノードが異なった情報を持っていたときが最大で、

1 番古い情報を持つノード $N-1$ 回

2 番目に古い情報を持つノード $N-2$ 回

$N-1$ 番目に古い情報を持つノード 1 回

1 番新しい情報を持つノード 0 回

となる。よって回復時処理を除いた全ノードへの配付は高々 $N(N-1)$ 回しか起きない。よって新しい情報の受信による配布で、ノード n あての送信はたかだか $N(N-1)/2$ 回である。

(v) 更新の結果行う配布でも (iii) と同様に情報が送られて来る。但し、1 回更新する毎に自分より新しい情報を持つノードは少なくとも 1 つづつ少なくなる。よってノード n が情報を更新した結果送信したメッセージに対して起きる n への送信は、高々

$$(N-1) + (N-2) + \dots + 2 + 1 + 0 = N(N-1)/2 \text{ 回.}$$

よって (i) ~ (v) より新しい情報の発生がない期間に、あるノード n にあてて起きる送信の回数は、 mT_w の間に高々 $(2m+N)(N-1)$ 回、発生も回復もなければ $N(N-1)$ 回である。また $m \cdot T_w$ の間に正常であり続けるノードへの送信は、高々 $(m+N)(N-1)$ 回である。□

補題 4 から次の系が得られる。

【系 4】 発生がない期間には、待ち行列のなかのデータの個数は、 $(N+1)(N-1)$ をこえない。□

(証明)

このアルゴリズムでは待ち行列が空になるまで送信は行わない。1 回の送信は高々 T_w で終了するので、この間に受信するメッセージは高々 $(1+N)(N-1)$ 個である。□

次に、配布処理に関する補題を示す。

【補題 5】 アルゴリズムにおけるある配布対象情報に関する一回の (全ノードに対する) 配布 (メインルーチンの for each 文) はたかだか $K_B \cdot T_w$ 以内に終了する。ここで、 $K_B = (N-1) + (2N-1)(N-1)T_c / (T_w - (N-1)T_c)$ である□

(証明) アルゴリズムにおける一回の配布は高々 $(N-1)$ 回の送信である。従って、配布途中に待ち行列に対する処理がなければ、 $(N-1) \cdot T_w$ 時間以内に処理は終了する。

ところがモデルから、各ノードは非同期に停止、回復、情報の発生を行なう。また、配布中に、情報が発生する、または他のノードから情報を受け取ることがある。その際には、発生したあるいは受け取ったときに行なっている送信が終了後、

・発生した場合、実行中の配布を中止して新しい配布を始める

・受け取った場合、受け取った情報と自分の情報を比較

して、

(1) 新しいとき、実行中の配布を中止して新しい配布を始める

(2) 古いとき、送り元ノードを再配布リストに書き込み、配布続行する。

(3) 同じとき、何も行なわず、配布続行する。

という処理を行なっている。

従って、配布中に情報を発生したときおよび新しい情報を受け取ったとき、その配布は中止する。よって、配布に要する時間が最大になるのは、受信した情報がすべて古いかまたは同じであった場合である。

一回の配布中、送信には最大 $(N-1)T_w$ 時間を要する。それと配布中に受信したメッセージの処理に要する時間上界の和が配布に要する時間の上限となる。

配布中に受信したメッセージの処理に要する時間を xT_w とする。受信した結果、更新は起きないので受信するメッセージ 1 つあたりの処理時間は高々 T_c 。補題 2 より、 $x \cdot T_w$ の間に受信するメッセージはたかだか $(x+N)(N-1)$ 回、 $(N-1)$ 回のメッセージの送信と $(x+N)(N-1)$ 回のメッセージの受信処理の時間が xT_w を越えないから、

$$xT_w < (N-1)T_w + (N+x)(N-1)T_c$$

$$x(T_w - (N-1)T_c) < (N-1)T_w + N(N-1)T_c$$

仮定 D より、 $T_w - (N-1)T_c > 0$ であるから、

$$x < ((N-1)T_w + N(N-1)T_c) / (T_w - (N-1)T_c)$$

$$= ((N-1)T_w + N(N-1)T_c) / (T_w - (N-1)T_c)$$

$$= ((N-1)T_w + (N+(N-1) - (N-1))(N-1)T_c) / (T_w - (N-1)T_c)$$

$$= (N-1) + (2N-1)(N-1)T_c / (T_w - (N-1)T_c)$$

よってたかだか

$$((N-1) + (2N-1)(N-1)T_c / (T_w - (N-1)T_c)) T_w$$

以内に一回の配布は終了する。□

【補題 6】 最新の配布対象情報をもつノード n_1 に配布対象情報を送ったノード n_2 は、停止しなければ、

$2K_B \cdot T_w$ 時間以内に最新の配布対象情報をもつ。□

(証明) n_1 は自分のもつ配布対象情報より古い配布対象情報を n_2 から受け取ると、配布処理終了後、 n_2 に n_1 の持つ配布対象情報を送信する。

従って、補題 5 より、 n_2 からのメッセージを受信した時点で n_1 が配布作業中だったとしても、それは $K_B \cdot T_w$ 時間以内に終了する。また、一回の配布中に古い配布対象情報を n_1 に送ってくるノードは n_2 を含めて高々 $(N-1)$ 個ゆえ、それに対する送信処理も $K_B \cdot T_w$ 時間以内に終了する。

よって合計 $2K_B \cdot T_w$ 時間以内に n_2 は最新の配布対象情報を受信する。□

【定理 1】 ネットワーク N が weak-connect (t, k, N) であれば、あるノード n_1 である最新の情報 inf が発生してから後、ネットワーク内で新たな情報の発生がないまま、時間 t が経過すれば、 inf をネットワーク上の全ノードが持っている。ただし

$$k > 3 K_B + ((N+1)(N-1)T_c + T_r) / T_w. \quad \square$$

(証明)

ネットワークが $\text{weak-connect}(t, k)$ を満たしているの
で任意の n_2 に対してパス $n_1, m_{2,1}, m_2$ が存在し、 m_1 と $m_{1,1}$
及び m_2 と n_2 が $K \cdot T_w$ 以上連続して同時に動作している期間
が存在する。

それぞれ期間の始まりに注目する。

(i) m_1 が配布対象情報を inf に更新した時点あるいは m_1 が
回復した時点が、期間の始まりである場合、 m_1 は情報を
配布するが、補題 6 より、 $K_B \cdot T_w$ 以内に m_1 は inf の配布を
終了する。系 5 より、 $(N-1)(N+1)T_c + T_r$ 以内に $m_{1,1}$ は待
ち行列から inf を取り出して処理し、配布対象情報を inf
に更新する。

(ii) $m_{1,1}$ が回復した時点が期間の始まりである場合。

補題 6 より $K_B \cdot T_w$ 以内に $m_{1,1}$ は回復時処理の配布を終了
する。補題 7 よりさらに $2K_B \cdot T_w$ 以内に $m_{1,1}$ は m_1 から inf
を受信する。系 5 より、 $(N-1)(N+1)T_c + T_r$ 以内に $m_{1,1}$ は
待ち行列から inf を取り出して処理し、配布対象情報を
 inf に更新する。

(i)(ii) より、 m_1 と $m_{1,1}$ が同時に動作し続ける期間が $3K_B \cdot T_w + (N+1)(N-1)T_c + T_r$ 以上あれば、 m_1 から $m_{1,1}$ に
 inf が伝わる。よって、

$K > 3K_B + (N+1)(N-1)T_c + T_r / T_w$ ならば、 n_1 で発
生した情報は t 以内にすべてのノードに伝わる。

n_1 で発生した inf は最新情報であるので系 1 より、 n_1 で
 inf が発生してから時間も後にはすべてのノードが inf を
保持している。□

5.4 アルゴリズムの停止性

5.3 節では、アルゴリズムによって全ノードが同じ情
報を持つようになることを示した。この節では、アルゴ
リズム実行によって、ネットワークが定常状態にはいる
ことを示す。

〔定理 2〕ある時点 T でネットワーク上の全ノードが
同じ情報を持って、以後発生、回復がなく、また発生し
て待ち行列に入れられてまだ配布プロセスによって処理
されていない配布対象情報がなければ、 $(N-1)T_w$ の後には定
常状態になる。□

(証明) 全ノードがある時刻 T に同じ配布対象情報 inf を
持ったとする。 T で、待ち行列に入っているメッセージ
が含む配布対象情報は、 inf と同じかそれより古いかであ
る。

T で配布中であるか、配布を始めたノードは $N-1$ 回送信
すると配布を終了し、待ち行列が保持している古い情報
の送信元へ自分の持つ情報を送る。この送信は高々 $N-1$ 回
である。 T 以降送信されるメッセージはすべて inf を含む
ので、単に無視される。よって T 以降送信されたメッセ
ージに起因する配布や送信は起きない。

T 以降あるノードが受信するメッセージ数は高々 $2(N-1)$ 。
 T に待ち行列が保持しているメッセージは系 4 よりたか

だか $(N+1)(N-1)$ 。

よって時刻 $T + (N-1) \cdot T_w + (N+3)(N-1)/T_c$ で送信と受信
したメッセージの処理は終了し、待ち行列は空になる。
よってすべての正常なノードで配布処理プログラムは待
ち行列待ちになり、発生処理プログラム、受信処理プロ
グラムはそれぞれ発生待ち、受信待ちとなる。すなわち
定常状態となる。

従って、ネットワーク上の動作中の全ノードが同じ情
報を持って、発生、回復なく、

$(N-1) \cdot T_w + (N+3)(N-1)/T_c$ 経過すれば必ず定常状態には
いる。□

これによって、新たな発生なく、全ノードが最新の情
報をもてば定常状態にはいることを示した。

定理 1, 2 よりこのアルゴリズムは正しいことが示せた。

6. むすび

本稿では、高速 LAN の性質を反映したネットワーク
モデルについて述べた。このモデルは送信と受信の同時
性や、非同期に起きるノードの停止と回復など高速 LAN
の特徴をモデル化しており、ネットワークの連結性の
尺度についても定義している。

またこのモデル上での情報配布問題を解く LAN アル
ゴリズムを作成し、正当性を証明した。ここで示したアル
ゴリズムは単純ではあるが、ノードの停止や回復に、
頻度以外には制限をおかずに解いている。

今後の課題としては、情報配布以外の問題を解くアル
ゴリズムの作成すること、このモデルでのネットワーク
の結合度やアルゴリズムの良さの尺度の研究などがある。

参考文献

- [1] 上谷晃弘: "ローカルエリアネットワーク
— イーサネット概説 —", 丸善株式会社, (1985).
- [2] 萩原兼一: "分散アルゴリズムの複雑度について",
Proceedings of The Logic Programming
Conference '87, pp11-20, (June 1987).
- [3] Leslie Lamport: "Time, Clocks, and the
Ordering of events in a Distributed System",
Communications of the ACM, Volume 21,
Number 7 (July 1978).