IPネットワークの経路設計と検証の手法について

鶴　正人[1*]、黒田　英夫[2†]
[1]長崎大学総合情報処理センター
[2]長崎大学工学部電気情報工学科

あらまし：LANの大規模化に伴い、その設計／保守の複雑さが増大し、勘と経験だけでは適正な解を求めることは難しくなって来ている。そこで、IPネットワークにおける構成／経路に関する設計と検証に段階的詳細化と形式手法を導入し、効率的体系的に行うことを検討した。これは設計の後の実環境における設定や保守作業を含む統合的なネットワーク管理支援を可能にする。

IP network routing design and its verification

Masato TSURU[1], Hideo KURODA[2]
[1] Science Information Center, Nagasaki university
[2] Dept. of Electrical Engineering & Computer Science, Nagasaki university

Abstract: According to growth of its scale, LAN design and management become so complex that it's hard to get a proper solusion within intuition and experience. In this paper, we discuss how to use formal methods and stepwise refinements for effective and systematic network design and its verification. It's also help for an integrated network management support including configuration and maintenance phase on the real network environment following design phase.

1　はじめに

ネットワーク (LAN) の構築において、構成／経路の設計は多様な要素に依存する複雑な作業[†1]であるが、依然として勘と経験に頼って行われることが多く、それでは客観的で適正な解を求めること困難である。

また、お金と時間を持つところは、高度なネットワークシミュレータ [1] による詳細な事前検討を行うようになってきたが、その効果は、プロトタイプの設計やパラメタの設定に大きく依存し、その部分は担当者の技量次第である。

そこで、本報告では、IPネットワーク (特にLAN) の構成／経路に関する抽象度の高いモデル化を行い、

- 書換え規則を用いた段階的詳細化設計を行う。
- 形式手法を用いてモデル上で検証する。
- 実環境上での設定や検証 (テスト) さらにはその後の保守作業も支援する。

ことを目標に検討を行った。

一般に、LANの構築は、以下のような手順で進められると考えられる。ここで、2. と 3. は抽象モデル上で可能な話であり、できるだけ形式手法を用いて体系的に行いたい。また、4. 以降は実環境上の話であるが、2. と 3. に連動させて、設定や検証の (半) 自動化を目指したい。

1. 要求仕様
   ネットワークの利用形態とそのトラフィック特性の予測 (業務分析) し、その上での以下のような必要条件／制約条件を決める[†2]。
   - トラフィック性能 (帯域／レスポンス)
   - 冗長性 (障害時の迂回路等)
   - 物理的／地理的な配線の制約
   - コスト (初期コスト、運用コスト)

*tsuru@net.nagasaki-u.ac.jp
†kuroda@ec.nagasaki-u.ac.jp

†1 物理的配線上の設計も物理的、経済的、政治的な制約が多く、対人折衝を含む大仕事だが、ここでは対象外とする。

†2 実際は明確にならないものもある。

- セキュリティや管理上の分割
- 管理の手間 (監視、変更、障害時等)

2. ネットワーク構成／経路の設計と検証
   1. の必要／制約条件を満たすよう設計することが目的であるが、条件が必ずしも明確には与えられず、また、暗黙の制約や計画途中で認識／変更される制約なども多い。そこで、「設計を段階的に詳細化していく上での、選択や推論過程、条件の検証を支援でき、できあがった設計も形式的に扱える仕組み」を目指す。また、設計の検討の中で 1. の条件の見直し、追加が発生することも考えられる。
3. 経路制御の設計と検証
   各ノード上での経路制御をうまく設計（設定）して、2. で意図した経路になるようにする。モデル上の制御方式としては、各ノードのネットワークインタフェースに「コスト」を与え、そのコスト最短木が実際の経路になるようなものが代表的である。適切な制御 (設定値) を求めること自体が検証を含む。
4. 実ネットワーク構成 (トポロジ、物理媒体、ノード装置)
   2. のモデル上の設計を実際の「もの」にマッピングする。1. の必要／制約条件のうち、この時点ではじめて考慮されるものもあると思われる。また、利用可能な製品（コスト、納期、etc) の制約もあり、2. への手戻りは十分ある。
5. 実ノードの経路制御パラメタ
   3. のモデル上の設定を実際のノード（ルータ、サーバ）上の設定にマッピングする。
6. 4. や 5. の実検証（実環境上の観測による検証）
   本来は形式手法で設計／検証されたものはすでに正しいことが証明されているはずだが、本報告の例がそうであるように、現実のシステムのある側面だけを粗く近似して扱うので、そこに含まれない要素もある。また、暗黙の前提部分がバグ等で満たされない場合もある。よって、実環境での観測による検証は依然として必須である。
   この検証は、基本的には、2. のモデル上で計算される値と、実際の観測値の比較であり、判定のかなりの部分が自動化できる。
7. 安定運用 (ネットワーク管理)
   以上の過程を経て、安定運用に入る。6. の実検証は、設計のバグや障害の監視の意味でも継続的に行うことが必要である。その監視を含め、一般にネットワーク管理のための各種設定も、前項までの抽象モデル上のデータから抽出できるものが多いと思われる。

2 ネットワーク構成／経路の設計と検証

2.1 基本モデル

ここでは、「ユニキャストのクライアント／サーバモデル」を前提[†3]とした。すなわち、

- 分散したクライアントと少数のサーバ
- サーバ間及びサーバ／クライアント間通信が大部分（他を考慮しない）
- 各クライアント（またはサーバ) からサーバへ要求が送信され、サーバがそれに応答する通信が大部分（他を考慮しない）

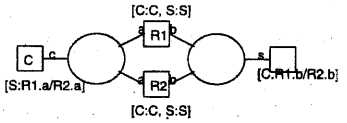
ネットワークを形式的に扱うには、例えば、静的なグラフ構造、動的なプロセス代数等で表現する方法がある。前者はグラフ理論／ OR の蓄積された解析手法を利用でき、また直感的に見易い。一方、後者は代数的／論理的な解析手法があり、特に、構成の動的変化も記述できる点が優れている。また実際に動作（シミュレート）できる。どちらも（基本的な方法では）実時間的／距離的概念は表現できない[†4]。そこで、この両方を併用し、構成要素は以下のようにする。

ノード	サーバ、クライアント、ルータ
インタフェース	ノードの出る送受信の口
ネットワーク	インタフェース間に介在し、つなぐもの
宛先	サーバまたはクライアント（のインタフェース）
経路表	各ノードが持つ、宛先毎の次中継先 (インタフェース)

このモデルでは経路を含めた構成を表現できる。また、ルータを介したネットワークの集まりをまたネットワークと見なせるので階層的である。

図 1 のグラフ表現で、各ノードの横で $[s : R_1.a/R_2.a]$ のように記述されるのが経路表であり、「サーバ s 宛の通信は、次は R_1のインタフェース a に転送するが、それができない場合は、R_2のインタフェース a に転送する」という代替経路を含めた意味を持つ。

[†3] そうでないモデルとしてはすべてのノード間通信が対等とかマルチキャストベースの通信とかがありうる。
[†4] プロセス代数においては実時間的／距離的概念を導入する拡張が研究されてきている。



$$C \stackrel{def}{=} in_s.(\underline{r_{1.a,s}}I_c.\underline{send_s} + D_{r1}.\underline{r_{2.a,s}}I_c.\underline{send_s}) \\ |send_s.c(x).(\underline{x}|I_s.\underline{out_s}.C)$$

$$S \stackrel{def}{=} s(x).(\underline{x}|I_c.\underline{recv_c}.S) \\ |recv_c.(\underline{r_{1.b,c}}I_s + D_{r1}.\underline{r_{2.b,c}}I_s)$$

$$R_1 \stackrel{def}{=} r_{1.a,s}(x).\underline{s}x.R_1|r_{1.b,c}(x).\underline{c}x.R_1$$

$$R_2 \stackrel{def}{=} r_{2.a,s}(x).\underline{s}x.R_2|r_{2.b,c}(x).\underline{c}x.R_2$$

図 1: グラフ表現とプロセス代数表現の例

一方、プロセス代数表現には、とりあえず単項π計算 [2] を使った。基本的には、並行動作 (|) する2つのプロセスの、名前$\underline{p}$の (送信) ポートと名前pの (受信) ポートの間で、同期または名前の受け渡しがアトミックに発生する。経路表は各ノードの記述に内在し、ネットワークは陽には見えない。

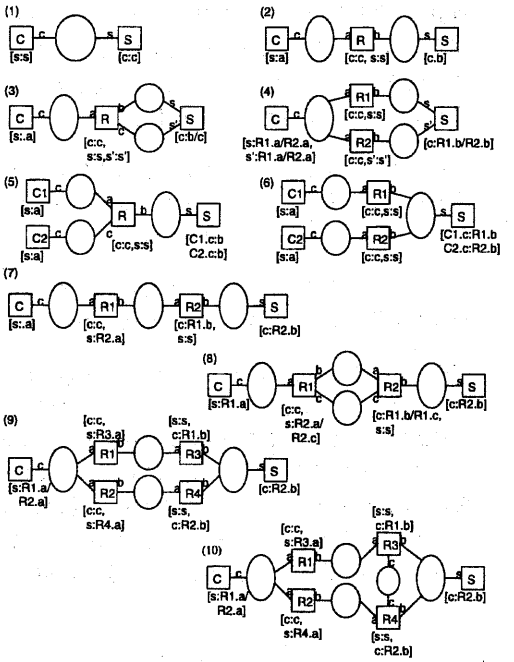
図1の例の場合、ネットワーク全体は$C|S|R_1|R_2$というプロセスで表現される。クライアント C が (例えば上位のアプリケーションから) サーバ S への送信要求in_sを受取ると動作 (状態遷移) が始まる。I_s, I_cはそれぞれ S や C の識別名であり、その識別名を送信することで受信側において誰から来たかがわかる。S は受信した要求に答えるがそのタイミングは不定である。C が S からの応答を受け取って、(上位のアプリケーション等の) 受信通知out_sを起こして一つのサイクルが終わる。C, S は送信プロセスと受信プロセスの並行動作 (|) として、R_1, R_2は各方向の中継プロセスの並行動作 (|) として、各々モデル化している。$r_{1.a,s}$は、R_1のaインターフェースでs宛の送信要求を受け取ることを意味するポートであり、D_{r1}はモデル外からの、R_1が死んでいることの通知を意味するポートである。

2.2 段階的詳細化設計

モデル上での設計において、必要十分な書換え (発展) 規則を用意し、それを使って基本的性質を満たす初期構成から出発して、追加される構成上の条件に合わせてこの書換えを行って段階的に詳細化設計を行いたい。

ルールは以下の3タイプに大別される。見易さのためにグラフ表現で示す。

1. 分散— 物理的／地理的制約やセキュリティや管理上の問題、あるいはクライアントやサーバの配置 (サーバの場合、負荷分散の意味もある) によってネットワークを分散／分割する。
2. 2重化— 冗長性 (障害時の迂回路) と負荷分散のためにネットワークの2重化やルータの2重化を行う。
3. インタフェースの集約／削除— 無駄な配線を整理統合する。



ネットワーク分散	(1)⇔(2), (2)⇔(7), (2)⇔(5)
ネットワーク2重化	(2)⇔(3), (7)⇔(8)
ルータ2重化	(3)⇔(4), (5)⇔(6), (8)⇔(9), (8)⇔(10)

図 2: 分散と2重化の書換え規則

図2のような分散や2重化は大域的な構造の複雑化である。

一方、インタフェースの集約／削除は、局所的な効率化である。図3の例以外にも考えられる。ただし、むやみに使うと設計の検証が困難になる。

これらのルールの適用場所と順序は一意ではな

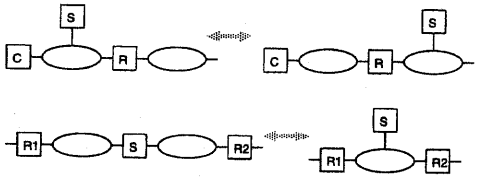


図 3: インタフェースの集約／削除の書換え規則

く、その適用が設計がうまくいくかどうかの分かれ目でもあるが、書換え毎に計算によって要求仕様を検証(次項参照)し、その書換えが妥当かはチェックできるので、効率的に試行錯誤ができる。また、基本的性質（条件）に関しては、その書換え時に保存されることが保証されるものもある。

このようなルールは、形式システム開発手法である refinement [3] と見ることもできるし、一種のデザインパターンと見ることもできる。

またルールの適用を試行錯誤して推論を進めるための "論証支援システム" との類似性もある。

そして、これらを実際に本学の LAN の設計に適用したところ、単純なネットワークから出発して、最終的には ATM+FDDI の 2 重バックボーン構成の全学 LAN を導出できた。図 4 はその中でも最も複雑な総合情報処理センター内の LAN の構成である。ただし、管理ネットワークの追加を行っている。(これも一つの汎用的な書換え規則と言えるが)

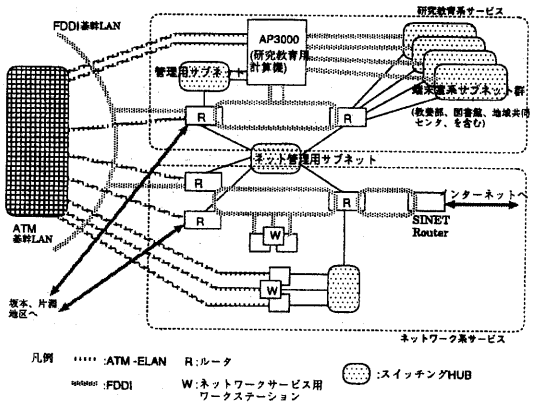


図 4: 複雑な LAN の導出例

2.3 抽象モデル上での検証

このモデル上で、各サーバ間及びサーバ／クライアント間通信の必要トラフィック、各インタフェースが処理できる許容トラフィックと通過時間と値段の選択肢、ルータが処理できる許容トラフィックと通過時間と値段の選択肢、などを与えると、要求仕様のいくつかは、モデル上の計算によってチェックできる。例えば、各主経路及び冗長経路に関して、経路にそった必要トラフィックと経路上の各インタフェース及びルータの許容トラフィックがわかるので、その構成が妥当かどうかが検証できる。

これは静的な構造で決まる計算なので機械的に扱えさえすればよく、プロセス代数表現である必然性はない。一方、プロセス代数表現に期待できることとして、以下のものがある。

- アクション（この場合はノード間の受け渡し）の発生系列に関する構造の各種「等価性」の解析手法を使い、2 つの設計の間の構造（性質）の保存を検証する。
- アクションの発生系列に関する様相「論理」の解析手法を使い、ある設計の性質を検証する。
- インタプリタ（シミュレータ）上で動作（状態遷移）させ、アクション発生を観察することで検証する。

3 経路制御の設計と検証

設計したネットワーク構成上の経路を実現するには、各ノード上での経路制御の適切な設定 (値) が必要である。抽象モデル上でどのような制御モデルを採用するかは、使用する経路制御プロトコルや各ノード装置での具体的な設定項目にマッピングできないといけないので、微妙な問題である。

ここでは経路制御プロトコルとして RIP や OSPF を想定した上で、基本的な制御モデルとして、

> 各ノードのネットワークインタフェースに数値「コスト」を与え、宛先毎のそのコスト最短木がパケットの経路になる。

を採用する。ただし、

1. モデルの制御能力 (経路の実現可能性)
   このような「インタフェースコストの最短木」モデル（だけ）では実現不可能な経路制御の例は簡単に見つかる。
2. 同一コスト (優先度) の経路

RIP の場合、RFC[4] に忠実に作ると同じ優先度の経路が2つあってもどちらか一方が選択される[†5]。一方、OSPF の場合は、RFC[5] 上も同じ優先度の経路がある場合などはトラフィック分散することが想定されている。

同一コストの経路の自動トラフィック分散は便利な機能ではあり、うまく使えば経路制御設計を簡単にする。しかし、必ずしも普及してない点と非決定性を形式的に扱うのが面倒な点から、今回のモデルでは、その機能は仮定せず、よってそういう同一コストの経路は作らないことを方針とした。

3. 経路の宛先 (ネットルート／ホストルート)

2章のモデルで、経路の宛先としてはサーバ (のインタフェース) やクライアントを指定した。これは、ホストルートにもマッピングできる。実際、マルチホームなサーバを置き、到達経路の複数化によるトラフィック分散と冗長 (可用) 性を実現したい場合、あまりスマートではないが、そのサーバ上 (の gated) から全インタフェースに対してホストルートを流す方法が考えられる。

これを使う場合、経路情報の import において自分が直接つながっているネットワークに属するホストルートは受け取らない方が安全である。

4. 各インタフェースへのコストの与え方

方式 A 送信コスト (RIP を受信した時に加算するメトリック) は自由に設定できる。受信コスト (RIP を送信する時に加算するメトリック) は 0 固定。

方式 B 送信コストと受信コストには同一値を自由に設定できる。

方式 C 送信コストと受信コストは、それぞれ独立に自由に設定できる。

のような選択肢があり、実現 (表現) 可能な経路制御の範囲が異なる。

以下ではもう少し細かく、

- インタフェースへのコストの与え方は方式 B。つまり、あまり送信したくないインタフェースからは受信も遠慮したい。
- 直接つながっているエンドノード (サーバ、クライアント) への送信コストは、その送信インタフェースのコストによらず 0 とする。
- インタフェースへ設定可能なコスト値は 1 以上 N 以下の整数とする。

の場合について検討する。

この制御モデルでは明らかに図 5 の経路制御は実現できない。なぜなら、R_1からも R_2からも、B と C への送信コストは 0 であり、A から見ると、B 行きも C 行きも、R_1 と R_2 のうち受信コストの小さい方を通ることになり、経路の分散ができない[†6]。

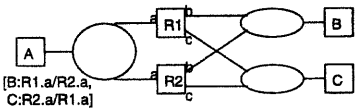


図 5: 制御不可能な経路構成

つまり、一般に、2章でのネットワーク構成／経路設計において、このような実現できない経路が導出されないようにする必要がある。

この制御モデルの上で、与えられたネットワーク経路を実現する「インタフェースコスト配置」を求めるには、最短木計算[†7] の逆問題を解くことになる。これは、連立線形不等式の整数計画問題と見なせるが、不等式の矛盾 (そもそもこのモデルでは実現できない経路) があると解けない。解がある場合は、いくつかの解法が知られているが、必ずしも効率的ではない。単純には、整数条件をはずして線形不等式問題として全コストの和を最小にするように解き、その解以上の整数値を当てはめて探すことができる。

解があるかどうか、すなわち「与えられた経路の、モデル上での制御 (実現) 可能性」については、2章の書換え規則の範囲での導出において[†8] は、保証できるのではないかと予想している。

4 実ネットワーク上での設定

1. ノード装置や物理媒体の選択

選択可能な製品情報が適切にデータベース化されていれば、モデル上の設計やデータから条件を満たす製品候補を一覧表示したりする支援ツールは容易であろう[†9]。

2. ノード装置の経路制御の設定

[†5] 実際は CISCO のルータ等は双方にトラフィック分散する。

[†6] A が同一コストの経路に対して自動で分散をしてくれるようなノードなら、あまり問題にはならないが。

[†7] ダイクストラ法などの効率的な解法が研究されている。

[†8] 必要ならもう少し制約条件を追加して。例えば初期値。

[†9] 今回は対象外である。

実ノード装置の経路制御パラメタに関して、モデル上の設定値から、製品に合わせた configuration ファイルや設定スクリプト (expect 等による) を自動生成できる。これによって、人手の設定による間違いの混入防止や、初期導入時の多数のルータへの設定作業の大幅効率化が可能になる。

今回は、富士通 LR ルータの RIP の設定に関して、expect スクリプトを試作した。これは、パスワードを入力すると telnet を起動し、設定を自動で行う。

5 実ネットワーク上の観測による検証

実検証に関して、今回は、以下のツールを試作中。

1. 各ノードの経路表の確認
   各ノード (ルータ及びサーバ) の IP 経路表を、SNMP(または telnet+expect) で定期的に取得し、2 章のモデル上での経路表と比較する。これにより、構成の間違い、設定の間違い、一時的障害、バグなどが検出できる。
2. 各インタフェースのトラフィックの確認
   各ノードの各ネットワークインタフェースの入力/出力トラフィックを、SNMP(または telnet+expect) で定期的に取得し、2 章のモデル上での経路にそったトラフィック予測 (仮定) から計算した各インタフェースのトラフィックと、(近似的に) 比較する。
   大きな違いがある場合、1. のような「間違い」の検出以外に、そもそものモデル (トラフィック予測) が適切でなかった可能性がある。
3. 経路及び応答時間の確認 (2 通りの方法)
   (a) 本学では主要なクライアントサブネット内には、スイッチング HUB が設置されているので、サーバ側からそれらのスイッチング HUB への ping and/or traceroute をかける。
   (b) すべてのクライアントサブネットから頻繁に WWW サーバへのアクセスがあるので、よくアクセスされるページに隠し機能をつけて、クライアントへの ping and/or traceroute をかける。

   1. で経路表が正しいことが確認されていれば普通は経路は正しい。ところが最近のルータは高速化のためキャッシュを使ったフォワードを行うので、この辺のバグで経路がおかしいことがある[†10]。また、レスポンスについてはモデル上に現れる値 (トラフィックやホップ数) から定性的に設計をチェックしただけであり、実際の応答時間が実用的に問題であれば、そもそものモデルが適切でなかった可能性がある。このような理由から、依然としてこの実経路にそった通信測定は必要である。

6 今後

LAN の構成/経路をモデル化し、そのモデル上で、段階的詳細化による設計や形式的な検証を試み、その方向性を得た。今後、技術的に精密化した検討を進める必要がある。

さらに、運用開始後に、"少しだけ構成や設定を変更した時に、どれだけのテストすれば十分だろうか" とか、"ネットワークの拡張や構成変更を行う時にどのような手順 (状態移行) で行うと影響が少なく手間も少ないだろうか" といった問題がある。形式化して取り扱うことはその第一歩ではあるが、今回の方向だけでは目処が立っていない。

参考文献

[1] 相沢りえ子. 離散型・連続型記述言語 slam ii. 情報処理, Vol. 37, No. 3, 1996.

[2] R. Milner. The polyadic π-calculus: a tutorial. *Report ECS-LFCS-91-180*, 1991.

[3] The RAISE Language Group. *The RAISE Specification Language*. Prentice Hall, 1992.

[4] C. Hedrick. Routing information protocol. *RFC 1058*, 1988.

[5] J. Moy. Ospf version 2. *RFC 1247*, 1991.

[†10] 本学でも某社のルータのこのバグに苦しめられた