第 IV 部

経路情報の解析および
次世代経路制御技術の検討

第 4 部
経路情報の解析および次世代経路制御技術の検討

第 1 章　はじめに

Routing ワーキンググループでは経路制御をテーマに、Routeview ワーキンググループを引き継ぎ発足したワーキンググループである。研究テーマとしては、経路情報の解析と次世代経路制御技術の検討であり、OSPF や BGP の経路情報の解析等を行なう。本ワーキンググループは他のワーキンググループとは異なり、合宿・研究会等で顔を合わせ、各自の研究を発表し、それを基に議論することを主な活動内容としている。

本年度は、本ワーキンググループで活動している慶應義塾大学　小原泰弘氏による経路制御シミュレーションソフトウェアの開発に関する成果を報告する。

第 2 章　ルーティングシミュレーションツール：simrouting

2.1 ツール作成の必要性

ルーティングアルゴリズムを評価することは以下の理由から困難である。評価結果はネットワークの構成や状態など、他の変数に大きく影響される。例えば、ルーティングアルゴリズムが目的とするのがネットワーク回線の帯域利用率の平滑化であるとき、結果となるネットワーク回線の帯域利用率は、ルーティングアルゴリズムの性質以外に、ネットワークトポロジ、ルーティングメトリック設定、各トラフィック負荷の量と方向など、さまざまな変数に影響される。その上、結果が目標を達成したとき、どの変数のどの性質がその主な理由であるのか断定することができない。例えば、あるネットワーク回線の帯域利用率が低いことは、ルーティングアルゴリズムの性質であるのか、ネットワークトポロジの性質であるのか、単にルーティングメトリック設定の結果であるのか、断定することができない。

ルーティングアルゴリズムの評価には、多くの変数が関わることから、以下の作業が必要となる。第 1 に、各変数を簡単に変更する機能である。たとえば、ネットワークトポロジのみを変更する、トラフィック負荷のみを変更するなど、各変数を簡単に詳細に変更する機能が必要となる。

第 2 に、比較が必要である。ルーティングアルゴリズムの評価をするためには、ネットワークトポロジ、トラフィック負荷などを固定し、他のルーティングアルゴリズムとの比較評価が必要である。

第 3 に、多くの変数の組み合わせに対して評価しなくてはならない。ルーティングアルゴリズムには、通常「このようなネットワークトポロジの場合に良好に動作する」などの特徴がある。このような特徴を考慮するためには、多くのネットワークトポロジとトラフィック負荷の組み合わせを調べるなどが必要である。

これらから、ルーティングアルゴリズムの評価には、多くの種類のネットワークトポロジを簡単に扱え、トラフィック負荷やルーティングメトリックを設定することが可能で、それらの組み合わせを簡単に調整できるシミュレーションツールが必要である。

2.2 simrouting の設計

simrouting の設計について述べる。

第 1 に、simrouting は多くの人が容易に理解し実行できるべきである。新しいトラフィックモデルを試したい、新しいルーティングアルゴリズムを作成しテストしたいなどのユーザが簡単に利用するために、可読性が高く、最も基本的な言語である C 言語で実現する。

第 2 に、シェルによるインタラクティブな設定が可能である必要がある。ディスタンスベクタの無限カウント問題などに見られるように、ルーティングシステムの不具合を発見するためには、時間的な推移を柔軟に追跡できる必要がある場合がある。各時点での仮想ネットワークの状態がどのようであるかをチェックしながら逐次的な実行が可能でないと、simrouting が目的とする「ルーティングシステムの

表 2.1. simrouting の独立内部データ構造の名称と説明

名前	説明
graph	グラフ構造を保持する。ネットワークトポロジを保持したり、特定の終点に対する経路を図示するために利用する。
weight	特定のネットワークトポロジに対するルーティングメトリックを保持する。
routing	graph と weight を利用して計算した経路を保持する。
traffic	特定のネットワークトポロジに対するトラフィックフローを保持する。
network	routing と traffic から、各リンクの帯域利用率などを計算する。

動作を入念に調査し特徴を示す」ことができない。これを解決するために、simrouting はシェルを装備し、シェルのコマンドによってシミュレーションを逐次的に実行する機能を持つ。

第 3 に、設定ファイルなどを利用したバッチ処理が可能である必要がある。まったく同じトポロジに対して、数百のトラフィック負荷の種類を定義し、それに対するシミュレーションを実行し結果を比較したいという要望がある。このとき、数百のトラフィック負荷をいちいち指定しながらシミュレーションを実行することは現実的でなく、一括処理したい。これを実現するため、simrouting ではシミュレータの実行コマンドをファイルに記述し、それを読み込みシミュレーションを実行する機能を提供する。一括処理は、トラフィック負荷の数百の種類ごとにシナリオファイル（設定ファイル）を用意し、それを引数としてシミュレータを起動することによって一括で処理できるようにした。

第 4 に、外部ツールとの連携が容易であるように、内部データ構造を抽象化する。ルーティングのシミュレーションは、様々なトポロジに対して実行する必要があり、そのようなトポロジは外部ツールによって提供されるものを利用する必要がある。また、ルーティングの問題は図示せねば理解できないようなグラフ問題を多く含むため、グラフの図示に関する外部ツールを利用する必要もある。これらを実現するため、simrouting では、グラフ構造（内部データ構造：graph）やトラフィックフロー（内部データ構造：traffic）を独立の対等のデータ構造として定義できるようにした。外部ツールからグラフ構造を読み込む部分は、その外部ツールと内部データ構造 graph を連結するドライバのようなものを追加すれば良く、外部ツールと連携が容易になる。

さまざまなネットワークモデルをグラフ構造として生成するツールには BRITE がある。また、ROCKETFUEL は実際のネットワークのグラフ構造を推測したデータを公開している。また、SNMP（Net-SNMP）を利用して実際のネットワークのグラフ構造を読み込むことからも、ネットワークのグラフ構造を取得できる。グラフ構造を図示する代表的なツールには、graphviz がある。

独立に対等な内部データ構造を定義する設計は、ルーティングアルゴリズムの違いを吸収するのにも役立つ。例えば、既存のルーティングアルゴリズムは cost という変数を自由に設定することによってルーティングメトリックを変更できる。しかし、新しいルーティングアルゴリズムにはこのような cost を用いないものも存在する可能性がある。これらを比較する場合には、cost を weight という別の内部データ構造として定義し、それぞれのルーティングアルゴリズムを実行する際に、（必要であれば）この weight を参照する。ルーティングアルゴリズムの結果は、経路セットとして routing というまた別の独立な内部データ構造として保持される。これにより、cost を用いるかどうかというルーティングアルゴリズムの特徴を隠蔽し、ルーティングアルゴリズムの実行結果（経路セット）を統一的に評価できる。また、ルーティングメトリックを固定し複数のルーティングアルゴリズムを比較するなども容易に実現できる。

simrouting が実現する主要な独立内部データ構造の種類とその説明を、表 2.1 に示す。

2.3 実装

simrouting は autoconf を利用しており、ポータブルに実装した。Linux FC4、FreeBSD-5.5 での動作を確認している。ヘッダファイルを含めて 60 ほどのファイルで実装された 12000 行ほどの C 言語のソフトウェアである。

実現した機能を以下に述べる。

2.3.1 ルーティングシミュレーション

ネットワークトポロジのグラフ構造を、graph に保持できる。ネットワークトポロジの読み込みには、以下の方法がある。

- シェルコマンドによるノードとリンクの手動定義
- snmp を使って実際のルータから OSPF LSDB を読み込む
- BRITE によって生成したものを読み込む

ルーティングメトリックは、ネットワークトポロジを保持する graph を指定し、以下の方法によって weight に設定できる。

- シェルコマンドによる手動設定
- 全て 1 に設定する（minimum-hop）
- リンク帯域に反比例するように設定する（inverse-capacity）

ネットワークのグラフ構造を持つ graph と、（必要であれば）weight を利用して、以下の方法で経路を計算できる。結果の経路は routing に保持される。

- dijkstra による最短経路計算。weight を指定する必要がある。
- LBRA（本ワーキンググループで開発している新しいルーティングアルゴリズム）。

ネットワークトポロジを指定し、その各ノードを始点終点とする、任意のトラフィックフローが定義できる。この定義方法には以下がある。定義されたトラフィックフローは traffic に保持される。

- 明示的に指定する手動設定
- 始点と終点をランダムに選ぶ手動設定
- fortz-thorup モデルのトラフィック生成

routing と traffic を利用すると、その経路設定とトラフィックフローでは各リンクがどれだけ帯域を消費するかが計算できる。これは、network に保持される。

network では、graph と routing と traffic を指定し、'network-load traffic-flows' コマンドを呼び出すと、各トラフィックフローが経路に従って（仮想的に）転送され、各ノードやリンクの消費帯域が計算される。計算途中や結果は、printf 出力される。

BRITE を利用してネットワークトポロジを生成し、その上で inverse-capacity のコストを使った dijkstra と新しいルーティングアルゴリズム LBRA を比較するシナリオを以下に示す。トラフィックは fortz-thorup モデルを利用している。

```
graph 100
 import brite etc/RTWaxman20CONS.brite
exit
show graph 100 structure

weight 100
 weight-graph 100
 weight-setting inverse-capacity
exit
show weight 100

routing 100
 routing-graph 100
 routing-weight 100
 routing-algorithm dijkstra
exit
show routing 100

routing 200
 routing-graph 100
 routing-algorithm lbra
exit
show routing 200

traffic 100
 traffic-graph 100
 traffic-seed 30
 traffic-model fortz-thorup alpha 100.0
exit
show traffic 100

network 100
 network-graph 100
 network-traffic 100
 network-routing 100
 network-load traffic-flows
exit
show network 100

network 200
 network-graph 100
 network-traffic 100
 network-routing 200
 network-load traffic-flows
exit
show network 200
```

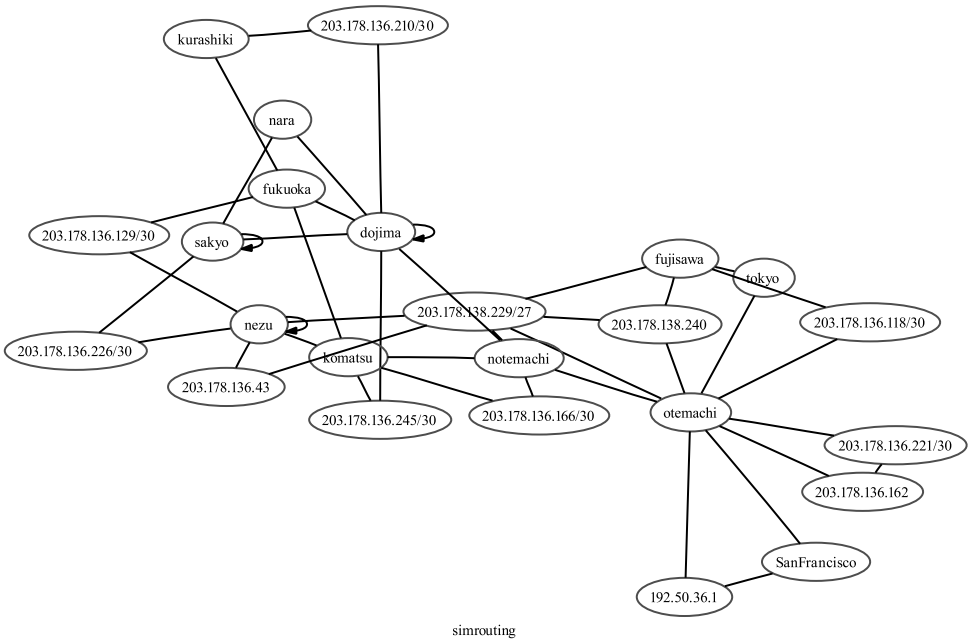

図 2.1.　WIDE プロジェクトネットワークトポロジ（2006/12/24）

2.3.2 ネットワークに関するグラフ生成

graph に定義したものは、すべて graphviz によって図示できる。graph には、前述のネットワークトポロジを保持するほか、routing に保持される経路のうち、特定の終点に対する経路を抜き出し有向グラフとして保持することもできる。これを graphviz によって図示することによって、ネットワーク経路を図示することができる。

snmp により実際のルータの OSPF LSDB からネットワークトポロジを取得し、それを図示すると、実際のネットワークトポロジを簡単に表示できる。しかし、実際のネットワークトポロジは複雑であり、また IP アドレスやルータ ID からではネットワークの全体像をつかむことが難しい。そこで、以下の二つの機能を実装した。

- ホスト名を解決する機能
- リンクを縮約し、複数のノードをグループ化して一つの抽象ノードとする

これらの二つの機能は組み合わせて使用できる。つまり、ホスト名を解決し、そのホスト名の条件によってグループ化するノードを決めることができる。

WIDE プロジェクトのネットワークを上記の方法で図示する際の設定ファイルを 2.4.3 に示す。結果として出力される graphviz の DOT ファイルを graphviz を使ってイメージ化したものを図 2.1 に示す。

2.3.3 データ構造とアルゴリズムのライブラリ化

新しいルーティングアルゴリズムを開発し、このツールでチェックするためには、このツールの内部に C 言語でそのアルゴリズムを実装しなければならない。

これを容易にするために、一般的データ構造とシェル部分をライブラリ化した。

- lib/vector.[ch]: 可変長配列
- lib/table.[ch]: パトリシアテーブル
- lib/pqueue.[ch]: プライオリティキュー
- lib/command.[ch], lib/shell.[ch], lib/command_shell.[ch]: シェル

```
yasu@plant Sun 12/24 15:39[~/simrouting](0)> ls
AUTHORS      Makefile.in     config.h.in  etc/        interface/   traffic/
COPYING      NEWS            configure*   function/   lib/
ChangeLog    README          configure.ac graph/      missing*
INSTALL      aclocal.m4      data/        includes.h  routing/
Makefile.am  autom4te.cache/ depcomp*     install-sh* simrouting.c
yasu@plant Sun 12/24 15:39[~/simrouting](0)> ls lib/
Makefile.am  command.h       log.c     pqueue.h  shell.c  table.h   vector.c
Makefile.in  command_shell.c log.h     random.c  shell.h  termio.c  vector.h
command.c    command_shell.h pqueue.c  random.h  table.c  termio.h
yasu@plant Sun 12/24 15:40[~/simrouting](0)>
```

2.4 使い方・例

2.4.1 MinHop v.s. InvCap dijkstra

以下は、シミュレーションのシナリオである。ここでは、BRITE からネットワークトポロジを読み込み、minimum-hop（すべてのリンクコストが 1）

である場合と、inverse-capacity（リンクコストはBRITEが生成した帯域の逆数）である場合についてそれぞれ経路を計算し、それらの経路を使ってfortz-thorupモデルのランダムトラフィックを仮想的に転送する。最後の‘show network’コマンドで、標準出力にそれぞれの結果のリンクの帯域利用率が出力される。

```
graph 100
 import brite etc/topology/sample.brite
exit
show graph 100 structure

weight 100
 weight-graph 100
 weight-setting minimum-hop
exit
show weight 100

routing 100
 routing-graph 100
 routing-weight 100
 routing-algorithm dijkstra
exit
show routing 100

weight 200
 weight-graph 100
 weight-setting inverse-capacity
exit
show weight 200

routing 200
 routing-graph 100
 routing-weight 200
 routing-algorithm dijkstra
exit
show routing 200

traffic 100
 traffic-graph 100
 traffic-seed 30
 traffic-model fortz-thorup alpha 10.0
exit
show traffic 100

network 100
 network-graph 100
 network-traffic 100
 network-routing 100
 network-load traffic-flows
exit
show network 100

network 200
 network-graph 100
 network-traffic 100
 network-routing 200
 network-load traffic-flows
exit
show network 200
```

2.4.2 InvCap dijkstra v.s. LBRA

次のシナリオファイルは、inverse-capacity のdijkstraによる最短経路と、LBRAという新しいルーティングアルゴリズムを比較するためのシナリオファイルである。

```
graph 100
 import brite etc/RTWaxman20CONS.brite
exit
show graph 100 structure

weight 100
 weight-graph 100
 weight-setting inverse-capacity
exit
show weight 100

routing 100
 routing-graph 100
 routing-weight 100
 routing-algorithm dijkstra
exit
show routing 100

routing 200
 routing-graph 100
```

```
 routing-algorithm lbra
exit
show routing 200

traffic 100
 traffic-graph 100
 traffic-seed 30
 traffic-model fortz-thorup alpha 100.0
exit
show traffic 100

network 100
 network-graph 100
 network-traffic 100
 network-routing 100
 network-load traffic-flows
exit
show network 100

network 200
 network-graph 100
 network-traffic 100
 network-routing 200
 network-load traffic-flows
exit
show network 200
```

2.4.3 ネットワークトポロジ図自動生成

次のシナリオファイルは、SNMP を使って OSPF LSDB を実際のルータから取得し、これをホスト名ごとにグループ化して、graphviz で表示するためのシナリオファイルである。図 2.1 は実際にこのシナリオを利用して生成された。

```
group fujisawa 10 match domain-suffix .fujisawa.wide.ad.jp
group otemachi 10 match domain-suffix .otemachi.wide.ad.jp
group notemachi 10 match domain-suffix .notemachi.wide.ad.jp
group tokyo 10 match domain-suffix .tokyo.wide.ad.jp
group yagami 10 match domain-suffix .yagami.wide.ad.jp
group osaka 10 match domain-suffix .osaka.wide.ad.jp
group nezu 10 match domain-suffix .nezu.wide.ad.jp
group dojima 10 match domain-suffix .dojima.wide.ad.jp
group sakyo 10 match domain-suffix .sakyo.wide.ad.jp
group kurashiki 10 match domain-suffix .kurashiki.wide.ad.jp
group fukuoka 10 match domain-suffix .fukuoka.wide.ad.jp
group komatsu 10 match domain-suffix .komatsu.wide.ad.jp
group SanFrancisco 10 match domain-suffix .SanFrancisco.wide.ad.jp
group nara 10 match domain-suffix .nara.wide.ad.jp

graph 100
 import ospf localhost public 0.0.0.0
 export graphviz tmp/wide-ospf-net-%Y%m%d-%H%M%S-raw.dot
exit

graph 200
 import graph 100
 contract edges both node group fujisawa name fujisawa
 contract edges both node group otemachi name otemachi
 contract edges both node group notemachi name notemachi
 contract edges both node group tokyo name tokyo
 contract edges both node group yagami name yagami
 contract edges both node group osaka name osaka
 contract edges both node group nezu name nezu
 contract edges both node group dojima name dojima
 contract edges both node group sakyo name sakyo
 contract edges both node group kurashiki name kurashiki
 contract edges both node group fukuoka name fukuoka
 contract edges both node group komatsu name komatsu
 contract edges both node group SanFrancisco name SanFrancisco
 contract edges both node group nara name nara
 remove stub-node
 remove stub-node
 realloc identifiers
 save graph config
 export graphviz tmp/wide-ospf-net-%Y%m%d-%H%M%S-noc.dot
exit

write config etc/test.config
exit
```

2.5 今後の予定

simrouting は 2007 年 4 月ごろ GPL ライセンスで公開する予定である。

2.6 Copyright Notice

Copyright (C) Yasuhiro Ohara (2007). All Rights Reserved.

第 3 章　まとめ

本年度、本ワーキンググループでは合宿、研究会を通し、次世代の経路制御技術に関して広く議論を行なった。その中で、慶應義塾大学　小原泰弘氏が作成したルーティングシミュレーションツールを本ワーキンググループの成果として報告した。