ラベルスイッチルータ におけるOSPF パケットフローアグリゲーションと優先転送サービス体系に関する研究

慶應義塾大学総合政策学部
今泉英明

指導教員
慶應義塾大学環境情報学部
村井純

平成 11 年 2 月 23 日

概 要

本研究ではラベルスイッチルータ (LSR) における優先転送サービス体系について議論する。

ラベルスイッチ技術は、ネットワーク層パケットストリームをデータリンク層のみで転送処理し、ルータで高速にスイッチング処理を行う技術である。データリンク層の固定長ラベルを各パケットに付加することにより、LSR によって構成されるネットワーク (LSR 網) 内では、ネットワーク層での中継ではなく、そのラベルを基にデータリンク層での高速なスイッチングが利用可能となる。また、ラベルに対して異なる優先度を持たせることによって、各パケットに対して、異なる転送処理をすることも可能である。

現在ラベルスイッチ技術は標準化に向けた議論が IETF MPLS 分科会で行われている。

LSR において優先転送サービス体系を実現するためには、優先度を持つラベルを利用する方法があるが、利用可能なラベル数が、インターネット上で交換されている通信の数に対して十分ではない。

本研究では、内部経路制御プロトコルである OSPF(Open Shortext Path First) 内の情報を能動的に用いたパケットフローアグリゲーションを実現することで、必要ラベル数を大幅に削減し、ラベル数の問題を解決した。このパケットフローアグリゲーションによって、従来の方式で必要となるラベル数の 56%を削減することができた。

また、Edge LSR においてトラフィック制御することで、LSR 網内での優先度を反映できることが分かった。

以上を踏まえラベルスイッチ技術における優先転送サービス体系の実現について議論する。

Abstract

MPLS technology performs a new kind of routers which can achieve high performance, scalabiliby and functionality on IP networks. MPLS uses short and fixed Layer 2 label and is currently in the standardization process in the IETF MPLS(Multi-Protocol Label Switching) WG. To achieve these initiatives, the mechanism of allocation labels of Layer 2 is very important.

This paper discusses the mechanism for Differentiated Service by using MultiProtocol Label Switching(MPLS) technology. The required number of labels is very important for each label to have different QoS infomation because IP packets with same label recieve same treatment on LSR. This paper performed the mechanism which can reduce the required number of labels by using the Link State infomation of OSPF protocol. The flow aggregation using OSPF Link State information has reduced 56% labels than that of the conventional traffic-driven/host-pair label mapping policy.

Furthermore, this paper discusses Differentiated Service on MPLS network especially functions of the boundary LSR for achieving Differentiated Service.

目 次

第 1 章　序論 **1**

1.1　はじめに 2

1.2　本論文の目的 3

1.3　本論文の構成 4

第 2 章　ラベルスイッチングテクノロジー **5**

2.1　MPLS(MultiProtocol Label Switching) 6

2.1.1　ラベルスイッチルータの機構 6

2.1.2　ラベル割り当てトリガ 8

2.1.3　ラベル割り当ての粒度 9

2.1.4　ラベル割り当て方針 9

2.2　Cell Switch Router 10

2.2.1　Cell Switch Router(CSR) の概要 10

2.2.2　CSR の特徴 10

第 3 章　ラベルスイッチルータの現状と問題点 **11**

3.1　ラベル数の問題 12

3.1.1　利用可能なラベル数 12

3.1.2　現状の LDP で利用できるパケットストリームの種類 12

3.1.3　パケットフローアグリゲーションの意味 12

3.1.4　さらなるパケットフローアグリゲーションの必要性 13

3.2　優先転送サービス体系 14

3.2.1　現状の優先処理 (Core LSR) 14

3.2.2　現状の優先処理 (Edge LSR) 15

3.2.3　現状の問題点のまとめ 15

第 4 章　OSPF リンクステートデータベース情報を用いたパケットフローアグリゲーション **17**

4.1　最適なパケットフローアグリゲーションと必要となる情報 18

4.2　Open Shortest Path First(OSPF) の概要 20

4.2.1　OSPF パケットの種類 20

4.2.2　LSA の種類 22

4.2.3　最短経路の計算 26

4.3　OSPF リンクステートデータベース情報を用いたパケットフローアグリゲーションの設計 27

4.3.1　概要 27

4.3.2　必要となる機能 27

4.3.3　実現する機構 29

4.4　実装 30

4.5 評価 . . . 36
4.5.1 実験内容と評価方法 . . . 36
4.5.2 実験結果 . . . 37
4.5.3 実験の評価 . . . 38

第 5 章 ラベルスイッチルータにおける優先転送サービス体系 40
5.1 優先転送サービス体系の概要 . . . 41
5.1.1 DS フィールド (Differentiated Service Field) . . . 41
5.1.2 優先転送サービス体系の機構 . . . 42
5.2 実装 . . . 43
5.3 実験と評価 . . . 44
5.3.1 実験の目的 . . . 44
5.3.2 実験に用いた機構 . . . 44
5.3.3 実験環境 . . . 45
5.3.4 実験 1 : CBQ のオーバーヘッドの測定 . . . 46
5.3.5 実験 2 : 送信元ホストによるクラス分け . . . 47
5.3.6 実験 3 : 特性の異なるトラフィック間でのリンク共有 . . . 47
5.3.7 実験 4 : 階層的なリンク共有 . . . 49

第 6 章 結論 53
6.1 結論 . . . 54
6.1.1 OSPF リンクステートデータベースを用いたパケットフローアグリゲーション . . . 54
6.1.2 優先転送サービス体系 . . . 54
6.2 今後の課題 . . . 54
6.2.1 Opaque LSA による Edge LSR の検出 . . . 54
6.2.2 エンドエンド通信の評価 . . . 56

第 7 章 謝辞 58

第 8 章 Appendix 62

図 目 次

2.1 ネットワーク層転送 . . . 7
2.2 LDP による通知とラベルスイッチ VC の設定 . . . 7
2.3 Label Switched Path . . . 7

3.1 LSR ネットワークトポロジ例 . . . 13
3.2 現状のパケットフローアグリゲーションされた LSP . . . 14
3.3 現状の Core LSR での優先処理 . . . 15
3.4 優先度処理が行われる範囲 . . . 16

4.1 最適なパケットフローアグリゲーション . . . 18
4.2 リンクステート型経路制御プロトコルを利用したネットワーク例 . . . 19
4.3 リンクステート型経路制御プロトコル内に持つトポロジー情報 . . . 19
4.4 OSPF Common Packet Header . . . 20
4.5 Link State Update Packet . . . 22
4.6 LSA Header . . . 22
4.7 Router LSA . . . 23
4.8 Link Infomation . . . 24
4.9 Network LSA . . . 25
4.10 他の経路のある LSR ネットワーク例 . . . 28
4.11 LSR を用いたネットワーク例 2 . . . 29
4.12 Edge LSR A の Router LSA . . . 30
4.13 最短経路ツリー 1 . . . 30
4.14 Core LSR の Router LSA . . . 31
4.15 NMBA Network の Network LSA . . . 31
4.16 最短経路ツリー 2 . . . 32
4.17 最短経路ツリー 3 . . . 32
4.18 パケットフローアグリゲーションのための設定例 . . . 33
4.19 CSR 内の構造 . . . 33
4.20 FANPD への ADD メッセージ . . . 34
4.21 FANPD への DEL メッセージ . . . 34
4.22 終点 CSR の検出と終点 CSR の継承に関するコード . . . 35
4.23 合宿ネットワークトポロジ . . . 36
4.24 実験 1 での総 LSP 数の推移 . . . 37
4.25 実験 2 での総 LSP 数の推移 . . . 38

5.1 優先転送サービス体系ネットワーク . . . 42
5.2 DS フィールドのフォーマット . . . 42

5.3 DS 境界ルータでのパケット処理 43
5.4 CBQ を用いたリンク共有の例 45
5.5 CBQ を用いた階層的なリンク共有の例 45
5.6 実験トポロジ 46
5.7 実験 1 でのリンク共有 46
5.8 実験 2 でのリンク共有 47
5.9 実験 2 の CBQ におけるスループットの測定 48
5.10 キューの様子 49
5.11 実験 4 でのリンク共有 50
5.12 トラフィック発生パターン 50
5.13 階層的なクラスを持った CBQ の様子 51

6.1 Opaque LSA 55
6.2 エンドエンド通信の優先度評価 56
6.3 異なる優先度の LSP を設定する例 56
6.4 他のトラフィックが影響する例 57

8.1 Hello Packet 63
8.2 Database Description Packet 64
8.3 Link State Request Packet 64
8.4 Link State Acknowledgement Packet 65
8.5 Summary LSA(inter-area) & Summary LSA(AS Border Router) 65
8.6 AS External LSA 66

表 目 次

2.1 CSR で設定されるトリガパケットフローの定義例 10

3.1 トラフィックドリブンで利用できるパケットストリームの種類 12
3.2 トポロジードリブンで利用できるパケットストリームの種類 12
3.3 ATM スイッチの設定 15

4.1 OSPF Packet Type 21
4.2 LSA Type 23
4.3 Link Type の種類 24
4.4 Link Type 別の Link ID、Link Data の持つ値 25
4.5 その他の LSA のフィールド 26
4.6 想定されたネットワークでの総 LSP 数 38
4.7 パケットストリームの種類別の比較 38

5.1 AF PHB グループの DS コード 43
5.2 図 5.4 の設定 45
5.3 実験 1 での CBQ の設定 47
5.4 CBQ を用いるオーバーヘッド 47
5.5 実験 2 での CBQ の設定 48
5.6 送信元ホスト別 TCP のスループット 48
5.7 キュー毎の TCP スループット 49
5.8 実験 4 での CBQ の設定 50

6.1 Opaque LSA の種類と意味 55

第1章

序論

1.1 はじめに

ユーザの要求の多様化や、それに伴うアプリケーションの多様化、特にリアルタイム・アプリケーションの出現により、現在のインターネットが提供している唯一のサービスであるベスト・エフォート型サービスでは、全ての要求を満たすことが限界になってきている。

リアルタイム性を求めるビデオストリームや多少の遅延を許容するファイル転送など、性質の異なるトラフィックの通信特性を活かした転送処理が必要である。

そのような要求に応えるために、ルータには以下のような機能が求められる。

- 高速パケット転送
- トラフィック制御
- QoS(Quority of Service) 保証

ラベルスイッチ技術はこのような要求を満たす技術として開発・標準化作業が推進されている。

また一方では、ネットワーク層でのトラフィック制御だけを用いて、配送保証の実現や、パケットの種類によって異なる転送処理を行う機構が標準化に向け、現在 IETF DiffServ 分科会で議論されている。

ラベルスイッチ技術はネットワーク層パケットストリームをデータリンク層の固定長ラベルに対応付けすることによって、積極的にデータリンク層の機能を利用する。そのため LSR 網を構成している中継ノードである LSR では、従来のネットワーク層での IP 処理を省略ために、高速なパケット転送が可能となり、また、データリンク層の技術に ATM や Frame Relay などのコネクション型リンクを利用することで、排他的な帯域資源制御や QoS 制御を可能とする。

LSR の実装例として、

- CSR (Cell Switch Router)[2]
- TSR (Tag Switch Router)[6]
- IP Switch[8]
- ARIS(Aggregate route-based IP switching)[10]

が提案されている。

本論文では以後、あるホストからあるホストへ流れる一方向のパケットの集合の中で、同一のネットワーク層ヘッダとトランスポート層ヘッダを持つパケットの集合をパケットフローと呼び、共通の性質を持ったパケットフローの集合をパケットストリームと呼ぶ。

LSR では、1 つのパケットストリームに対して、1 つのラベルが必要とされる。つまりパケットストリームはひとつのラベルを割り当てられる単位になるため、パケットストリームの決定方針は必要ラベル数に対して大きな影響を与える。例えば、パケットストリームを一つのパケットフローとした場合、必要ラベル数はその LSR を通過するパケットフローの数となり、大規模なネットワークのバックボーンルータでは膨大な数になる。一方、パケットストリームを同一宛先ネットワークアドレスであるパケットフローの集合とした場合は、それよりも必要ラベル数を減少させることができる。

また、LSR を利用するためには、このパケットストリームとデータリンク層の利用するラベルの対応付けを行うプロトコルが必要である。これは、IETF(Internet Engineering Task Force) の MPLS(Multi Protocol Label Switching) 分科会で議論されている。このプロトコルは、LDP(Label Distribution Protocol)[11] と呼ばれている。

ラベルを割り当てるタイミングをラベル割り当てトリガと呼び、トラフィックドリブンとトポロジードリブンの二つがある。トラフィックドリブンは、ラベルを割り当てられるべきパケットストリームの

パケットが LSR を通過した時点で、そのパケットストリームのためのラベルを割り当て、それ以降の同じパケットストリームに属するパケットはデータリンク層で転送させる方式である。ラベルを割り当てたパケットストリームに属するパケットが、一定時間流れない場合は、そのラベルは解放される。一方トポロジードリブンでは、経路表などの情報を基に、あらかじめラベルを割り当てる方式である。

このラベル割り当てトリガと、ラベルを割り当てる単位であるパケットストリームの決定方針によって、ラベル割り当て方針が構成される。

現在、主に 2 つのラベル割り当て方針が提案されている。一つは、IFMP(Ipsilon Flow Management Protocol)[9] と FANP(Flow Attribute Notification Protocol) [3] で定義されているもので、パケットストリームは送信元アドレスと宛先アドレスが同一のパケットフローの集合とし、ラベル割り当てトリガとしてトラフィックドリブンを利用する。もう一方のラベル割り当て方針は、TDP(Tag Distribution Protocol)[7] で用いられている。これは、パケットストリームを宛先ネットワークアドレスの同一なパケットフローの集合にし、ラベル割り当てトリガとしてトポロジードリブン を利用する。このラベル割り当て方針を利用すると、経路表と同じだけのラベル数が必要になる。

以上にラベルスイッチ技術の概要を説明したが、ラベルスイッチ技術の大きな問題点として、ラベルスイッチ技術が実現する機構が利用できるのは、LSR 網内のみであることが挙げられる。従って LSR 網に直接接続していないホスト間の通信に対して、配送保証をすることはできないため、IETF DiffServ 分科会で議論されている優先転送サービス体系のような機構を、ラベルスイッチ技術の特徴を生かし、適用する必要がある。

優先転送サービス体系は、大規模なネットワークでも適用可能な、パケット毎の異なるサービスを実現する機構である。優先転送サービス体系を適用し、同じ管理方針で運用されるネットワークを DS ドメインと呼び、そのネットワークの境界のルータによって、各パケットはそれぞれクラス分けされ、パケットにマーク される。DS ドメイン内部のルータは、パケットのマークによって異なる転送処理をするため、複雑なクラス分けなどによる負荷を軽減している。つまり、優先転送サービス体系の機構は、境界のルータに複雑な処理をさせて、内部のルータにはなるべく負荷を増加させない方針によって実現される。

数段階の優先度を実現するために、優先転送サービス体系ではパケットのキューイングやパケットスケジューリングなどによって実現するが、ラベルスイッチ技術ではデータリンク層が提供する優先度を利用する方法がある。この場合はラベルに優先度情報を持たせることで、パケットストリーム毎に異なる優先度を実現する。

しかし、ラベルに対して優先度を持たせる場合には、必要ラベル数の増加という問題が発生する。必要なラベル数は、ラベル割り当てトリガと、ラベルを割り当てるパケットストリームの決定方針によって決まる。ラベル付けされたパケットストリームに属するそれぞれパケットは、ラベルのみを参照されて転送されるために、同一パケットストリームに属するのパケット間では異なる優先度を持たせることはできない。従って異なる優先度を実現するには、優先度の異なるラベルを割り当てる必要がある。

永見健一は、LAN、WAN 環境における、優先度を利用しない場合に必要なラベルを評価したが、特に WAN 環境においては、非常に多くのラベル数が必要であることが分かった。[1] つまり数段階の優先度を実現するには、その数倍のラベルが必要になる。

必要なラベル数を減らすことのできるパケットストリームの決定方針を実現する (パケットフローアグリゲーションする) 必要がある。

1.2 本論文の目的

本論文では、LSR 網における優先転送サービス体系を実現する手法について議論する。

1. OSPF のリンクステートデータベース (LSDB) の情報を用いて、必要ラベル数の大幅な削減を実現するための手法を提案した。
2. この手法の評価を行うために、LSR の技術として CSR を用い、1 を実装・評価した。
3. LSR 網における優先転送サービス体系について議論し、特に境界 LSR に必要な機能を明らかにした。またこの機能の有用性を評価するために、実装・評価した。

1.3 本論文の構成

第 2 章ではラベルスイッチ技術全般にわたる概要、第 3 章ではラベルスイッチ技術の問題点を示す。それらをふまえて、第 4 章では OSPF の概要とそれを利用したパケットフローアグリゲーションの手法について論じる。またこの手法の実装、評価をし、それに対する考察をする。第 5 章では優先転送サービス体系の概要とラベルスイッチ技術に必要となる機能について論じる。また境界 LSR に対して必要となる機能について実験し、評価した。第 6 章では、本論文での結論とこれからの研究課題について論じる。

第2章

ラベルスイッチングテクノロジー

2.1 MPLS(MultiProtocol Label Switching)

2.1.1 ラベルスイッチルータの機構

ラベルスイッチルータ (LSR) で構成されるネットワーク (LSR 網) 内の LSR は、接続するルータによって、以下の 2 種類に区分される。

- Core LSR:
  - LSR 網内の LSR
  - ネットワーク層転送モジュール (図 2.1, Layer3 Forwarder) とデータリンク層転送モジュール (図 2.1:Layer2 Forwarder) で構成される。
- Edge LSR
  - LSR 網と他のネットワークとの境界の LSR
  - Core LSR と接続可能なネットワークインターフェース (NIC) を持つネットワーク層転送モジュール (後述の LDP が実装されていなければならない)

ネットワーク層転送モジュールは、通常のルータと同様にネットワーク層のパケットのヘッダを参照してパケットを転送し、データリンク層転送モジュールは、パケットのヘッダを参照せずに、データリンク層の固定長ラベル (ATM の場合 VPI/VCI 値) によってパケット転送する。ネットワーク層転送では、ネットワーク層ヘッダの宛先アドレスをキーとして、経路表を検索し送出インターフェースをソフトウェアで検索するが、データリンク層転送では、固定長ラベルをハードウェアで検索が行うことができるため、低遅延でのパケット転送が可能になる。

通常のルータと同様のネットワーク層転送は、図 2.1 のデフォルト VC を利用して転送される。図 2.1 の Edge LSR1 は受け取った Edge LSR2 宛のパケットの宛先アドレスをキーにルーティングテーブルを検索し、Core LSR に転送する。この時、Edge LSR1 はデフォルト VC を通して Core LSR のネットワーク層転送モジュールに転送する。Core LSR のネットワーク層転送モジュールも同様にルーティングテーブルを検索し、デフォルト VC を通して、Edge LSR2 へ転送する。このようにそれぞれの LSR でネットワーク層処理をして転送することをホップホップ転送と呼ぶ。

次に LSR の特徴であるデータリンク層転送の場合を説明する。まず、パケットを転送する前に、あるラベルに流すパケットストリームを LDP(Label Distribution Protocol) を用いて隣接ノード間で通知する。例えば、図 2.2 では、Edge LSR1 と Core LSR 間、Core LSR と Edge LSR2 間で LDP を用いて、Edge LSR2 宛のパケットストリームを流すことを通知し、Edge LSR1 と Core LSR 間、Core LSR と Edge LSR2 間に新たなラベルスイッチ VC を設立している。Core LSR はこの前後の 2 つのラベルスイッチ VC が設立されたことを確認し、データリンク層転送モジュールを、両者をデータリンク層で転送できるように設定し、図 2.3 の様に 2 つのラベルスイッチ VC がデータリンク層転送モジュールで接続される。

上記の通知が終了した後、Edge LSR1 は Edge LSR2 宛のパケットを、通知してあるラベルを持つラベルスイッチ VC を通して、Core LSR のデータリンク層転送モジュールに送られる。Core LSR のデータリンク層転送モジュールでは、受信したパケットのラベルを検索し、先程登録されたラベルを取得、それをもとに Edge LSR2 へのラベルスイッチ VC にパケットを転送する。これによってネットワーク層転送処理を行うことなく、パケット転送をすることができる。図 2.3 に示すようなラベルスイッチ VC を通して、データリンク層転送処理のみでパケットが転送される経路を LSP(Label Switchied Path) と呼び、LSP を通して転送されることをカットスルー転送と呼ぶ。

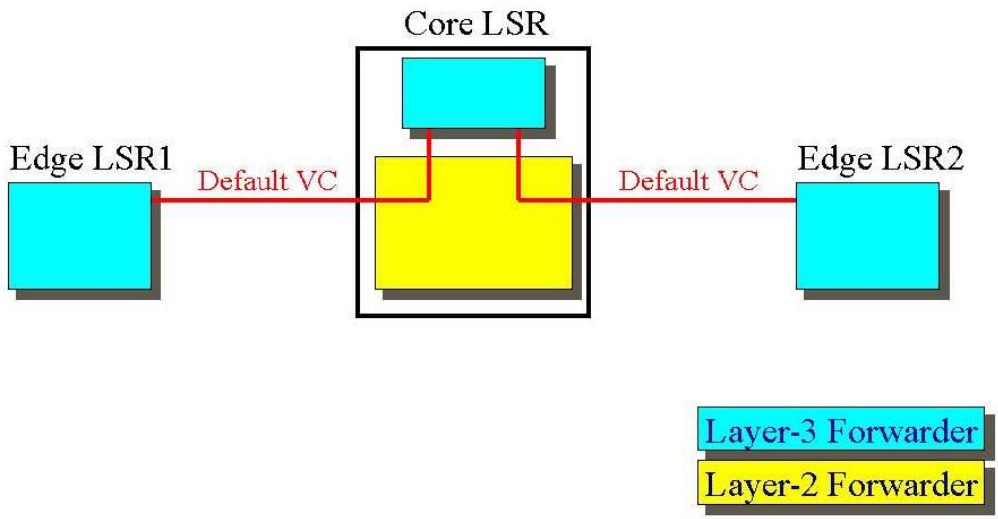


図 2.1: ネットワーク層転送

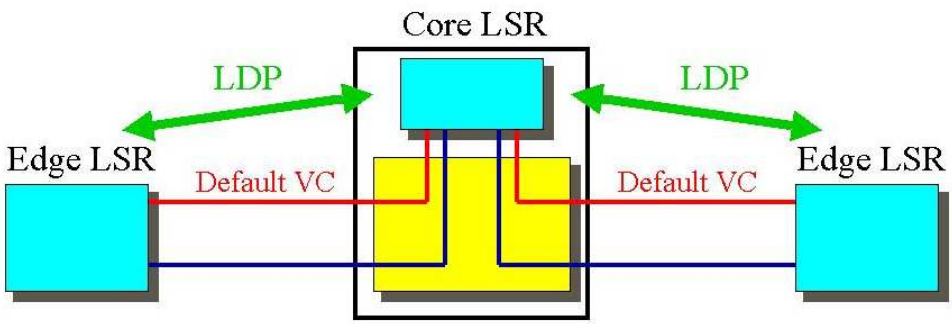


図 2.2: LDP による通知とラベルスイッチ VC の設定

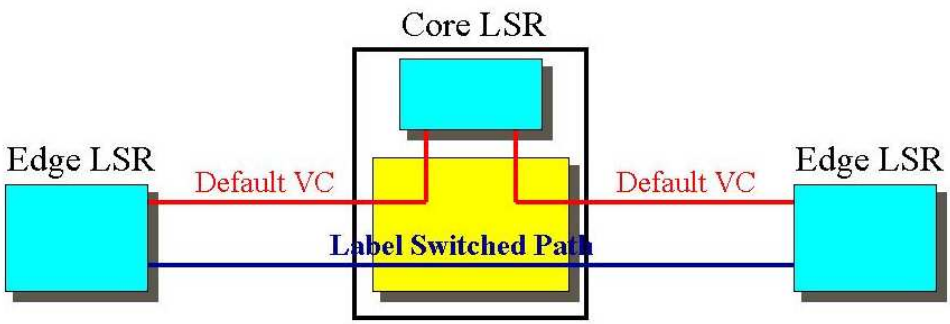


図 2.3: Label Switched Path

2.1.2 ラベル割り当てトリガ

LSP を設定するタイミングには、以下の 2 つの方式がある。

- トラフィックドリブン
- トポロジードリブン

それぞれの特徴を以下に示す。

トラフィックドリブン

トラフィックドリブンとは、ある特定のパケットを隣接 LSR に転送する時に LSP を設定する方式である。この場合、LSP にトラフィックが一定時間流れないときに LSP は解放される。

以下にトラフィックドリブンを用いる利点を挙げる。

- LSP を動的に設定するため、必要ラベル数はその時に存在しているパケットストリームの種類数になる
- ラベル毎、つまりパケットストリーム毎ごとに排他的帯域制御や QoS 制御が可能

以下にトラフィックドリブンを用いる欠点を挙げる。

- パケットストリーム数が多いと必要ラベル数が急増
- 動的な LSP 設定による遅延
- LSP を設定する間、また割り当てられるラベルが無い場合は、そのパケットフロー のパケットは従来のホップホップ転送される

トポロジードリブン

トポロジードリブンは、ルーティングテーブルに経路エントリが加えられた時に、その経路エントリに従ってデータパケットが流れる前に予め LSP を設定する。また、ルーティングテーブルから経路エントリが削除された時に、その LSP は解放される。

以下にトポロジードリブンを利用する利点を挙げる。

- 全てのパケットがカットスルー転送可能
- LSP 設定による遅延がない
- 必要ラベル数はルーティングテーブルの経路エントリ数になる

以下にトポロジードリブンを欠点する利点を挙げる。

- パケットストリーム毎の QoS 制御ができない

2.1.3 ラベル割り当ての粒度

それぞれの LSP には、それぞれ割り当てられたパケットストリームが流される。パケットストリームの種類として、例えば (送信元アドレス, 宛先アドレス) が同じであるパケットフローの集合が考えられる。これはある特定のホストから他の特定のホストへのパケットフローの集合である。パケットストリームの種類によって、LSP 数が増減するため、必要ラベル数も変化する。このラベルを割り当てる単位であるパケットストリームの種類を決定することをラベル割り当ての粒度と呼ぶ。ひとつのラベルに割り当てられたパケットストリームに属するパケットは全て同じ転送処理が行われる。従ってこのラベル割り当ての粒度によって QoS 制御できるパケットフローの集合が決定される。

パケットストリームの種類として、以下に示すようなものが考えられる。

- (送信元アドレス, 送信元ポート番号, 宛先アドレス, 宛先ポート番号, トランスポートプロトコル) が同一のパケットフロー
- (宛先アドレス, 宛先ポート番号, トランスポートプロトコル) が同一のパケットフローの集合
- (宛先ネットワークアドレス, 宛先ポート番号, トランスポートプロトコル) が同一のパケットフローの集合
- (送信元アドレス, 宛先アドレス) が同一のパケットフローの集合
- (送信元アドレス, 宛先ネットワークアドレス) が同一のパケットフローの集合
- (送信元ネットワークアドレス, 宛先ネットワークアドレス) が同一のパケットフローの集合
- (宛先アドレス) が同一のパケットフローの集合
- (宛先ネットワークアドレス) が同一のパケットフローの集合
- (始点 **LSR**, 終点 **LSR**, 宛先ポート番号, トランスポートプロトコル) が同一のパケットフローの集合
- (始点 **LSR**, 終点 **LSR**) が同一のパケットフローの集合

などが考えられる。

必要ラベル数を減らすために、よりパケットフローを集約したパケットストリームを選ぶことをパケットフローアグリゲーションすると呼ぶ。

2.1.4 ラベル割り当て方針

現在提案されているラベル割り当て方針は 2 つある。

一つは、IFMP[9] と FANP[3] で定義されている、パケットストリームとして (送信元アドレス, 宛先アドレス) の同じパケットフローの集合を選び、ラベル割り当てトリガとして、トラフィックドリブンを利用する手法である。

もう一つは、TDP[7] で定義されている、パケットストリームとして宛先ネットワークアドレスの同じパケットフローの集合を選び、ラベル割り当てトリガとして、トポロジードリブンを利用する手法である。

2.2 Cell Switch Router

2.2.1 Cell Switch Router(CSR) の概要

CSR はデータリンク層技術として ATM を利用した LSR であり、LDP として FANP を用いる。Edge CSR は、ATM のネットワークインターフェースを持ち、FANP が実装されたルータで構成される。Core CSR は、ネットワーク層転送モジュールとして、ATM のネットワークインターフェースを持ったルータを利用し、データリンク層転送モジュールとして ATM スイッチを利用する。

2.2.2 CSR の特徴

現在の CSR は FANP version2 を利用している。FANP version2 では、

- Distributed Control Mode(DC Mode)[4]
- Ingress Control Mode(IC Mode)[5]

の 2 つの動作モードをサポートしている。DC Mode では、ラベル割り当てトリガとしてトポロジードリブンを利用する。一方 IC Mode では、ラベル割り当てトリガとしてトラフィックドリブンを利用する。

現状の DC Mode ではパケットストリームとして、宛先ネットワークアドレスの同じパケットフローの集合を利用する。

また IC Mode ではパケットストリームとして、送信元アドレスと宛先アドレスの同じパケットフローの集合を利用する。DNS query などの継続時間の短いパケットフローなどには、LSP を設定せず、予め表 2.1 に示すようなトリガパケットフローを定義し、その条件に一致するパケットを受け取った時に LSP を設定する。この LSP には図 2.1 で決められた priority が割り当てられ、また、このパケットの送信元アドレスと宛先アドレスの組というパケットストリームが対応付けられる。一度設定された LSP には送信元アドレス、宛先アドレスの組が同一であるパケットストリームが流れる。一定期間パケットの流れない LSP は解放される。

この priority は、ATM での VC の優先度を示している。例えば、telnet 用に設定された LSP と、http 用に設定された LSP がある ATM スイッチで輻輳した時、priority の低いセルから落されることになる。

表 2.1: CSR で設定されるトリガパケットフローの定義例

src: source
dst: destination
net: network
trans: transport
lyr: layer
addr: address
proto: protocol
pri: priority
Asterisk : WildCard

net lyr proto	src net addr	src net mask	dst net addr	dst net mask	trans lyr proto	src port	dst port	pri
ip	*	*	*	*	tcp	*	telnet	1
ip	*	*	*	*	tcp	telnet	*	1
ip	*	*	*	*	tcp	*	http	0
ip	*	*	*	*	tcp	http	*	0
ip	*	*	*	*	tcp	*	ftp	0
ip	*	*	*	*	tcp	ftp	*	0
ip	*	*	*	*	tcp	*	nntp	0
ip	*	*	*	*	tcp	nntp	*	0

第3章

ラベルスイッチルータの現状と問題点

3.1 ラベル数の問題

3.1.1 利用可能なラベル数

LSR の一つの実装形態として、データリンク層転送技術に ATM を利用した ATM-LSR がある。これは、データリンク層転送モジュールとして ATM スイッチを用いる。この ATM-LSR を利用した場合の利用可能なラベル数は、使用している ATM スイッチや ATM ネットワークインターフェースに依存する。それぞれのラベルの値毎にリアセンブルバッファを割り当てる必要があるため、このリアセンブルバッファの容量によって、利用可能なラベル数の上限が決定される。

3.1.2 現状の LDP で利用できるパケットストリームの種類

第 2.1.3 節で、利用するパケットストリームの種類によって必要ラベル数が増減することを示した。現在提案されている LDP で利用できるパケットストリームの種類をトラフィックドリブンとトポロジードリブンについて表 3.1, 3.2 に示す。

表 3.1: トラフィックドリブンで利用できるパケットストリームの種類

(送信元アドレス, 送信元ポート番号, 宛先アドレス, 宛先ポート番号, トランスポートプロトコル)
(送信元アドレス, 宛先アドレス)
(送信元アドレス, 宛先ネットワークアドレス)
(送信元ネットワークアドレス, 宛先ネットワークアドレス)

表 3.2: トポロジードリブンで利用できるパケットストリームの種類

(宛先アドレス, 宛先ポート番号, トランスポートプロトコル)
(宛先ネットワークアドレス, 宛先ポート番号, トランスポートプロトコル)
(宛先アドレス)
(宛先ネットワークアドレス)

表 3.1,3.2 両者とも、上から必要ラベル数の多い順にパケットストリームの種類が並べてある。順に並んだ下のパケットストリームは、より必要ラベル数が少なくなる。

下の方にあるパケットストリームを用いると、必要ラベル数がより少なくできる (パケットフローアグリゲーションできる)。従って、現在の LDP ではルーティングテーブルを基にパケットストリームが決められるために、トラフィックドリブン, トラフィックドリブン両者とも、最もパケットフローアグリゲーションできる単位はネットワークアドレスであることが分かる。

3.1.3 パケットフローアグリゲーションの意味

パケットフローアグリゲーションするということは、より必要ラベル数を減少させるようなパケットストリームを選択することである。

しかしトラフィックドリブンにおいてパケットフローアグリゲーションは、必要ラベル数を減少させるという意味だけではなく、QoS 制御を提供できる単位という意味でもある。従ってパケットストリームがより多くのパケットフローを集約することによって、QoS 制御できる範囲が広くなり、ひとつのパケットフローに対して QoS 制御をすることができない。

3.1.4　さらなるパケットフローアグリゲーションの必要性

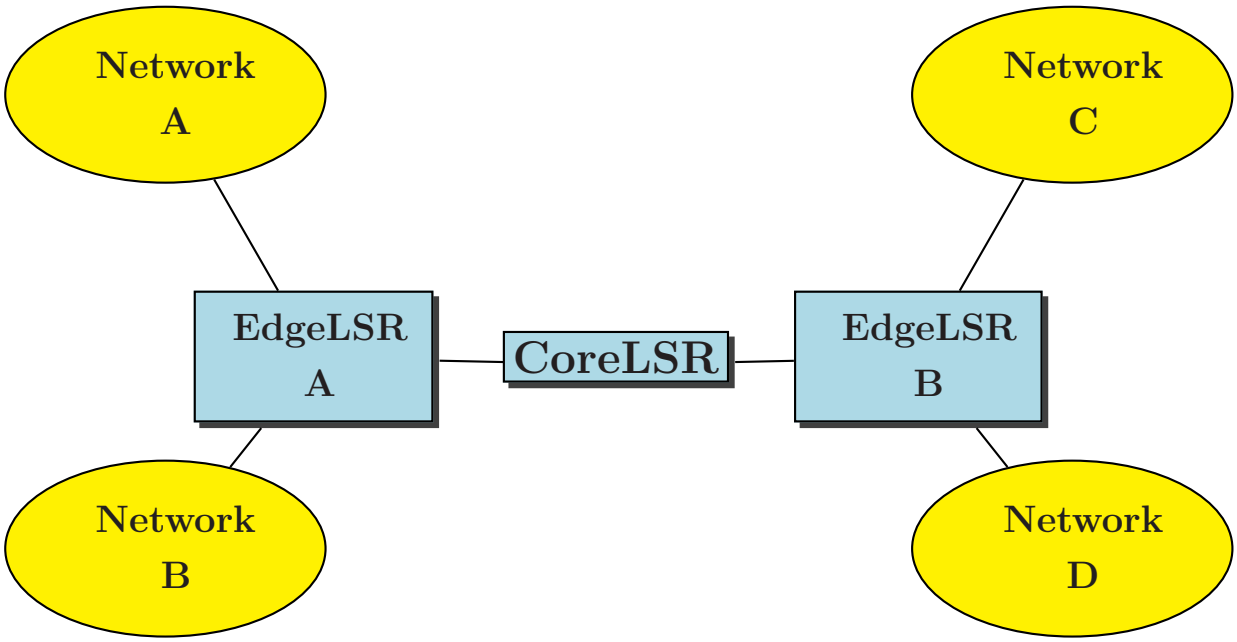


図 3.1: LSR ネットワークトポロジ例

図 3.1 の LSR を用いたネットワークがパケットストリームを宛先ネットワークアドレスの同じパケットフローの集合としたトポロジードリブンで運用されているとする。EdgeLSR A は、Network C、Network D それぞれのネットワークアドレスに、一つのラベルを割り当てる。従って図 3.2 に示すように 2 つの LSP が EdgeLSR A から設定される。

パケットの送信元アドレスが Network A、Network B に関わらず、EdgeLSR A は宛先アドレスが Network C のネットワークアドレスに属するパケットは Network C 宛の LSP を通って、同様に Network D のネットワークアドレスに属するパケットは Network D 宛の LSP を通って転送される。

この 2 つの LSP はそれぞれ、始点 LSR が EdgeLSR A であり、終点 LSR が EdgeLSR B である LSP となる。よって、同一 EdgeLSR 間に 2 本の LSP が張ってあることになる。

- カットスルー転送できるのは 2 つの EdgeLSR 間だけである
- トポロジードリブンでは、LSP 毎に QoS 情報を持たせることができない
- 同優先度の LSP 間では、異なる優先度を持たせることはできない
- ひとつの LSP 内を通過するパケット間に異なる優先度を持たせることはできない

以上の理由より、同一 EdgeLSR 間に、同優先度の LSP が複数設定されることは不必要である。

また、同一 EdgeLSR 間に異なる優先度の LSP が複数設定された場合、一つの EdgeLSR で必要な最大ラベル数は倍増する。例えば n 個のクラスの優先度の LSP を設定する場合、必要なラベル数は n 倍になる。従って PS を宛先ネットワークアドレスのパケットフローの集合にした時、n レベルの優先度をラベルに持たせるには、ルーティングテーブルの経路エントリ数の n 倍必要になる。

現在のパケットフローアグリゲーションを用いて LSP に複数の優先度を持たせるには、経路エントリ数の多いネットワーク上のルータでは、ラベルが十分でない。

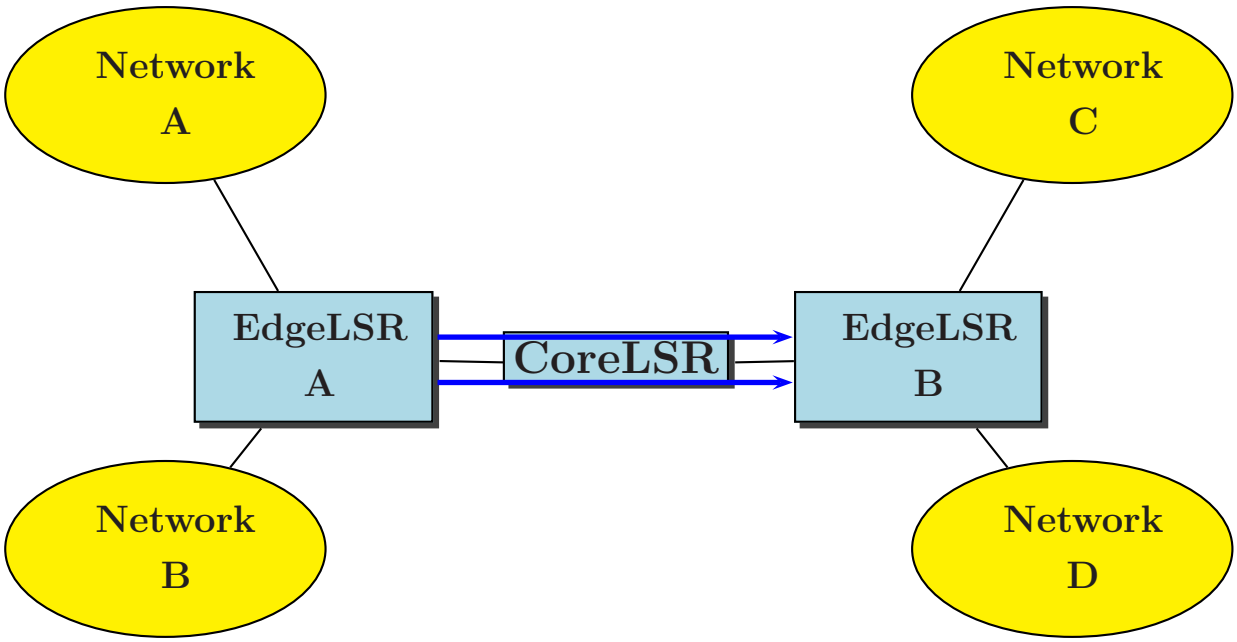


図 3.2: 現状のパケットフローアグリゲーションされた LSP

3.2 優先転送サービス体系

現状のラベルスイッチ技術は、パケットストリームの優先度によっていくつかの異なる処理が可能である。例えば

1. 優先度の低いパケットストリームはホップホップ転送
2. 優先度の高いパケットストリームはカットスルー転送

が考えられる。また 2 には、データリンク層転送技術の優先度を利用することによって、更に数段階の優先度を利用できる。

このようにラベルスイッチ技術は、データリンク層転送技術を積極的に利用する。本節では現状の Core LSR、Edge LSR での転送処理を以下に示し、問題点を示す。

3.2.1 現状の優先処理 (Core LSR)

Core LSR では、以下の 2 種類のパケットに対して異なる転送処理が行われる。

1. ホップホップ転送パケット
2. カットスルー転送パケット

ホップホップ転送パケットに対する処理は、ネットワーク層転送処理が伴う。第 3.2.2 節で論述する。

一方カットスルー転送パケットに対する転送処理は、データリンク層転送技術が提供するものである。データリンク層転送技術に、ATM のような VC 毎に優先度を提供できる技術を利用した例を図 3.3 で説明する。図の ATM スイッチは複数の OC3 のポートを持ち、表 3.3 のように設定されている。高優先度 VC と低優先度 VC は、それぞれ異なるポートから同一のポートへスイッチされ、合流している。高優先度 VC には 30Mbps の cell、低優先度 VC には 155Mbps の cell が流れており、合流するポートで輻輳が起きている。この場合、ATM スイッチによって、図のように高優先度 VC 上のトラフィックが優先さ

れて破棄されること無く流れている。一方低優先度 VC 上のトラフィックは適度に廃棄されている。

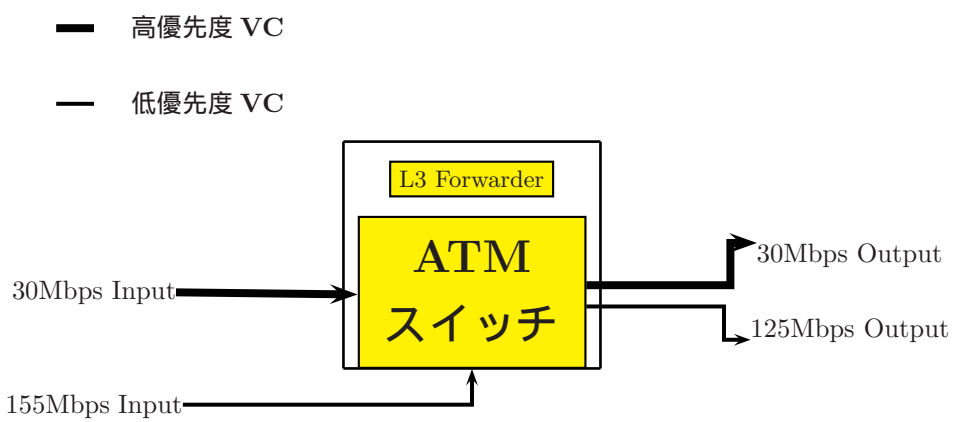


図 3.3: 現状の Core LSR での優先処理

表 3.3: ATM スイッチの設定

入力		出力		優先度	利用帯域
ポート	vpi/vci	ポート	vpi/vci		
A	0/40	C	0/40	高	35Mbps
B	0/41	C	0/41	低	155Mbps

本論文ではデータリンク層転送技術として、この様な優先処理が設定可能な技術が適用されていることを前提にする。

3.2.2 現状の優先処理 (Edge LSR)

Edge LSR でのネットワーク層転送処理や、Core LSR でのホップホップ転送処理において、輻輳時の優先処理については現在特に規定されていない。

一般的な FIFO(First In First Out) キューによる転送処理を用いた場合、低優先度のバーストトラフィックがキューを占有し、高優先度のパケットが落ちることになる。従って LSR 網内での優先度が Edge LSR において反映されないことになる。

3.2.3 現状の問題点のまとめ

LSR ネットワークのカットスルー転送によるデータリンク層転送処理では、データリンク層技術を利用することで優先度処理をしている。

しかし、ネットワーク層転送処理をする Edge LSR では、どのような処理をするべきかを特に規定していない。よって、LSR 網における優先度を反映するトラフィック制御を Edge LSR において実現しなければならない。

また、この優先度が評価されるのは LSR ネットワーク内のみであり、LSR ネットワーク以外のネットワークでは評価されない。従って図 3.4 に示されるホスト A とホスト B が通信した場合、この優先度が

評価されるのは、LSR ネットワーク内だけである。Network A と Network B で優先度が評価されないために、ホスト A とホスト B のエンドエンドの通信では優先度が反映されない。

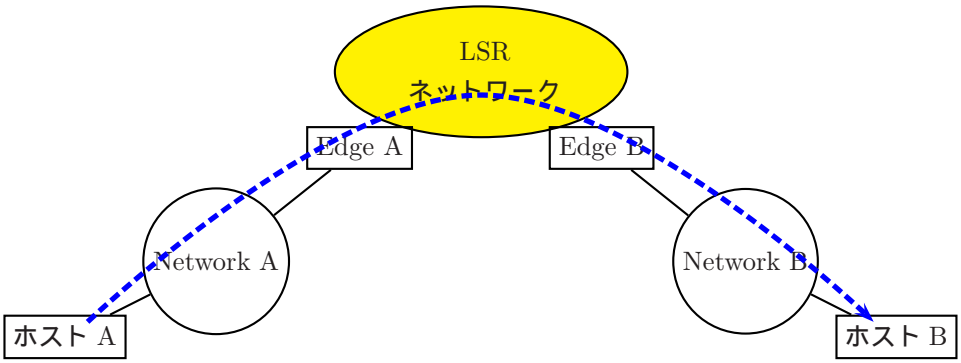


図 3.4: 優先度処理が行われる範囲

第4章

OSPF リンクステートデータベース情報を用いたパケットフローアグリゲーション

4.1 最適なパケットフローアグリゲーションと必要となる情報

LSP の優先度を一つしか利用しない場合、最も必要ラベル数の少ないパケットフローアグリゲーションは、パケットストリームを (始点 LSR, 終点 LSR) が同一のパケットフローの集合にすることである。この場合図 4.1 の様に、始点 LSR は終点 LSR の数の LSP しか張らない。つまりそれぞれの Edge LSR は自分以外の Edge LSR に対してそれぞれ 1 本の LSP を張るため、ネットワーク全体の必要ラベル数は、以下のようになる。

(Edge LSR 数) * (Edge LSR 数 - 1)

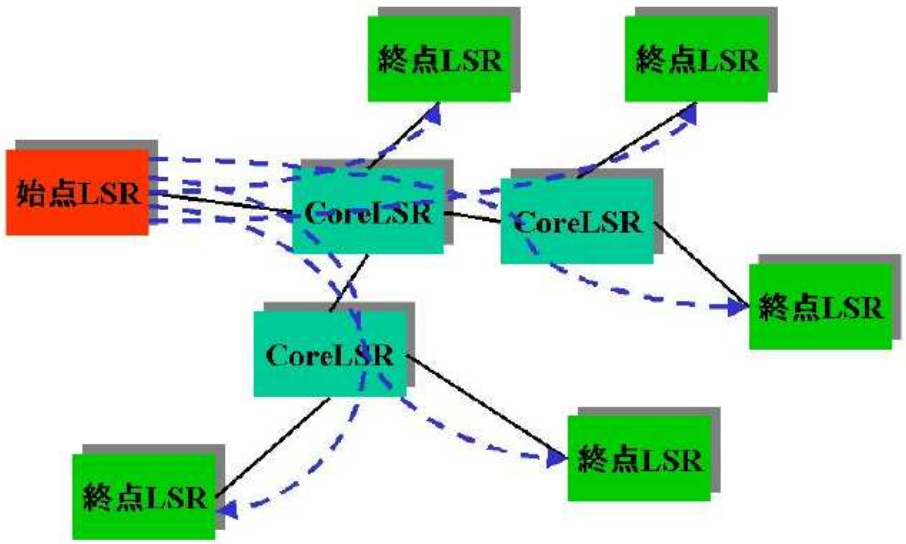


図 4.1: 最適なパケットフローアグリゲーション

ルーティングテーブルの持つ情報はネクストホップと送出インターフェースだけである。この情報だけでは、このような LSP を設定することは不可能である。よって、このような LSP を利用するためには、各 Edge LSR には新しい機能が必要になる。各始点 LSR は、パケットの宛先アドレスを見た時、この宛先アドレスはどの終点 LSR に転送するべきかを判断しなければならない。図 3.2 を例にすると、EdgeLSR A は Network C、Network D 宛のパケットは EdgeLSR B に転送すれば良いという情報を持たなければならない。

この様な情報を取得するためには、幾つか方法がある。

まずひとつの方法は、Edge LSR 間での経路情報交換することである。各 Edge LSR は、ルーティングテーブル内の、LSR ネットワークに接続されていないネットワークインターフェースへ転送される経路エントリと参考になるメトリック情報を他の Edge LSR に知らせることによって実現できる。この方法を取る場合、新たなプロトコルを作成しなければならず、またルーティングプロトコルで既に送られた経路情報を異なるプロトコルを利用し再度伝えることは、無駄な情報交換であるし、経路情報の不整合を引き起こす可能性がある。

異なる方法として、リンクステート型のルーティングプロトコルによって交換された情報を利用することである。Open Shortest Path First(OSPF)[12] や IS-IS のようなルーティングプロトコルは内部にトポロジー情報を持ち、最小コストの経路選択をする。

例を挙げると図 4.2 に示すネットワークは、5 つのルータで構成される。ルータ A は交換された各ルー

タの持つリンク情報とそのコスト情報を基に、図 4.3 の最小コストの経路 (最短経路) を導くツリーを作成する。このようなリンク情報のデータベースをリンクステートデータベースと呼ぶ。ルータ E へは、

1. ルータ A → ルータ B → ルータ E(コスト 30)
2. ルータ A → ルータ C → ルータ D → ルータ E(コスト 40)
3. ルータ A → ルータ C → ルータ E(コスト 20)

の 3 つの経路が考えられる。しかしルータ E への最小のコストである 3 つめの経路が選択されるため、図 4.3 のツリーは他の 2 つの経路を表示していない。

ルータ A、ルータ D、ルータ E が Edge LSR であり、ルータ C が Core LSR であるとして、もしもルータ D 以下に更にツリーがあった場合、ルータ A はそのルータ D 以下の全てのネットワークやルータに対しては、ルータ D へ転送すれば良いことが分かる。

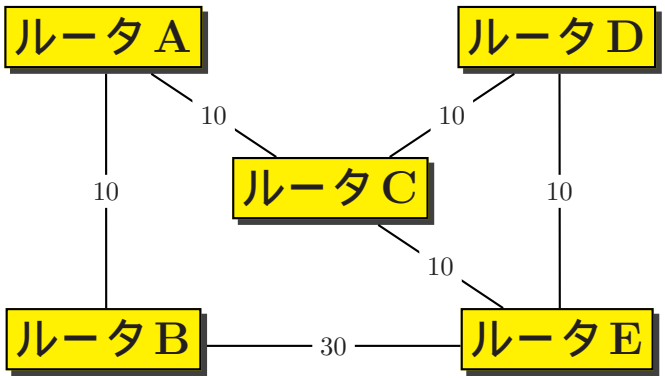


図 4.2: リンクステート型経路制御プロトコルを利用したネットワーク例

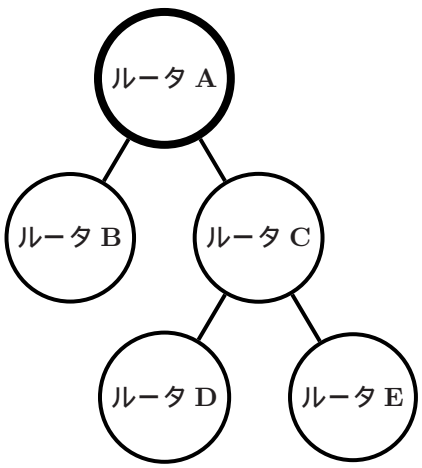


図 4.3: リンクステート型経路制御プロトコル内に持つトポロジー情報

4.2 Open Shortest Path First(OSPF) の概要

Open Shortest Path First(OSPF)[12] は IP をサポートしたリンクステート型の内部ゲートウェイプロトコルである。OSPF によって運用されるネットワークのルータは、リンクステート広告 (Link State Advertisement, LSA) を、それぞれ隣接ルータと交換する。新たに接続されたルータは、自分の持つ LSA の情報を隣接ルータと交換する。自分の持たない LSA を隣接ルータが持っていた場合は要求し受信する。従って、そのネットワークに属する全てのルータで、同じ LSA の集合を保つことができる。この LSA の集合をリンクステートデータベースと呼ぶ。ルータはリンクステートデータベースから、そのネットワークの最短経路を導くツリー (最短経路ツリー) を作るため、OSPF によって運用されるルータは同一のツリーを保つことができる。

また、OSPF で運用されるネットワークが大きくなると、交換される LSA の数が増大する。それによって、LSA による帯域の使用率が上昇したり、ネットワーク情報の変更を瞬時にすべてのルータが知ることが難しくなる。そのために OSPF にはエリアという概念を用いて、LSA の交換する範囲を限定し、エリア間ではそのエリアの概要 (Summary) だけを交換する仕組みを提供している。

各エリアは 32bit のエリア ID によって一意に識別され、エリアにに属するルータは 32bit のルータ ID によって一意に識別される。

4.2.1 OSPF パケットの種類

OSPF は IP の上位層である。OSPF のパケットは 5 種類あり、それぞれ図 4.4 に示す共通のヘッダを持つ。

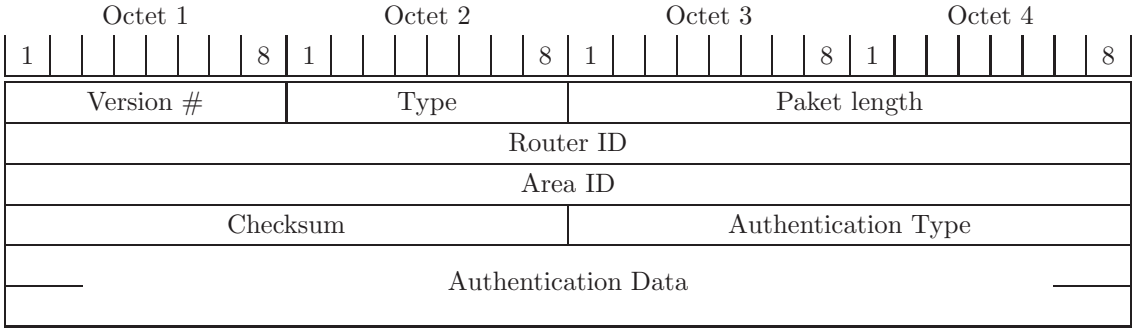


図 4.4: OSPF Common Packet Header

以下に各フィールドの説明をする。

Version: OSPF の version 番号 (現在は 2)

Type: このパケットの種類 (後述)

Packet Length: 共通パケットヘッダを含むパケット長 (byte 単位)

Router ID: このパケットを転送するルータの ID

Area ID: このパケットを転送するルータの所属する Area ID

Checksum: Authentication Data を除くパケット全体のチェックサム

Authentication Type: 以下のどれかの認証の方式を指定

- 0: 認証無し
- 1: 単純なパスワードによる認証
- 2: 暗号化されたパスワードによる認証

Authentication Data: Authentication Type で指定されたパスワード

Type フィールドの値によって、そのヘッダに続くパケットの内容が変わる。OSPF のパケットの種類を表 4.1 に示す。

表 4.1: OSPF Packet Type

Type	Packet Type
1	Hello Packet
2	Database Description Packet
3	Link State Request Packet
4	Link State Update Packet
5	Link State Acknowledgement Packet

Hello Packet はリンク上の新たなルータを探し出すために利用される。自分の持っている LSA と新たに見つかったルータの持っている LSA を Database Description Packet によって互いに知らせ、Link State Request Packet によって互いに要求しあう。LSA の転送は Link State Update Packet を用いて行い、受け取ったルータは Link State Acknowledgement Packet を用いて正常に受信したことを送信したルータに知らせる。

Hello Packet、Database Description Packet、Link State Request Packet、Link State Acknowledgement Packet は本論文と直接関わらないため、第 8 章の図 8.1,8.2 ,8.3,8.4 にそれぞれパケットフォーマットを示すのみとする。

図 4.5 に Link State Update Packet のパケットフォーマットを示し、以下にその説明をする。

 # LSAs このパケットに含まれる LSA の数

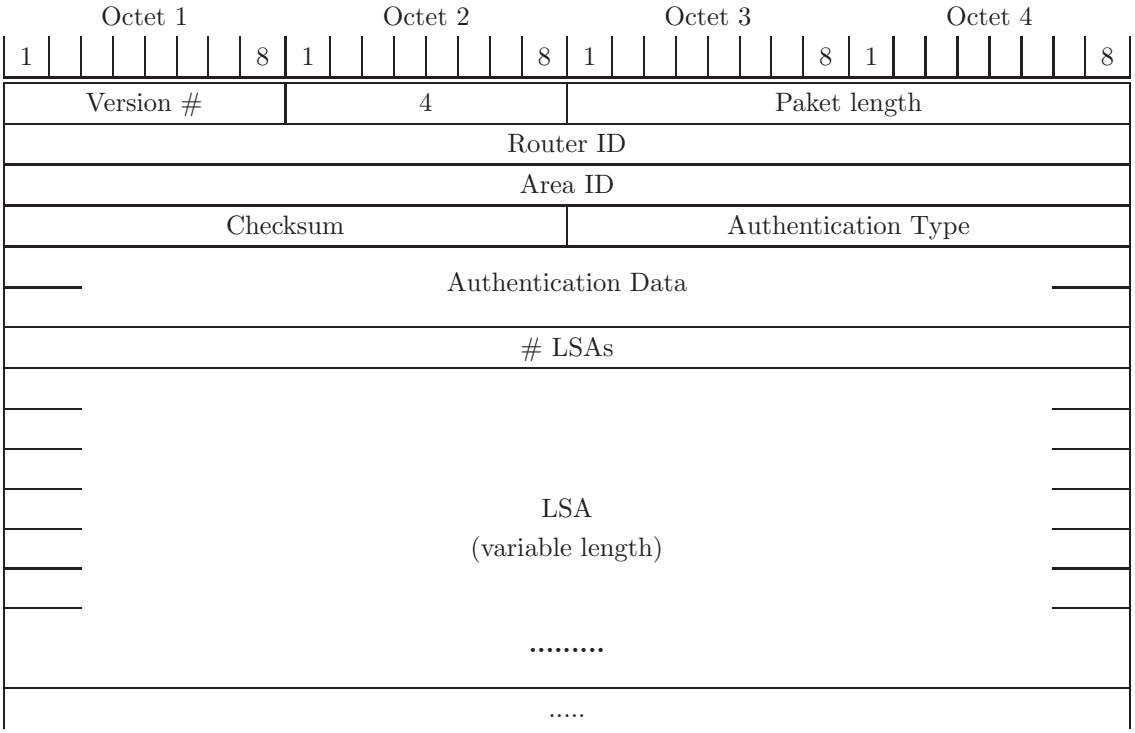


図 4.5: Link State Update Packet

4.2.2 LSA の種類

LSA はそれぞれ共通のヘッダを持つ。そのヘッダを図 4.6 に示す。

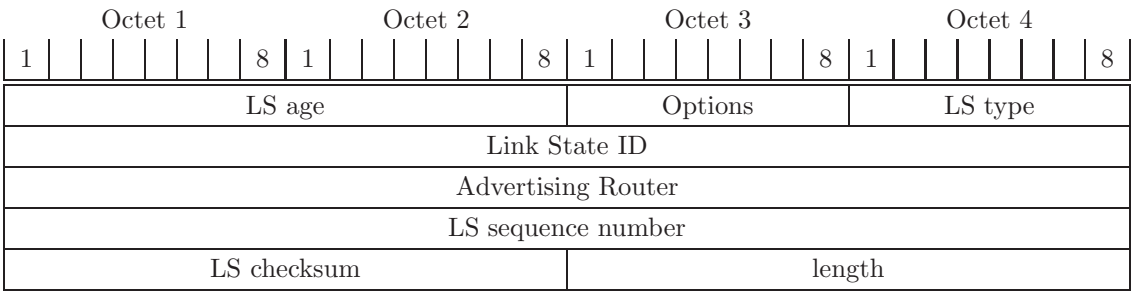


図 4.6: LSA Header

以下に各フィールドの説明する。

LS age: その LSA の秒単位の年齢 (3600 秒で LSA が無効になる)

Options: 付加機能を記述

LS Type: LSA の種類を記述 (後述)

Link State ID: LS Type によって異なる

Advertising Router: この LSA を生成し、広告したルータのルータ ID

LS sequence number: LSA を識別する番号

LS checksum: LS age を除く、LSA 全体のチェックサム

length: byte 単位の LSA の長さ

LS Type の種類を表 4.2 に示す。

表 4.2: LSA Type

LS Type	Type of LSA
1	Router LSA
2	Network LSA
3	Summary LSA(inter-area)
4	Summary LSA(AS Border Router)
5	AS External LSA

以下にそれぞれの LSA のフォーマットと説明をする。

Router LSA

各ルータは Router LSA を生成する。Router LSA は、LSA を生成したルータの持つリンクやそのコストなど、ルータ自身の情報を記述する LSA である。図 4.7 にフォーマットを示す。

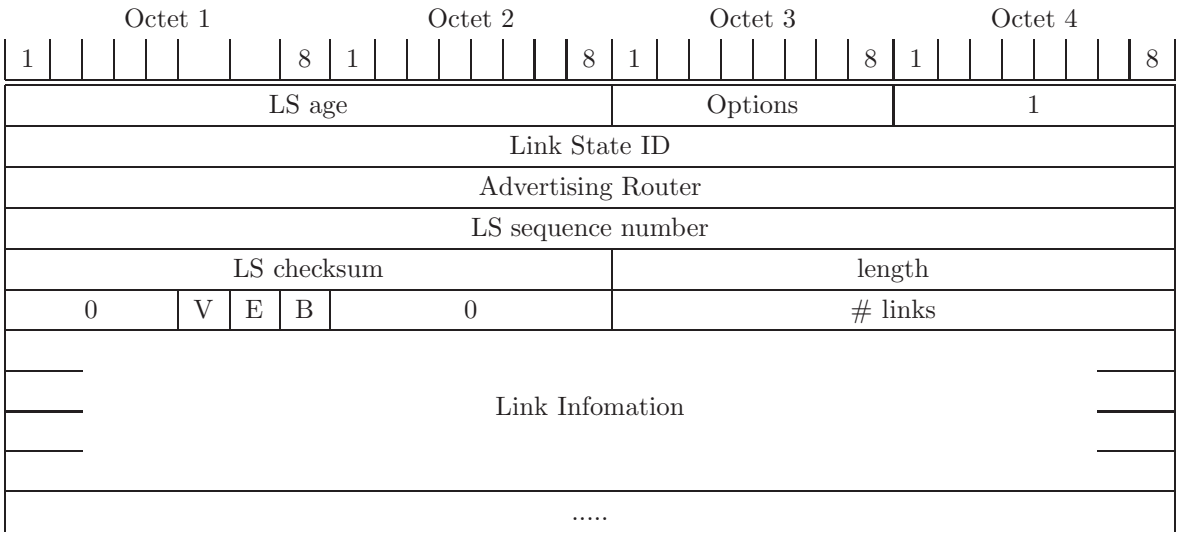


図 4.7: Router LSA

以下に Router LSA の各フィールドの説明する。

Link State ID: この LSA を生成したルータのルータ ID

V bit: 仮想リンクの端点であることを示す

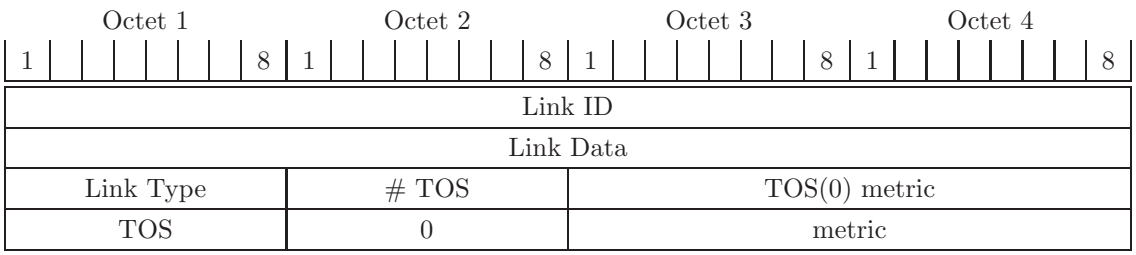
E bit: AS 境界ルータであることを示す

B bit: エリア境界ルータであることを示す

 # links: この LSA に含まれる Link Infomation の数

Link Infomation: ルータの持つリンク情報

ルータのリンク情報は図 4.8 に示すフォーマットで記述される。



Octet 1								Octet 2								Octet 3								Octet 4							
1							8	1							8	1							8	1							8
Link ID																															
Link Data																															
Link Type								# TOS								TOS(0) metric															
TOS								0								metric															

図 4.8: Link Infomation

TOS(0) metirc フィールドは IP ヘッダの TOS(Type Of Service) の値が 0 のパケットに対しての、このリンクを使用するコストが記述される。

TOS フィールドは、このリンクにおいて、パケットの TOS によってコストを変える場合、TOS の値 0 以外いくつの TOS をサポートするのかを示す。

TOS は 0 以外の TOS の値を記述する。metric にはその TOS のパケットに対するこのリンクを使用するコストが示される。

Link Type はこのリンクの種類を示している。Link Type によって示されるリンクの種類を表 linktype1 に示す。

表 4.3: Link Type の種類

Link Type	そのリンクの種類
1	ポイントツーポイントのリンク
2	通過ネットワークへのリンク
3	スタブネットワークへのリンク
4	仮想リンク

OSPF では、ルータ間のリンクは基本的には 1 対 1 のポイントツーポイントのネットワークとして抽象化される。しかしブロードキャスト型や ATM などの NBMA(Non-Broadcast MultiAccess) 型のようなマルチキャスト可能なリンク上では、それぞれのルータが同じリンク上の他のルータとそれぞれ隣接ルータになると、それぞれのルータ間で LSA の交換が行われるため、リンクに流れる LSA の数が増大する。

そのため OSPF では、マルチキャスト可能なリンク上に代表ルータという特別なルータを選出させる。そのリンク上の全てのルータは代表ルータに対して LSA を転送する。代表ルータは他のルータから受け取った LSA をマルチキャストアドレスに転送し、リンク上に流れる LSA の数を減らしている。通過ネットワークへのとは、このマルチキャスト可能なリンクである。

Link Type、Link ID, Link Data それぞれの関係を表 4.4 に示す。

表 4.4: Link Type 別の Link ID、Link Data の持つ値

Link Type	Link ID	Link Data
1	隣接ルータのルータ ID	このリンクの IP アドレス
2	このネットワーク上の代表ルータの IP アドレス	このリンクの IP アドレス
3	ネットワークアドレスプレフィクス	ネットワークマスク長
4	隣接ルータのルータ ID	このリンクの IP アドレス

Network LSA

マルチキャスト可能なリンク上の代表ルータは Network LSA を生成する。Network LSA は、その通過ネットワークに属する全てのルータのリンク ID を表す。Network LSA のフォーマットを図 4.9 に示す。

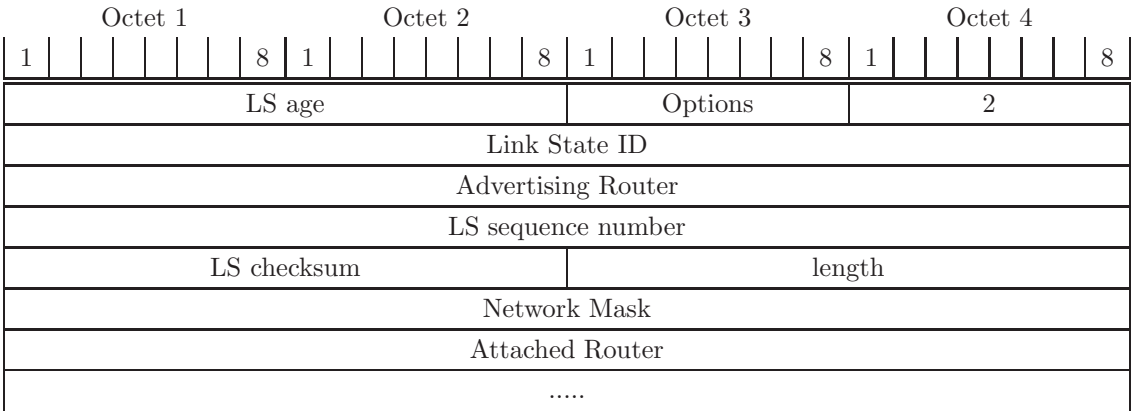


図 4.9: Network LSA

Network LSA の各フィールドについて以下に説明する。

Link State ID: 代表ルータのこのネットワーク上の IP アドレス

Network Mask: そのネットワークのネットワークマスク

Attached Router: この Feild とこれ以降の Feild にはこのネットワークに所属するルータのルータ ID が並ぶ

他の LSA

その他の LSA について簡単に説明する。

Summary LSA(inter-area): 他のエリアの経路の概要を示す。

Summary LSA(AS Border Router): AS 境界ルータの存在を示す。

AS External LSA: AS 外経路を示す。

これらの LSA のフォーマットを第 8 章の図 8.5、8.6 に示す。また、本論文に関係あるフィールドの説明を表 4.5 に示す。

表 4.5: その他の LSA のフィールド

LS Type	Link State ID	Netmask
3	他エリアの経路のネットワークアドレス	そのネットワークのネットマスク
4	AS 境界ルータのルータ ID	0
5	AS 外経路のネットワークアドレス	そのネットワークのネットマスク

4.2.3 最短経路の計算

最短経路計算の第一段階として、各ルータは Router LSA と Network LSA に対し、Dijkstra のアルゴリズムを利用して図 4.3 のような自分をルートで末端がスタブネットワークまたはエリア境界ルータ、AS 境界ルータである最短経路ツリーを作る。このツリーが同じエリアに属するネットワーク、ルータへの最短経路ツリーになる。

次に第二段階として他エリアの経路や AS 外経路は、最短経路を広告したエリア境界ルータか AS 境界ルータの下に追加する。

4.3 OSPF リンクステートデータベース情報を用いたパケットフローアグリゲーションの設計

4.3.1 概要

以下では、LSP とパケットストリームとの対応情報を管理し、ラベル情報を隣接 LSR と交換するモジュールを LDP モジュールと呼び、OSPF を利用してルーティングテーブルの経路を操作するモジュールを OSPF モジュールと呼ぶ。

まず、各 Edge LSR(始点 LSR) 内の OSPF モジュールはリンクステートデータベースの情報から最短経路ツリーを作る経過の中で、他の Edge LSR(終点 LSR) があること認識し、その終点 LSR から先のトポロジー情報を取得し、LDP モジュールに対して、同じ終点 LSR を通過する宛先ネットワークアドレスの集合を渡す。

始点 LSR の LDP モジュールは、終点 LSR に対し 1 つの LSP を設定し、その LSP に対して、OSPF モジュールから渡された宛先ネットワークアドレスの集合を対応付けする。

以上の操作によって、各終点 LSR へは一つのみ LSP が設定され、パケットフローアグリゲーションが実現される。

4.3.2 必要となる機能

リンクステートデータベースによる最短経路ツリーを利用してパケットフローアグリゲーションするために、必要となる機能を以下に示す。

- 他の Edge LSR のルータ ID を知る機能
- LDP モジュールに終点 LSR の情報を渡す機能と渡す情報の定義
- OSPF モジュール内の最短経路ツリーを作る機能に付加する、それぞれのルータが Edge LSR か調べる機能

他の Edge LSR のルータ ID を知る機能

Edge LSR が、リンクステートデータベースによる最短経路ツリーを利用してパケットフローアグリゲーションするには、最短経路ツリーの中でどのルータが Edge LSR なのか識別しなくてはならない。つまり各 Edge LSR では、他の Edge LSR のルータ ID を知る必要がある。これには以下のような方法が考えられる。

静的な設定 各 Edge LSR で、他の Edge LSR 全てのルータ ID を静的に設定する

LDP の拡張 LDP を拡張し、この LSR ネットワーク上にある全ての Edge LSR のルータ ID を各 Edge LSR に知らせる

Opaque LSA の利用 Opaque LSA[13] を利用し、各 Edge LSR に知らせる。第 6.1.1 節で詳細に説明する。

LDP モジュールに渡すべき情報

LDP モジュールに対して行う操作は、少なくとも以下のようなメッセージによって実現される。

ADD Edge LSR への LSP へ新たな経路を追加

DELETE 無くなった経路を削除

ADD メッセージに必要な情報を以下に示す。

1. 宛先ネットワークアドレス
2. 宛先ネットワークマスク
3. 終点 LSR ルータ ID
4. 終点 LSR の LSR ネットワーク側ネットワークインターフェースアドレス
5. その宛先へのネクストホップまたは出力ネットワークインターフェース情報

LDP モジュールは各終点 LSR に対する LSP と OSPF のルータ ID との対応表を持ち、3 から一意に LSP を検索する。LSP が設定されていない場合は、4 に対して LSP を設定する。1 と 2 の二つをその LSP に対応付ける。

図 4.10 のようなネットワークで、LSR ネットワークを通過する以外の経路がある場合、LSR ネットワーク以外を通る経路を Edge LSR B への LSP に対応付けするべきでない。この場合は 5 を使って、LSP に対応付けする経路をフィルターする。

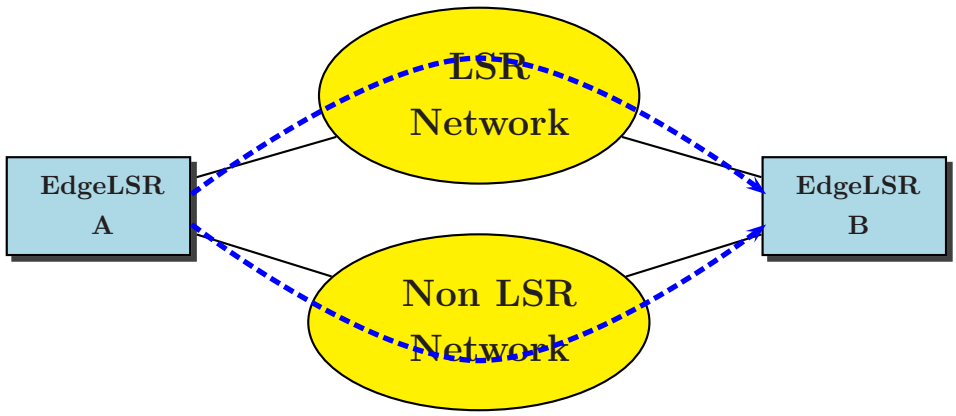


図 4.10: 他の経路のある LSR ネットワーク例

DEL メッセージに必要な情報は、その経路を識別できるだけの情報があれば十分である。

4.3.3 実現する機構

OSPF で運用され、全て同一エリアに所属する図 4.11 の LSR ネットワークで、実現するパケットフローアグリゲーションの機構を Edge LSR A を中心にして説明する。

このネットワークには、Edge LSR A、Edge LSR B、Core LSR の 3 つの LSR があり、各 Edge LSR には、ルータ以外のホストが接続するネットワークがそれぞれ接続されている。 Edge LSR A と Core LSR はポイントツーポイントで接続され、Core LSR と Edge LSR B は NMBA Network に接続し、Core LSR が代表ルータになっているとする。TOS 値 0 以外のルーティングはしないものとし、Edge LSR A には、Edge LSR B のルータ ID が静的に登録されているものとする。

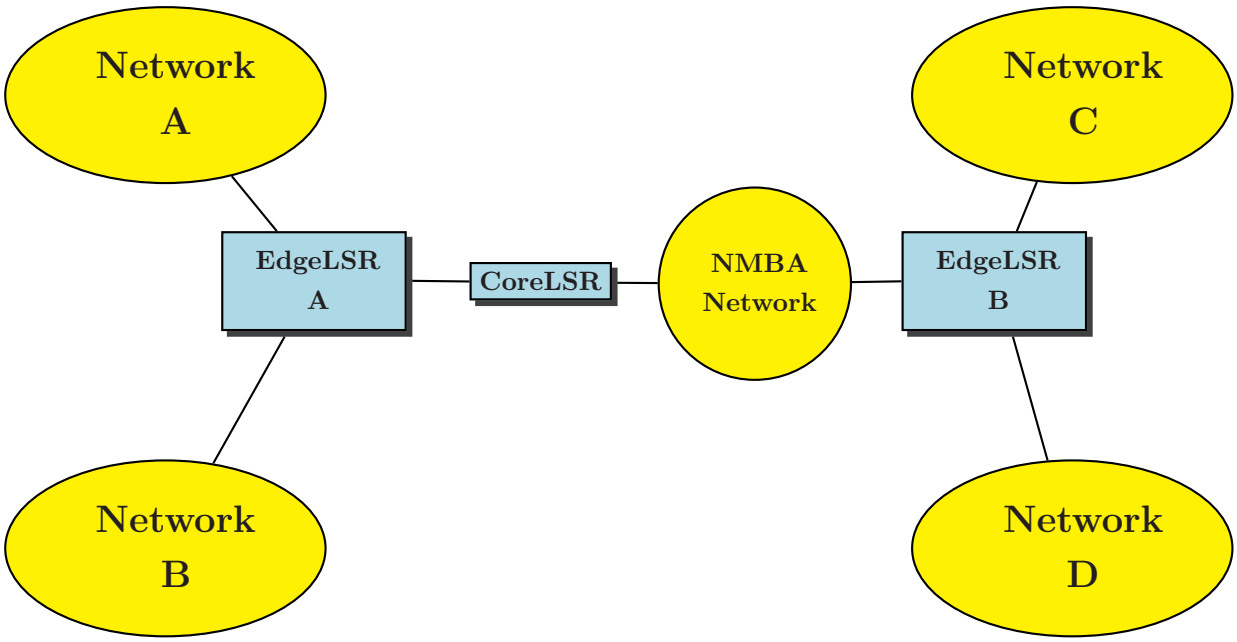


図 4.11: LSR を用いたネットワーク例 2

まず Edge LSR A は、Edge LSR A のルータ ID が Link State ID である、Router LSA を自分の保持するリンクステートデータベースから検索する。その LSA に記述される必要な情報を図 4.12 に示す。その Router LSA の持つリンク情報から図 4.13 に示すツリーを作る。Network A、Network B へのリンクはスタブネットワークであるため、これ以上ツリーの枝が作られることはない。Core LSR へのリンクはポイントツーポイントのリンクであるため、リンクステートデータベースから Link State ID が Link ID(Core LSR のルータ ID) である Router LSA を検索する。その LSA を図 4.14 に示す。

まず、Edge LSR A は自分以外の Router LSA の Link State ID を、登録された他の Edge LSR のルータ ID のリストにあるかどうか調べる。これは Core LSR の Link State ID のため、リストには記述されていない。次に Edge LSR A は、この Router LSA の Edge LSR A 側でないリンク情報を調べる。Core LSR には、Edge LSR A 側以外のリンク (NMBA Network) があり、このリンクが通過ネットワークへのリンクであるため、リンクステートデータベースから、Link State ID が Link ID(このリンク上の Core LSR の IP アドレス) である Network LSA を検索する。その LSA を図 4.15 に示す。この LSA を調べることで、図 4.16 に示すツリーを作ることができる。

今までと同様に、このリンクに接続される Core LSR 以外のルータのルータ ID が Link State ID である Router LSA を検索することで Edge LSR B の Router LSA を取得することができる。

先と同様に、この Router LSA の Link State ID が、Edge LSR のルータ ID か調べる。この Router

LSA は Edge LSR B のものなので、このルータ ID は登録されている。Edge LSR A は Edge LSR B を認識し、このツリーに終点 LSR として印を付ける。これ以降の LSA は通常通りに処理されるが、ツリーの葉には、終点 LSR の情報が継承され、図 4.17 になる。

OSPF モジュールはこの最短経路ツリーを利用して、ルーティングテーブルに経路情報を追加する。この際に、LDP モジュールに対して、終点 LSR を通過する経路の集合と終点 LSR の情報を知らせる。

LDP モジュールは、第 4.3.1 節の通り、終点 LSR への LSP へ対応付けする。

Octet 1								Octet 2								Octet 3								Octet 4							
1							8	1							8	1							8	1							8

LS age					Options	1
Edge LSR A のルータ ID						
Edge LSR A のルータ ID						
LS sequence number						
LS checksum					length	
0	V	E	B	0	3	
Network A のアドレスプレフィクス						
Network A のネットワークマスク						
4				0	TOS(0) metric	
Network B のアドレスプレフィクス						
Network B のネットワークマスク						
4				0	TOS(0) metric	
Core LSR のルータ ID						
このリンクに付いている Edge LSR A の IP アドレス						
1				0	TOS(0) metric	

図 4.12: Edge LSR A の Router LSA

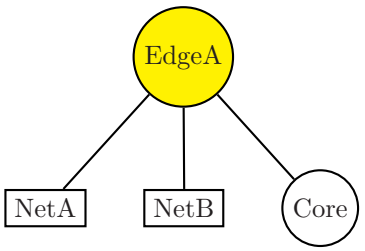


図 4.13: 最短経路ツリー 1

4.4 実装

本節では、実装の重要な点に関して簡単に説明する。

実装は CSR 上で、gated-3.5.9 を利用した。

また、終点 CSR のルータ ID は静的に設定するようにした。第 4.3.2 節で示したような、CSR ネットワーク以外を通過する経路をフィルタするために、CSR ネットワークへのネットワークインターフェー

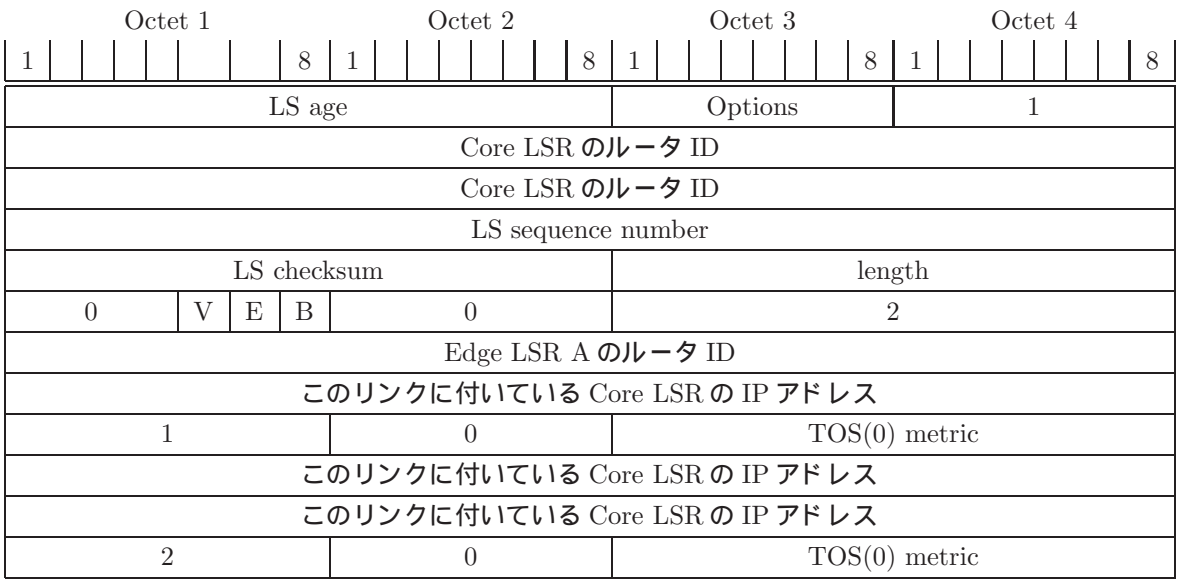


図 4.14: Core LSR の Router LSA

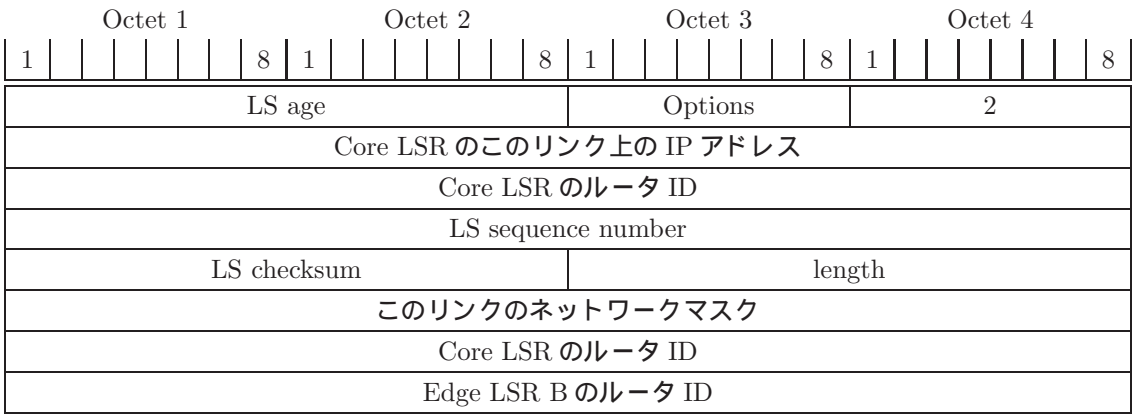


図 4.15: NMBA Network の Network LSA

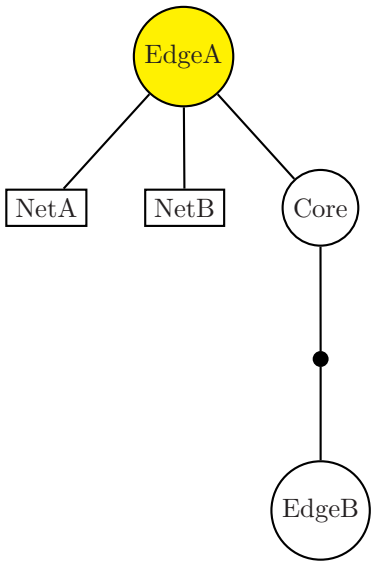


図 4.16: 最短経路ツリー 2

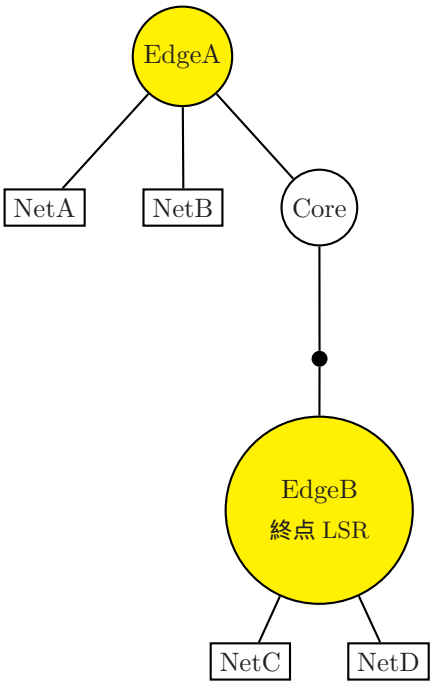


図 4.17: 最短経路ツリー 3

スも同時に設定できるようにした (図 4.18 参照)。

```
# Router ID(Interface name)
10.2.2.1(ac0) # Edge2
10.2.3.1(ac0) # Edge3
10.2.4.1(ac0) # Edge4
10.2.5.1(ac0) # Edge5
```

図 4.18: パケットフローアグリゲーションのための設定例

FANP やラベルとパケットストリームの対応情報を管理する fanpd と gated との通信インターフェースには UNIX ドメインソケットを利用した。その図を図 4.19 に示す。

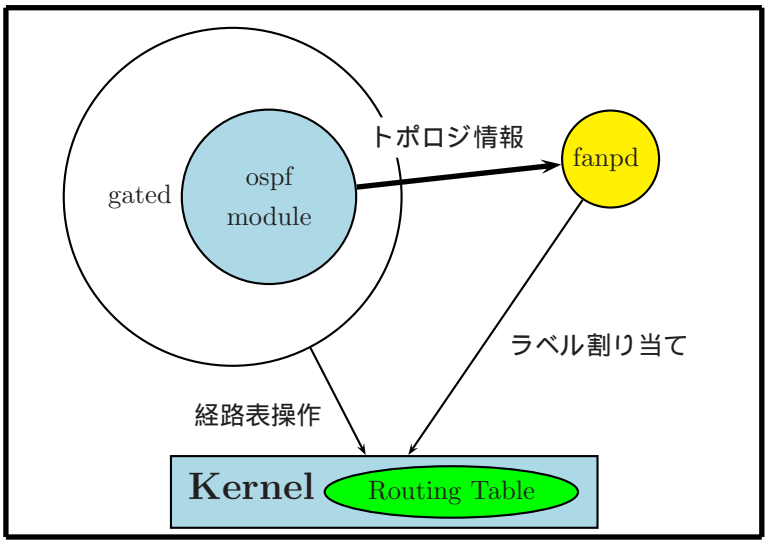


図 4.19: CSR 内の構造

また、gated から fanpd への ADD メッセージと DEL メッセージをそれぞれ、図 4.20、4.21 に示す。ADD メッセージの説明の必要なフィールドを以下に説明する。

version: fanpd へのメッセージのプロトコルバージョン

opcode: 操作コード (1: Add, 2: Delete, 3: Change, 4: Delete All)

Address Family: アドレスファミリを示す。

Routing Protocol: ルーティングプロトコルを示す。OSPF(1) が設定される。

Prefix Length: Destination Address のプリフィクス長

Destination Address: 終点 CSR のルータ ID

Network Interface index: 出力インターフェースの ifindex

Egress CSR Network Mask: 終点 CSR の Core 向きネットワークインターフェースのネットマスク

Nexthop Address: ネクストホップのアドレス

Egress CSR Network Address: 終点 CSR の Core 向きネットワークインターフェースのネットワークアドレス

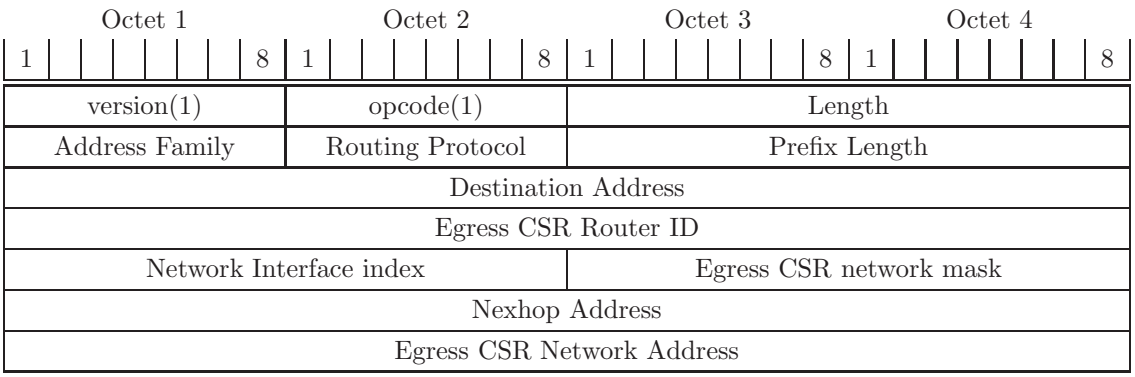


図 4.20: FANPD への ADD メッセージ

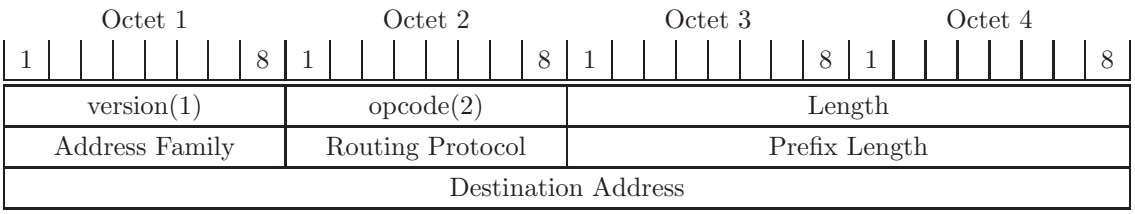


図 4.21: FANPD への DEL メッセージ

このパケットフローアグリゲーションを実現する上で最も重要な終点 CSR を通る経路の発見に関するコードを図 4.22 に示し、以下に説明する。

変数 v, parent は最短経路ツリーのノード (ルータまたはネットワーク) を構成する LSDB 構造体である。parent は親ノードであり、v は子ノードであり、両者は接続している。この構造体は LSA や Edge 情報 (lsdb_edgenode) を保持している。lsdb_edgenode メンバは、Edge 構造体へのポインタであり、終点 CSR に関係する情報を保持している。

2,3 行目で v が Network LSA である場合は、parent の lsdb_edgenode(Edge 情報) を継承する。6 行目からは v が Router LSA の場合の処理である。7 行では、この Router LSA を生成したルータが、終点 CSR か

```
edge = (*fc_ospf.fc_get_edgeent)(&fc_ospf, LS_ID(v))
```

で調べている。二つ目の引数の LS_ID(v) は v の中に保持される LSA の Link State ID を参照するためのマクロである。

この関数では、引数に渡された Link State ID が終点 CSR のルータ ID のリストにあれば、その終点 CSR の情報を保持している Edge 構造体のポインタを返す。もしなければ、NULL ポインタを返す。

この Router LSA が終点 CSR が生成したものでない場合、8,9 行目のように parent の Edge 情報を継承する。

もしこの Router LSA が終点 CSR の生成したものであれば、v の Edge 情報を edge にし (13 行目)、v と parent が共有するリンクの v のネットワークインターフェースのアドレスを調べ (14 行目)、Edge 情報に登録する (16 行目)。

```
/*
 *  ospf_spf.c: ospf_add_parent() 内
 */
1        switch(LS_TYPE(v)){
2        case LS_NET: /* Network LSA */
3         v->lsdb_edgenode = parent->lsdb_edgenode;
4        break;
5
6       case LS_RTR: /* Router LSA */
7         if(!(edge = (*fc_ospf.fc_get_edgeent)(&fc_ospf, LS_ID(v)))){
8                        v->lsdb_edgenode = parent->lsdb_edgenode;
9                        break;
10         }
11        else{
12              /* this LS_ID is Router ID of egress node */
13              v->lsdb_edgenode = edge;
14              ifaddr = fanpdif_ospf_get_ifaddr(v, parent);
15              if(ifaddr){
16                      edge->ed_ifaddr = ifaddr;
17                      edge->ed_state = EDGE_OSPFRT_STATE_FOUNDIF;
18              }
19              break;
20       }
21
22      /* Other LSA */
23      default:
24       advrtr = ADV_RTR(v);
25       adv = FindLSA(v->lsdb_area, advrtr, advrtr, LS_RTR);
26       if(adv)
27              v->lsdb_edgenode = adv->lsdb_edgenode;
28       break;
29      }
30
```

図 4.22: 終点 CSR の検出と終点 CSR の継承に関するコード

その他の LSA であれば、この LSA を広告したルータのルータ ID を調べる。(24 行目) 次にそのルータ ID が Link State ID である Router LSA を保持する LSDB 構造体を検索する。(25 行目) この LSDB 構造体があれば、この構造体の持つ Edge 情報を継承する。

4.5 評価

OSPF リンクステートデータベースを用いたパケットフローアグリゲーションの実験・評価を、1998 年秋 WIDE 合宿 [14] で行った。この WIDE 合宿は 4 日間に渡って行われ、約 200 人程度の参加者者がこの合宿用に作られるネットワークを利用する。

以下に実験・評価の詳細を示す。

4.5.1 実験内容と評価方法

WIDE 合宿ネットワークのバックボーンにこのパケットフローアグリゲーション機能を利用できる CSR を利用した。CSR ネットワークのトポロジーを図 4.23 に示す。

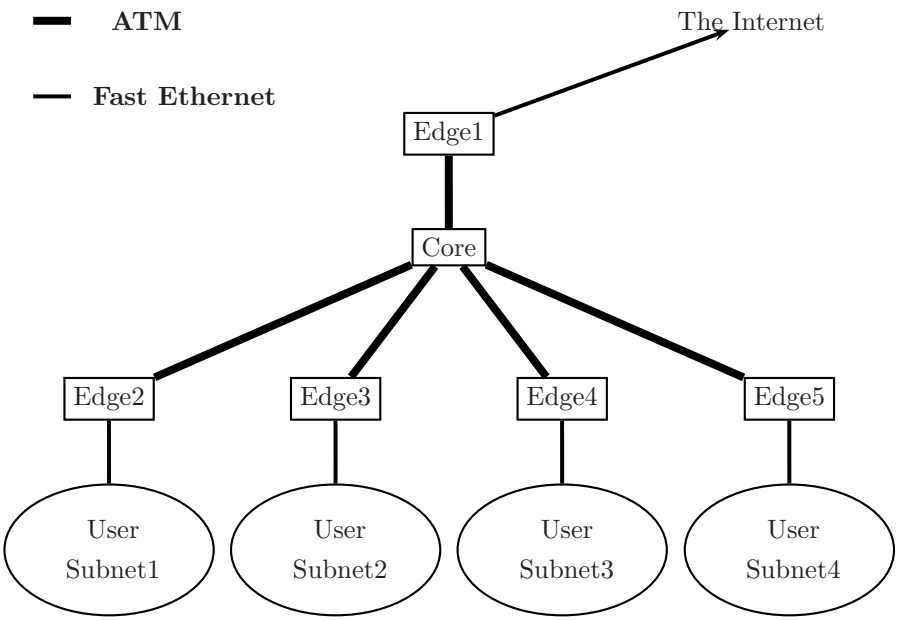


図 4.23: 合宿ネットワークトポロジ

各 Edge CSR は Core CSR に ATM を利用してポイントツーポイントで接続している。Edge2 から Edge5 は Fast Ethernet のネットワークインターフェースを持ち、利用者に接続性を DHCP によって提供する。Edge1 もまた Fast Ethernet のネットワークインターフェースを持ち、外部のネットワークへの接続性を提供している。また Edge1 は他の CSR にデフォルト経路 (0.0.0.0) を広告している。

実験内容を以下に示す。

実験 1 パケットストリームを OSPF パケットフローアグリゲーション機能を利用して、(始点 CSR, 終点 CSR) の同じパケットフローの集合としたトポロジードリブンで運用し、総 LSP 数の統計情報

を取得する。

実験 2　この機能を使用せずに、パケットストリームを送信元アドレスと宛先アドレスの同じパケットフローの集合としたトラフィックドリブンで運用した場合の利用されている総 LSP 数 (総ラベル数) の統計情報を取得する。

また、このネットワークにおいて、従来の方式で最も必要ラベル数が少ないラベル割り当て方針 を用いたと想定した場合を想定し、総 LSP 数を算出する。

以上 3 種類の総 LSP 数を比較する。

4.5.2　実験結果

実験 1

OSPF の情報を用いたパケットフローアグリゲーションを利用した場合の LSP 数の推移を図 4.24 に示す。各 Edge CSR それぞれで、他の Edge CSR に対して、それぞれ一つの LSP を割り当てる設定で運用した。そのため総 LSP 数は

Edge CSR 数 (5) * (Edge CSR 数 - 1) = 20

となった。

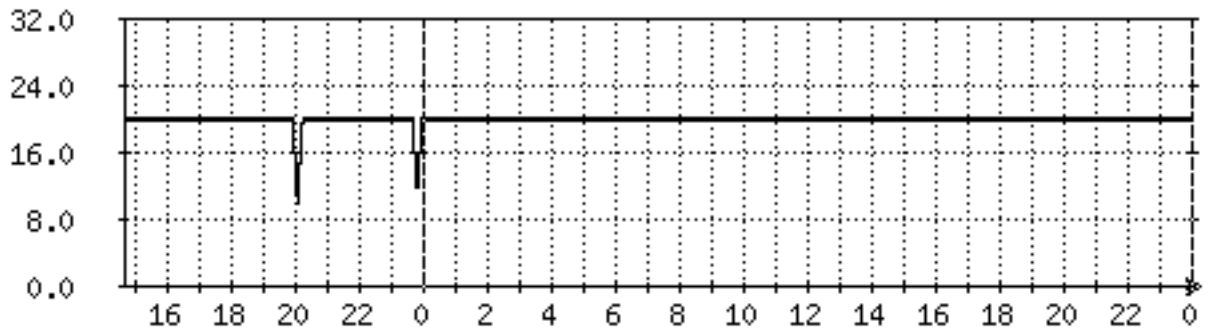


図 4.24: 実験 1 での総 LSP 数の推移

実験 2

パケットストリームを送信元アドレスと宛先アドレスの同じパケットフローの集合としたトラフィックドリブンに運用方式を変更した場合の総 LSP 数を図 4.25 に示す。LSP を設定するトリガパケットフローは、http/ftp/telnet/nntp に、60 秒間トラフィックが流れない場合はその LSP は解放される設定になっていた。

総 LSP 数はその瞬間の流れたパケットフローの数と種類に依存する。そのため早朝や食事の時間は激減している。

最大の総 LSP 数は 45 であった。

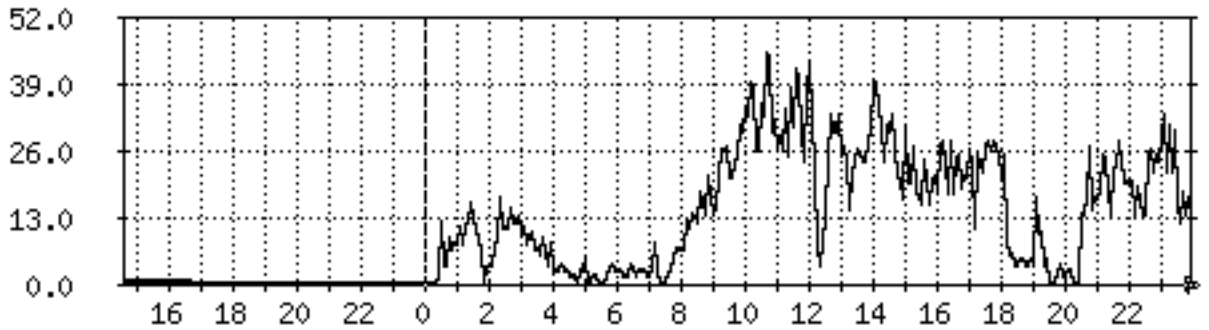


図 4.25: 実験 2 での総 LSP 数の推移

従来の方式で最も必要ラベル数の少ない運用の想定

従来の方式で最も必要ラベル数の少ないラベル割り当て方針は、パケットストリームを宛先ネットワークの同じパケットフローの集合とし、トポロジードリブンを用いることである。

このラベル割り当て方針を、このネットワークで運用した場合を想定し、総 LSP 数を算出する。

各 Edge が設定する LSP を表 4.6 にまとめて示す。総 LSP 数は少なくとも 40 である。

表 4.6: 想定されたネットワークでの総 LSP 数

Edge CSR	LSP を設定するべき IP アドレス	設定される LSP 数
Edge1	他のエッジのコア側のネットワークインターフェースの IP アドレス	4
	各ユーザネットワークのネットワークアドレス	4
	合計	**8**
Edge2,3,4,5	他のエッジのコア側のネットワークインターフェースの IP アドレス	4 * 4
	他のエッジに接続されている ユーザネットワークのネットワークアドレス	3 * 4
	デフォルト経路	1 * 4
	合計	**32**
	合計	**40**

4.5.3 実験の評価

表 4.7 に、以上の 3 つ方式の比較を示す。

表 4.7: パケットストリームの種類別の比較

パケットストリームの種類	最大ラベル数	ホップホップ転送	ラベル数が十分でない可能性	QoS 制御
1 (送信元アドレス, 宛先アドレス)	45	あり	あり	可
2 (宛先ネットワークアドレス)	40	なし (Core を除く)	あり	不可
3 (始点 LSR, 終点 LSR)	30	なし (Core を除く)	なし	不可

1 はトラフィックドリブンによって実現される場合

2,3 はトポロジードリブンによって実現される場合

この実験のネットワークにおいて、OSPF のリンクステートデータベースを利用したパケットフローアグリゲーションは、1 に対して約 56%ラベルを削減、2 に対して、50%にラベルを削減させた。

第5章

ラベルスイッチルータにおける優先転送サービス体系

第 3.2 節で述べたように、現在のラベルスイッチ技術には以下の問題がある。

1. Edge LSR での輻輳時の動作に対する規定が無い
2. LSR ネットワーク外では、優先度処理ができない

本論文では 1 の問題点に対して、IETF diffserv 分科会で議論されている優先転送サービス体系のトラフィック制御機構を Edge LSR に適用し、輻輳時のトラフィック制御の有効性を評価する。

5.1 優先転送サービス体系の概要

現在のパケット転送処理は、届いたパケットをできるだけ早く転送するものである。しかしインターネットのバックボーンルータでは、通過するパケットのために輻輳が起き、パケットが破棄される。通過するパケットは、全て同じ優先度であるため、どのパケットが破棄されるかは分からない。

このようなネットワークでは、確実にパケットを転送する配送保証や QoS 保証などのサービスを提供することは不可能である。

このような問題を解決するために優先転送サービス体系は、パケットをクラス分けし、そのクラスによってによって異なる扱いをする機構を実現する。クラス分けや一時的なトラフィック制御は、この機構を運用しているネットワークの境界ルータのみで行い、一方内部のルータではクラスによって集約されたパケットストリームに対して、適切なサービスを提供する。このような方針によって、できるだけ通過トラフィックによって負荷の掛かりやすい内部ルータに、余計な負荷を与えずに、大規模なネットワークでも利用できる機構を目指している。つまり、なるべく複雑で負荷の高い処理は、トラフィックを生成するホストに最も近いルータに行わせて、内部のルータになるべく負荷を掛けないようにしている。[19, 20, 21]

特徴を以下に示す。

- 大規模なネットワークにも適用可能 (スケーラビリティ)
- 内部のルータはパケットフロー毎の状態を管理しない
- ホップ毎のシグナリングはしない
- 提供するサービスはエンドエンドまたはドメイン内
- パケットフローに対するクラスの定義

同じ管理方針で運用されるネットワークを DS ドメイン (Differentiated Service Domain)、境界ルータを DS 境界ルータ (Differentiated Service Boundary Node)、内部のルータを DS 内部ルータ (Differentiated Service Interior Node) と呼ぶ。

DS ドメインの様子を図 5.1 に示す。

5.1.1 DS フィールド (Differentiated Service Field)

優先転送サービス体系は、IP version 4 のパケットの IP ヘッダ内 TOS フィールド値によって、ルータの転送処理 (Per Hop Behavior, PHB) を変化させるものである。この TOS フィールドを DS フィールドと呼ぶ。図 5.2 に DS フィールドのフォーマットを示す。DS フィールドの先頭 6 ビットを DSCP(Differentiated Service Code Point) と呼び、残りの 2 ビットを CU(Currently Unused) と呼ぶ。DSCP は PHB を選ぶために利用される。CU は現在利用されていない。DSCP に適切な値をいれることをマーキング (Marking) と呼び、DSCP の値を DS コードと呼ぶ。DS コードが 000000 の場合、デフォルトの PHB が選択される。

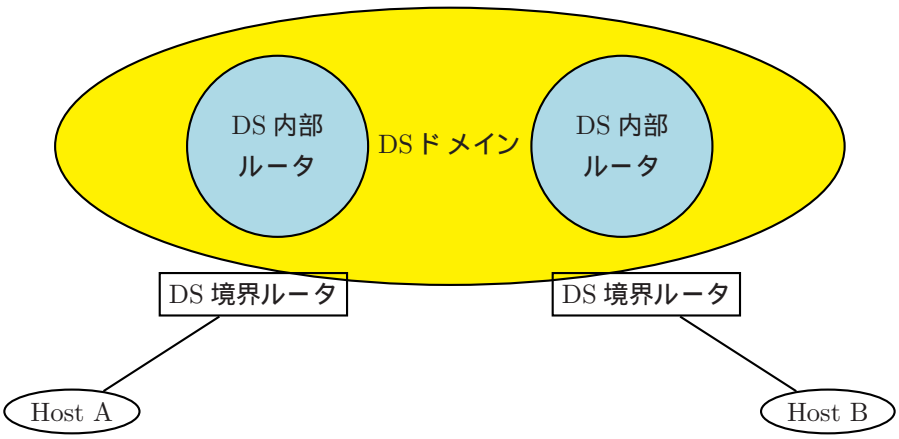


図 5.1: 優先転送サービス体系ネットワーク

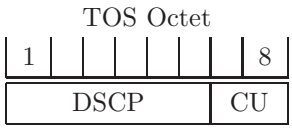


図 5.2: DS フィールドのフォーマット

5.1.2 優先転送サービス体系の機構

優先転送サービス体系を実現するためには、ルータに以下の機能を実装することが求められる。

- PHB(Per Hop Behaviors) の集合
- パケットのクラス分け機構 (Pachet Classifier)
- トラフィック調整機構 (Traffic Conditioner)

トラフィック調整機構は以下の機能から構成される。

測定機構 (Meter) パケットストリームの一時的な状態を測定する。それぞれのパケットが規定帯域 (Profile) 以内か、超過しているか他の機能に知らせる機能

マーカー (Marker) DS フィールドに適切な値を設定する機能

遅延機構 (Shaper) 規定帯域を超過したパケットを、規定帯域に沿うように遅らせる機能

破棄機構 (Dropper) 規定帯域を超過したパケットを、破棄する機能

図 5.3 に DS 境界ルータでの制御機構を示す。パケットは、クラス分け機構によってクラス分けされる。クラス分け機構は測定機構に知らせ、測定機構は、そのパケットの属するトラフィックの一時的な状態 (利用帯域など) を測定し、マーカー、遅延機構/破棄機構に知らせる。クラス分け機構からパケットを渡されたマーカーは、クラスと測定機構の情報を基に適切な DS コードをマークする。マーカーからパケットを渡された遅延機構/破棄機構は、測定機構の情報と、そのパケットに定められた規定帯域を基に、パケットを遅延/破棄するか決定する。破棄されなかったパケットは、転送される。

優先転送サービス体系は大規模なネットワークに適応可能にするために以下の項目を達成する。

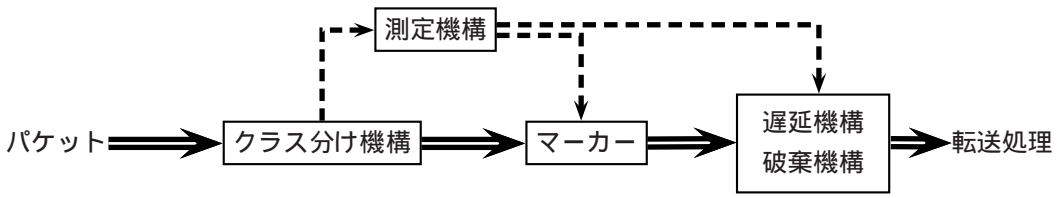


図 5.3: DS 境界ルータでのパケット処理

- 複雑なクラス分け機構とトラフィック調整機構を DS 境界ルータにのみ動作させる
- DS コードが同じトラフィックの集約に対して PHB を動作させる

現在 PHB として、AF PHB グループ (Assured Fowarding PHB Group)[24] と EF PHB(Expedited Forwarding PHB)[23] が提案されている。以下にその定義を示す。

EF PHB

EF PHB は、トラフィックを低遅延、低ジッタで転送し、また帯域保証をする PHB である。推奨される DS コードは 101110 である

AF PHB グループ

AF PHB グループは、N 個の独立した AF クラスによって、パケットを配送する。AF クラス内では M 個の独立した破棄優先度を、パケットに割り当てる。現在 N は 4、M は 3 と定義されている。各 AF クラスは互いに独立したキューによって実現される。

ルータは各 AF クラスに対して、帯域、バッファなどの転送資源を割り当てる。$x < y$ である場合、AF クラス x は AF クラス y よりも転送資源を割り当てられてはならない。また同様に、破棄優先度 x のパケットは破棄優先度 y のパケットよりも、高い確率で破棄されてはならない。

表 5.1 に AF PHB グループの DS コードを示す。

表 5.1: AF PHB グループの DS コード

破棄優先度	AF クラス 1	AF クラス 2	AF クラス 3	AF クラス 4
1	0010000	011000	100000	101000
2	0010010	011010	100010	101010
3	0010100	011100	100100	101100

5.2 実装

優先転送サービス体系を実現するために、ALTQ[25](altq-1.1.2) を CSR に実装した。ALTQ とは、BSD 系 UNIX の汎用キューイング・インターフェースであり、既に CBQ(Class-Based Queueing)[27]、

RED(Random Early Detection)[18]、RIO(RED In and Out)[19]、 WFQ(Weighted Fair Queueing)[26]などのキューイング方式が実装されている。

5.3 実験と評価

5.3.1 実験の目的

現状では Edge LSR での輻輳時の動作に対する規定が無い。つまり、データリンク層での優先度がここで評価されていないために、LSR ネットワーク全体で、パケットフローに対しての優先度を実現することができない。

優先転送サービス体系のトラフィック制御機構を Edge LSR に適用した場合、輻輳時のトラフィック制御に対する有効性を評価する。

5.3.2 実験に用いた機構

以下にこの実験で、優先転送サービス体系を実現するために用いた機構を説明する。

パケットのクラス別の優先度や帯域の分割のために、CBQ(Class-Based Queue)[27] を用いた。CBQ はリンク共有サービスや、リアルタイムアプリケーションのトラフィックに対するサービスを提供するキューである。

クラス構造を基本として、階層的なクラス構造を作ることができ、クラスは独立したキューを持つ。ルートのクラスはリンクの利用可能な帯域全てを持ち、下位のクラスに帯域を割り当てる。下位のクラスは上位のクラスから帯域を借りることが可能で、上位のクラスの使用中の帯域に余裕がある場合は、下位のクラスは割り当てられた以上の帯域を利用することができる。

しかし輻輳が起きた時、他のクラスが割り当てられた以上に帯域を使用しているために、割り当てられた以下の帯域しか使えないクラスがある場合は、割り当てられた以上に帯域を使用しているクラスは割り当てられた帯域まで調整される。

どのクラスにも属さないパケットは、最上位のクラスの帯域を利用して転送される。

CBQ を用いたリンク共有の例を、図 5.4 を用いて、以下に説明する。この例は、一つのレベルのリンク共有を表し、表 5.2 のように設定されている。ルート ノードであるリンクは、利用可能な帯域全てを表す、ルートクラスである。また、各子ノードは、上位から割り当てられた帯域を持つクラスである。この例では、video クラスはリアルタイムビデオストリームを転送するためのクラスであり、全体の帯域の 30%を割り当てられている。同様に、telnet クラス、ftp クラスはそれぞれ、全体の帯域の 50%、20%を割り当てられている。mail クラスは帯域を割り当てられていないため、上位の帯域が余り、借り入れ可能な場合のみ転送される。例えば mail クラス以外の 3 つのクラスが割り当てられた帯域を全て利用していない場合を考えると、リンククラスは帯域に余裕があるために、mail クラスに属するパケットを転送する。しかし輻輳輻輳が起きた場合、各クラスはそれぞれに割り当てられた帯域まで制限されるため、帯域の割り当てられていない mail クラスに属するパケットは転送されない。

また、CBQ では優先スケジューラと同様の機能を持ったスケジューラを使用しており、表 5.2 の様な優先度を各クラスに持たせることができる。リアルタイムアプリケーションのトラフィックや、対話型トラフィックには高い優先度を持たせている。video クラスに属するパケットは最も優先されて転送されるが、上位からの借り入れを許していないため、割り当てられた帯域に制限される。上位からの借り入れを許している telnet クラスに属するパケットは、ftp クラス、mail クラスに属するパケットよりは優先して転送されるが、従来の優先スケジューラと異なり、割り当て帯域が効くために低優先度クラスが、高優先度クラスによって、割り当て帯域を利用できなくなることはない。同一優先度のクラス間ではウ

エイテッド・ラウンドロビン・スケジューリングとなる。

より複雑なリンク共有として、CBQ は図 5.5 のような階層的なリンク共有が可能である。この例では、ルートクラスであるリンククラスからプロトコルファミリ毎に帯域を割り当て、更にそれぞれのプロトコルファミリのクラス内で、下位クラスに帯域を割り当てている。

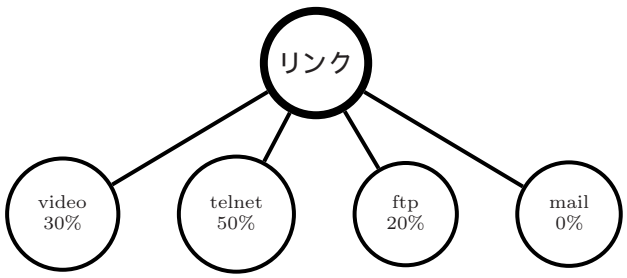


図 5.4: CBQ を用いたリンク共有の例

表 5.2: 図 5.4 の設定

クラス	割り当て帯域	上位クラスからの借り入れ	優先度
video	30%	不可	5
telnet	50%	可	3
ftp	20%	可	2
mail	0%	可	2

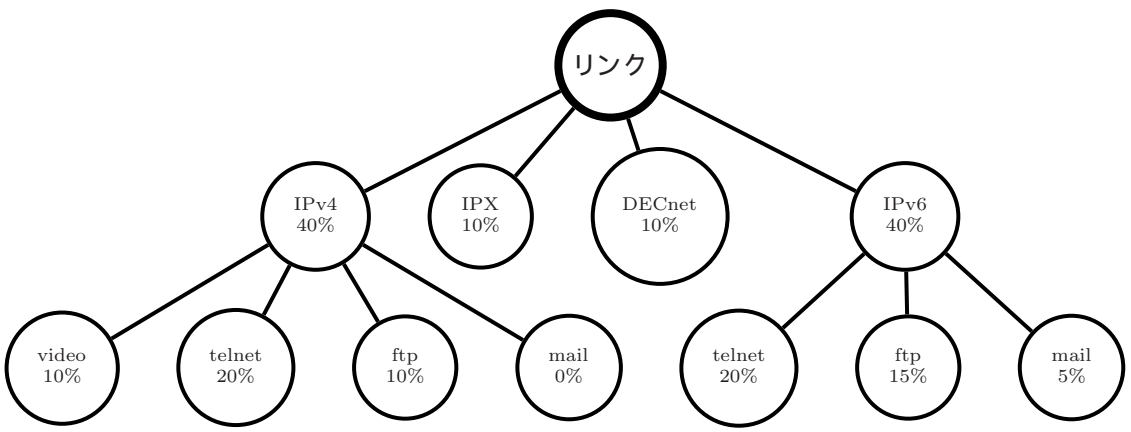


図 5.5: CBQ を用いた階層的なリンク共有の例

CBQ の各クラス内のキューは FIFO キューを用いている。ALTQ は、CBQ の各クラスのキューに RED や RIO を用いることができる。

5.3.3 実験環境

CBQ を用いたトラフィック制御機構を Edge CSR に適用し、輻輳時のトラフィック制御の有効性を評価するための実験について以下に説明する。

実験は図 5.6 に示すネットワークで行った。

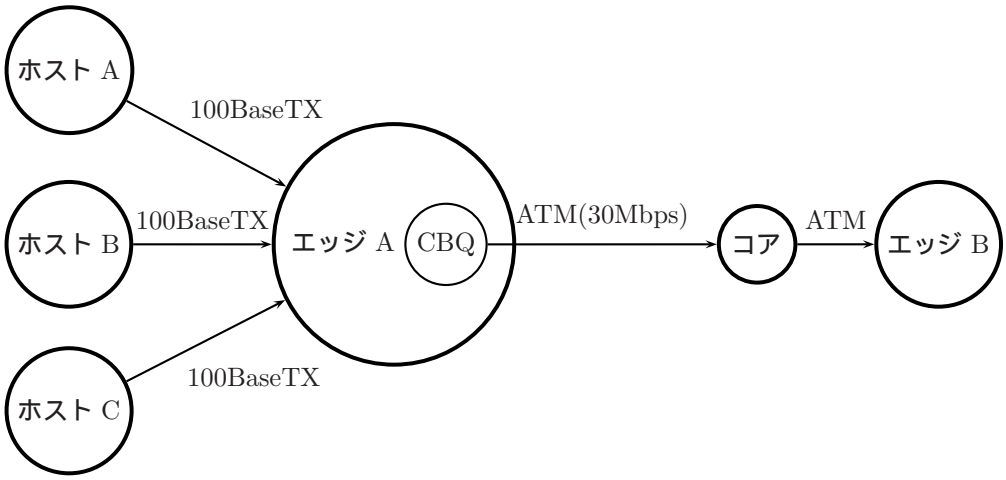


図 5.6: 実験トポロジ

コアは Core CSR であり、エッジ A、エッジ B と 155Mbps の ATM で接続されている。エッジ A、エッジ B は Edge CSR であり、エッジ A にはホスト A、ホスト B、ホスト C がそれぞれ 100BaseTX で接続され、全二重モードに設定されている。またエッジ A は、コアと接続されたネットワークインターフェースに CBQ を利用し、輻輳を容易に起こすためにコアと接続している ATM をコア方向だけ 30Mbps に帯域を制限 (シェーピング) している。

ホスト A、ホスト B、ホスト C、エッジ A、エッジ B は、PentiumPro マシン (180MHz, Memory 64Mbytes) を用い、コアには Pentium マシン (120MHz, Memory 32Mbytes) を用いた。

実験全てを通して、スループットの測定には netperf[28] の TCP スループット測定機構を用いた。

5.3.4 実験 1: CBQ のオーバーヘッドの測定

ホスト A からコアへのスループットを、エッジ A のコア側のネットワークインターフェースに従来の FIFO キューを用いた場合と CBQ を用いた場合について測定し、CBQ を用いた場合のオーバーヘッドを測定した。

CBQ のリンク共有を図 5.7 に、各クラスの設定を表 5.3 に示す。TCP クラスは TCP のパケットが属するクラスであり、UDP クラスは UDP のパケットが属するクラスである。

TCP のトラフィックは、割り当てられた帯域が 60%であるが、上位クラスからの借り入れができるため他のトラフィックが流れていない場合はリンク帯域全てを利用できる。

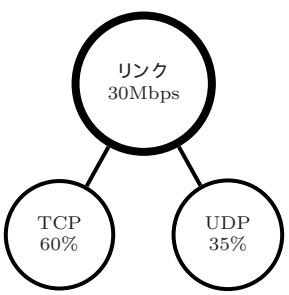


図 5.7: 実験 1 でのリンク共有

表 5.3: 実験 1 での CBQ の設定

クラス	割り当て帯域	上位クラスからの借り入れ	優先度
TCP	60%	可	1
UDP	35%	可	1

測定結果を表 5.4 に示す。ほとんど CBQ によるオーバーヘッドはないことが分かった。

表 5.4: CBQ を用いるオーバーヘッド

FIFO (Mbps)	CBQ (Mbps)	overhead (%)
29.73	29.71	0.07

5.3.5 実験 2：送信元ホストによるクラス分け

送信元ホストによって異なるサービスを提供するために、CBQ に図 5.8、表 5.5 に示す設定をし、ホスト A とホスト B からコアへのスループットを同時に測定した。クラス A は送信元ホストがホスト A であるパケットの属するクラスであり、クラス B は送信元ホストがホスト B あるいはホスト C であるパケットの属するクラスである。CBQ では転送したパケット数やバイト数などの統計情報を取得した。また同様の測定を、クラス A とクラス B を同優先度にした場合と、CBQ の代わりに FIFO キューを用いた場合で行った。

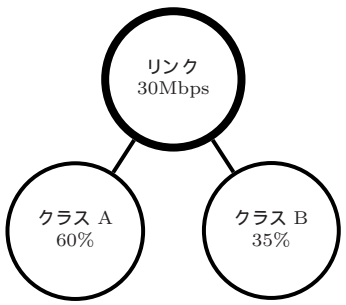


図 5.8: 実験 2 でのリンク共有

実験結果を表 5.6 に示す。FIFO キューを用いた場合は二つ送信元ホストは、ほぼ同じスループットを示した。一方 CBQ を用いた場合は、送信元ホストによってスループットが変化している。クラス A とクラス B を同じ優先度にした場合は、余剰帯域の 5%がクラス B に割り当てられている。しかしクラス A にクラス B よりも高い優先度を持たせた場合は、余剰帯域の 5%がクラス A に割り当てられている。

図 5.9 に CBQ において測定したスループットを示す。表 5.6 と同様の結果が得られた。

5.3.6 実験 3：特性の異なるトラフィック間でのリンク共有

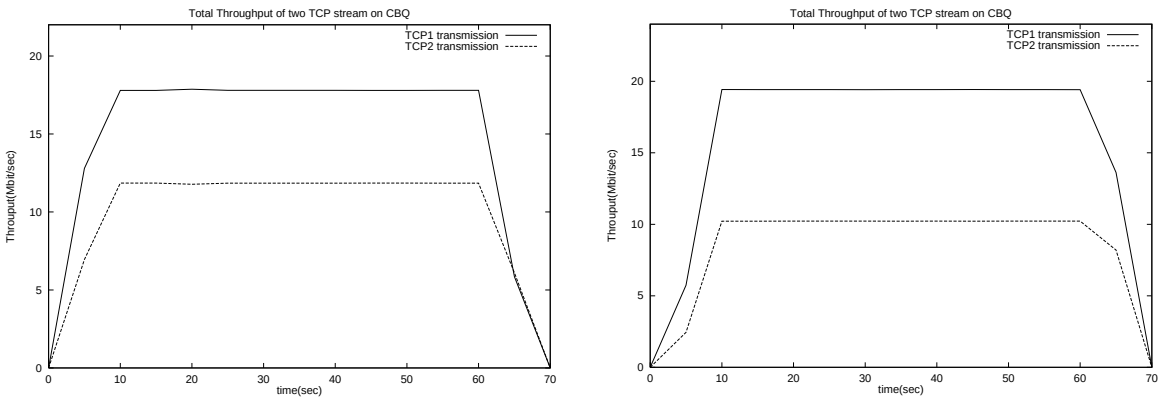
実験 3 では、特性の異なる 2 つのプロトコルのトラフィック間で、リンク共有がどのように行われるか評価する実験を行った。

表 5.5: 実験 2 での CBQ の設定

クラス	割り当て帯域	上位クラスからの借り入れ	優先度
クラス A	60%	可	2
クラス B	35%	可	1

表 5.6: 送信元ホスト別 TCP のスループット

	FIFO キュー		同じ優先度の CBQ		異なる優先度の CBQ	
送信元ホスト	Mbps	%	Mbps	%	Mbps	%
ホスト A	15.02	50	17.94	59.8	19.49	65
ホスト B	15.05	50	12.09	40.3	10.39	34.7



同じ優先度の設定　　異なる優先度の設定

図 5.9: 実験 2 の CBQ におけるスループットの測定

ホスト B から約 21Mbps の UDP のバーストトラフィックをエッジ B に対して生成した状態で、FIFO キューを用いた場合と実験 1 と同様の設定の CBQ を用いた場合でホスト A からコアへのスループットを測定した。特に FIFO キューを用いる場合は、キューが保持できるパケット数 (キュー長) を 50 と 100 の場合の二通り測定を行った。

また、CBQ を用いた場合のリンク共有のトレースを取るために、CBQ の統計情報を収集した。

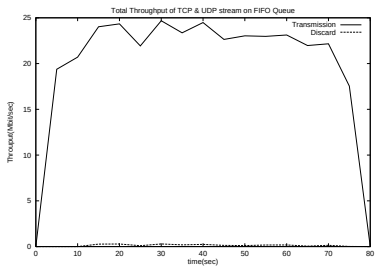
実験結果を表 5.7 に示す。キュー長が 50 の FIFO の場合は、TCP のスループットが最も低い。図 5.10 に各キューの様子の様子を示すが、キュー長が 50 の FIFO の場合は、10 秒目からパケット破棄が発生している。これはバーストトラフィックのために、キューが一杯になったためである。UDP とは異なり、TCP は輻輳制御を持つプロトコルであるために、転送レートを下げてしまう。そのために結果的にスループットが低くなってしまったと考えられる。

従ってキュー長が 100 の場合は、十分にバーストトラフィックを許容できたために、スループットは、キュー長が 50 の場合よりも良い結果である。

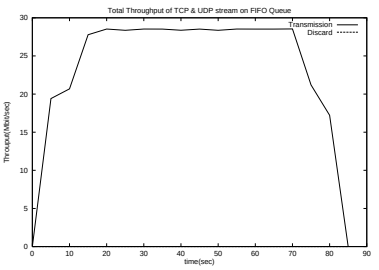
CBQ を用いた結果によって、TCP のトラフィックが、UDP のトラフィックに対して保護されていることが分かる。

表 5.7: キュー毎の TCP スループット

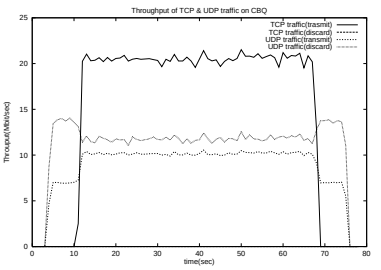
キュー	TCP のスループット
FIFO(キュー長:50)	2.69Mbps
FIFO(キュー長:100)	7.81Mbps
CBQ	19.67Mbps



FIFO(キュー長が 50 の場合)



FIFO(キュー長が 100 の場合)



CBQ

図 5.10: キューの様子

5.3.7 実験 4 : 階層的なリンク共有

実験 2 と同様にホスト A とホスト B に対して提供するサービスを異なるものにする。同時に実験 4 では階層的なクラス分けを用いることで、トラフィック特性の異なるトラフィックを他のトラフィックに影響を与えないようにする。設定を図 5.11、表 5.8 に示す。

Agency A はホスト A からのトラフィックのためのクラスであり、子クラスである TCP A と UDP A はホスト A からの TCP トラフィック、UDP トラフィックのためのクラスである。同様に Agency B はホスト B からのトラフィックのためのクラスであり、子クラスの TCP B と UDP B はホスト B からの TCP トラフィック、UDP トラフィックのためのクラスである。

この CBQ に対して発生させるトラフィックの発生パターンを、図 5.12 に示す。

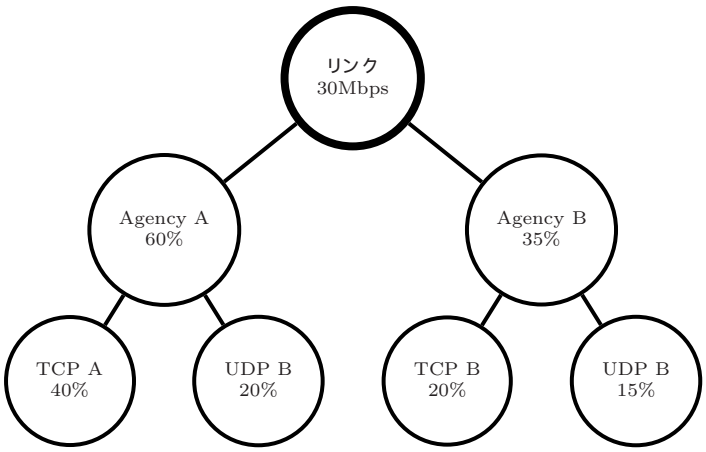


図 5.11: 実験 4 でのリンク共有

表 5.8: 実験 4 での CBQ の設定

クラス	割り当て帯域	上位クラスからの借り入れ	優先度
TCP A	40%	可	1
UDP A	20%	可	1
TCP B	20%	可	1
UDP B	15%	可	1

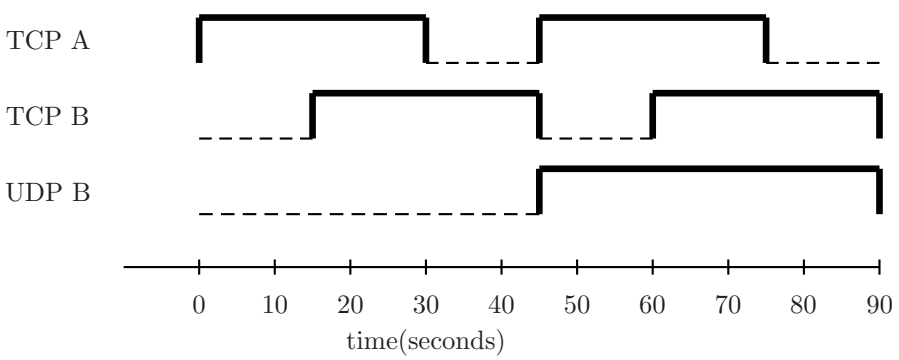


図 5.12: トラフィック発生パターン

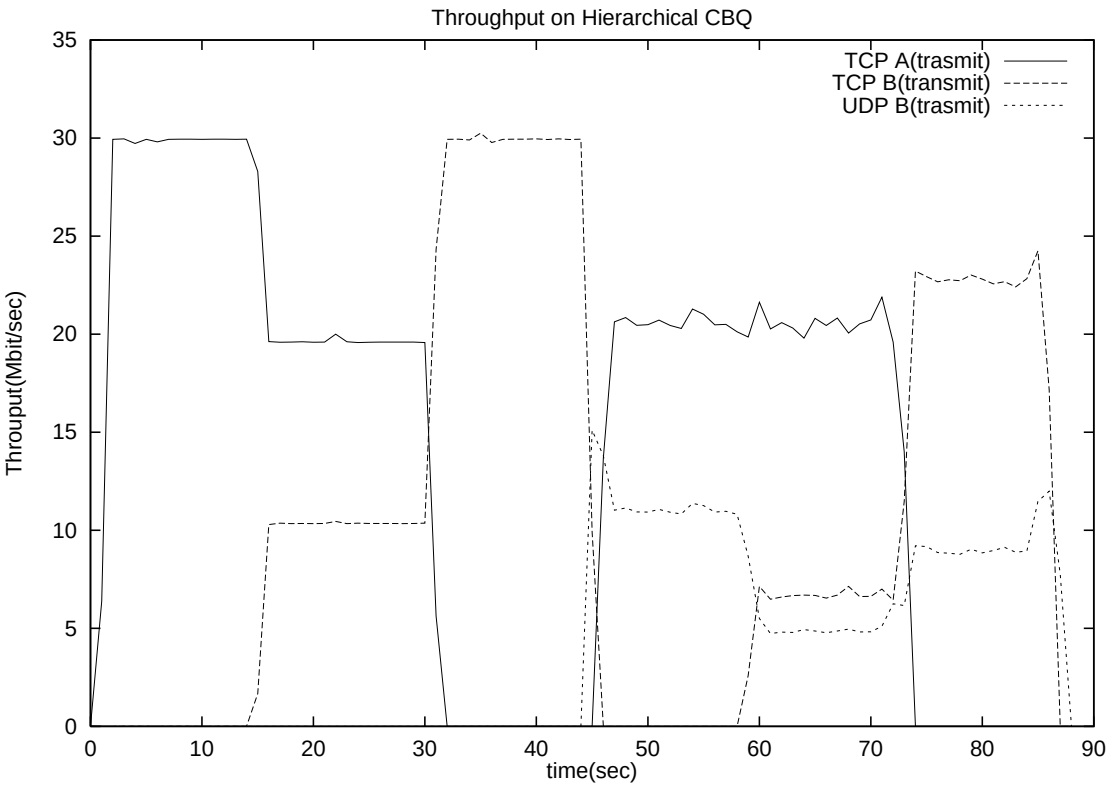


図 5.13: 階層的なクラスを持った CBQ の様子

CBQ でのリンク共有の様子を図 5.13 に示す。ほぼ期待通りの動作をしていることが分かる。

第6章

結論

6.1 結論

6.1.1 OSPF リンクステートデータベースを用いたパケットフローアグリゲーション

以下に第 4 章で実装したパケットフローアグリゲーションに関しての結論をまとめる。
LAN 環境において、以下の二つのラベル割り当て方針による運用をし、ラベル数を比較した。

1. トラフィックドリブン/パケットストリームは送信元ホスト/宛先ホストが同じパケットフローの集合
2. トポロジードリブン/パケットストリームは始点 LSR/終点 LSR が同じパケットフローの集合

1 では最大ラベル数は 45 であり、2 では 20 であった。2 を用いることによって 56%のラベルを削減することができた。

2 の手法は瞬間のパケットフロー数や経路エントリ数では無く、Edge LSR 数に依存する。Edge LSR 数を N とすると、以下のオーダーで増加する。

$$\text{総ラベル数} = O(N^2)$$

大規模なネットワークにおいても、従来の方式に比べてラベル数が十分でないという問題を引き起こしにくい。

6.1.2 優先転送サービス体系

第 5 章では Edge LSR での輻輳時のトラフィック制御機構について実験・評価した。

CBQ では、トラフィックの送信元ホストによって、または同時にプロトコルによっても異なるサービスを提供できる。このような機構を Edge LSR に利用し、ラベルスイッチ技術における優先度の方針と一致させることで、LSR 網内でのデータリンク層における優先度を Edge LSR の送出インターフェースにおいても反映させられることが分かった。

6.2 今後の課題

6.2.1 Opaque LSA による Edge LSR の検出

本論文の実装では、図 4.18 の様に、終点 CSR のルータ ID は静的に設定した。

今後の課題として、Opaque LSA を利用して、自動的に他の Edge CSR のルータ ID を検出する方法が考えられる。

以下に Opaque LSA に関する説明と考えられる利用法について述べる。

表 6.1 の示すように、Opaque LSA は LS Type フィールドによって 3 種類に分けられる。これらの相違は、この LSA が流出する範囲を制限するものである。

Opaque LSA のフォーマットを図 6.1 に示す。

各 Edge CSR が定期的に自分のルータ ID を Opaque Infomation フィールドに含め、広告することで、自動的に他の Edge CSR の存在を認識することができる。パケットフローアグリゲーションできるのは、最短経路ツリーの範囲に依存するため、同一エリア内に制限される。そのため、LS Type 10 の Opaque LSA を利用するべきである。

表 6.1: Opaque LSA の種類と意味

LS Type	scope	意味
9	Link local	広告ルータの、この LSA を出力したリンク上にある ルータのみに広告される。他のリンクには流出しない。
10	Area local	広告ルータの、この LSA を出力したリンクの属するエリア 内のルータのみに広告される。他のエリアには流出しない。
11	AS	OSPF で運用されるド メイン内の全てのルータに広告される

広告ルータ: この LSA を生成したルータ

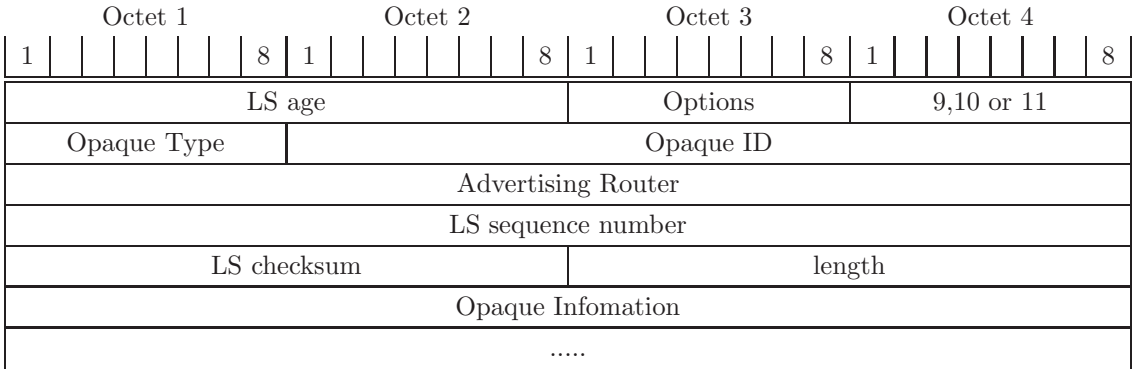


図 6.1: Opaque LSA

6.2.2 エンド エンド 通信の評価

本論文では、特に Edge LSR におけるトラフィック制御によって、ラベルスイッチ技術における優先度の反映することができた。今後の課題として、LSR 網におけるエンド エンド 通信の優先度評価である。

例を、図 6.2 に示す。

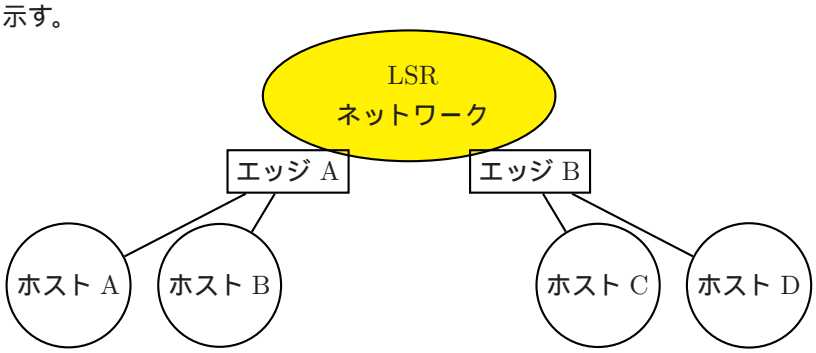


図 6.2: エンド エンド 通信の優先度評価

エッジ A において、ホスト A からホスト C へのパケットには高い優先度を、ホスト B からホスト D へのパケットには低い優先度を与えた場合、現実的にどの程度異なるサービスを提供できるかを評価する。

このネットワークには、図 6.3 に示すような機構を用いることができると考えられる。エッジ A からエッジ B へ二つの優先度の異なる LSP が設定される。これらの LSP は、第 4 章で実現したパケットフローアグリゲーションを用いて設定される。

ホスト A からホスト C へのパケットは CBQ によって、あらかじめ決められた規定帯域が保証され、高い優先度の LSP を通ってエッジ B へ転送されるように設定する。一方ホスト B からホスト D へのパケットは低い優先度として CBQ で制御され、低い優先度の LSP を通ってエッジ B へ転送されるように設定する。

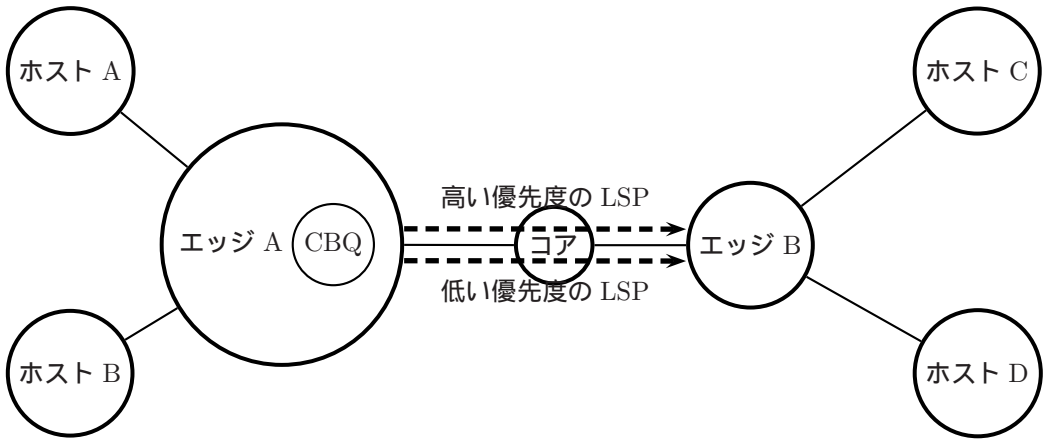


図 6.3: 異なる優先度の LSP を設定する例

この例では、二つのエンド エンド 通信は、互いに影響を与えるトラフィックがエッジ A の CBQ で制御されるために、ほぼ第 5.3 節と同様の結果を得ると推測できる。

しかし、コアのデータリンク層転送モジュールのエッジ B 側インターフェースにおいて、他のエッジからのトラフィックが同時に転送される場合に、どうデータリンク層転送モジュールが制御するかが問題になる。図 6.4 を例に説明する。

このネットワークにおいて、それぞれの 3 つのエッジ間には、2 レベルの優先度の OSPF を用いた LSP が互いにそれぞれ設定されている。また、エッジ A、エッジ C のコア側のインターフェースには CBQ が実装され、2 レベルの優先度が設定されている。

送信元ホストがホスト A であるパケットフローの集合は、エッジ A によって高い優先度の LSP を通ってエッジ B に転送され、送信元ホストがホスト B であるパケットフローの集合は、低い優先度の LSP を通ってエッジ B に転送されるように設定され、また送信元ホストがホスト E であるパケットフローの集合は、エッジ C によって高い優先度の LSP を通ってエッジ B に転送されるように設定されているとする。

エッジ A からエッジ B へのトラフィックとエッジ C からエッジ B へのトラフィックは、コアのデータリンク層転送モジュールのエッジ B 側のインターフェースで合流する。この場合、どのように優先度が制御されるかは、データリンク層転送モジュールのパケットスケジューリング機構に依存する。

データリンク層転送モジュールが、基本的な優先度スケジューリングをする場合は、優先度の高いトラフィックが、そのリンク帯域を越えるぐらい流れて来た時に、優先度の低いトラフィックに対して帯域を提供できなくなる。

ATM をデータリンク層転送技術に用いた LSR において、このような問題は ATM スイッチに依存するため、他のエッジからのトラフィックを交えたエンドエンド通信の定量的な評価はできない。

データリンク層転送モジュールも交えた、優先転送サービス体系の実現は今後の課題である。

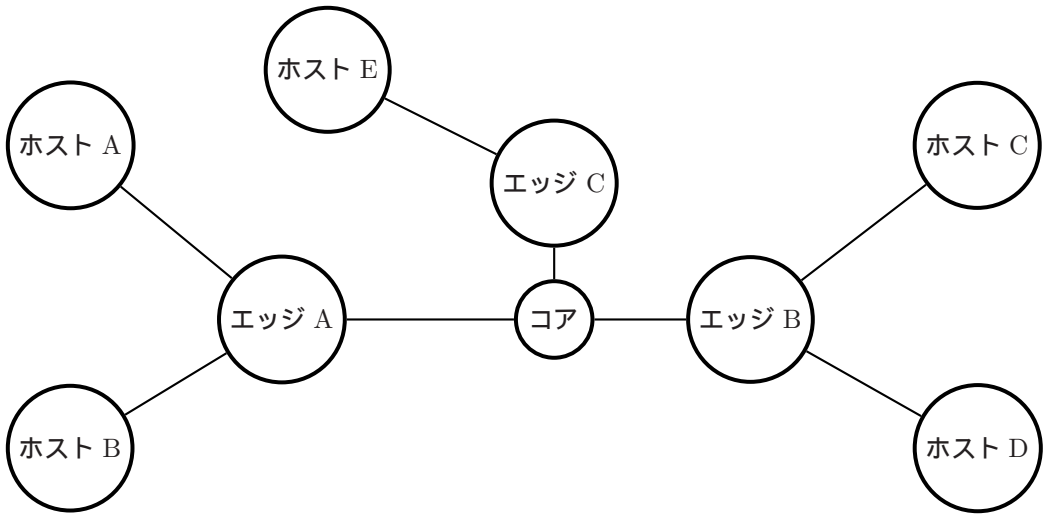


図 6.4: 他のトラフィックが影響する例

第7章

謝辞

本研究を進めるにあたり、指導教員である慶應義塾大学環境情報学部教授村井純博士と徳田英幸博士に感謝します。

絶えず指導して下さった慶應義塾大学環境情報学部講師中村修博士、東芝研究開発センターの永見健一氏、東京大学大型計算機センターの江崎浩助教授と加藤朗助手、ALTQ に関する助言を下さった SONY CSL 研究所の長健次郎氏に感謝します。

暖かく見守って、励ましてくれた慶應義塾大学政策・メディア研究科の杉浦一徳氏、土本康生氏、重近範行氏、小川晃通氏、中澤仁氏、他の学部生に感謝します。

関連図書

[1] 永見健一, 江崎浩, 勝部泰弘, 中村修, “ラベルスイッチルータにおける必要ラベル数の評価とフローアグリゲーションの効果”, インターネットカンファレンス’98, December 1998

[2] Y.Katsube, K.Nagami, H.Esaki, “Toshiba’s Router Architecture Extensions for ATM : Overview”, IETF RFC2098, Feb 1997

[3] K.Nagami, Y.Katsube, Y.Shobatake, A.Mogi, S.Matsuzawa, T. Jinmei, H.Esaki, “Toshiba’s Flow Attribute Notification Protocol(FANP)”, April 1997

[4] K.Nagami, Y.Katsube, S.Matsuzawa, A.Mogi, Koutarou, “Toshiba’s Flow Attribute Notification Protocol Version 2(FANPv2) Distributed Control Mode”, December 1997

[5] Y.Ohba, K.Nagami, S.Matsuzawa, Y.Katsube, “Toshiba’s Flow Attribute Notification Protocol Version 2(FANPv2) Ingress Control Mode”, December 1997

[6] Y.Rekhter, B.Davie, D.Katz, E.Rosen, G.Swallow, “Cisco Systems’s Tag Switching Architecture Overview”, IETF RFC2105, Feb 1997

[7] P.Doolan, B.Davie, D.Katz, Y.Rekhter, E.Rosen, “Tag Distribution Protocol”, IETF Internet-Draft, May, 1997

[8] P.Newman, T.Lyan, G.Minshall, “Flow labelled: Connectionless ATM Under IP”, Engineer COnference, Networld+Interop ’96 Las Vegas, April 1996

[9] P.Newman, W.Edwards, R.Hinden, E.Hoffman, F.Ching Liaw, T.Lyon, G.Minshall, “Ipsilon Flow Management Protocol Specification for IPv4 Version 1.0”, May 1996

[10] A.Viswanthan, N.Feldman, R.Boivie, R.Woundy, “ARIS: Aggregate route-based IP switching”, IETF Internet-Draft, March, 1997

[11] L.Andersson, P.Doolan, N.Feldman, A.Fredette, R.Thomas, “Label Distribution Protocol”, IETF Internet-Draft, March, 1998

[12] John Moy, “Open Shortest Path First(OSPF) version 2”, RFC2328, April 1998

[13] R.Coltun, “The OSPF Opaque LSA Option”, RFC2370, July 1998

[14] http://www.wide.ad.jp

[15] A.Fredette, C.White, L,Andersson, P.Doolan, “Stream Aggregation”, IETF Internet-Draft, November 1997

[16] E.Rosen, A.Viswanathan, R.Callon, “Multiprotocol Label Switching Architecture”, IETF Internet-Draft, July 1998

[17] L.Andersson, P.Doolan, N.Feldman, A.Fredette, B.Thomas, "LDP Specification", IETF Internet-Draft, November 1998

[18] V.Jacobson S.Floyd, "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, 1993

[19] D.Clark, W.Fang, "Explicit Allocation of Best Effort Packet Delivery Service", IEEE/ACM Trans. on Networking, vol.6, no.4, August 1998, pp. 362-373

[20] K.Nichols, V.Jacobson, L.Zhang, "A Two-bit Differentiated Services Architecture for the Internet", ftp://ftp.ee.lbl.gov/papers/dsarch.pdf, November 1997

[21] S.Blake, D.Black, M.Carlson, E.Davies, Z.Wang, W.Weiss, "An Architecture for Differentiated Service", IETF RFC2475, December 1998

[22] K.Nichols, S.Blake, F.Baker, D.Black, "Difinition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", IETF RFC2474, December 1998

[23] V.Jacobson, K.Nichols, K.Poduri, "An Expedited Forwarding PHB", IETF Internet-Draft, November 1998

[24] J.Heinanen, F.Baker, W.Weiss, J.Wroclawske, "Assured Forwarding PHB Group", IETF Internet-Draft, Nevember 1998

[25] Kenjiro Cho, "Alternate queueing for bsd unix", http://www.csl.sony.co.jp/person/kjc/programs.html, 1997

[26] A.Demers S.Keshav S.Shenker, "Analysis and Similation of Fair Queueing Algorithm", Proc. SIGCOMM '89, 1989

[27] S.Floyd, V.Jacobson, "Link-sharing and Resource Management Models for Packet Networks", IEEE/ACM Transactions on Networking, Vol.3 No.4, August 1995

[28] http://www.cup.hp.com/netperf/NetperfPage.html

第8章

Appendix

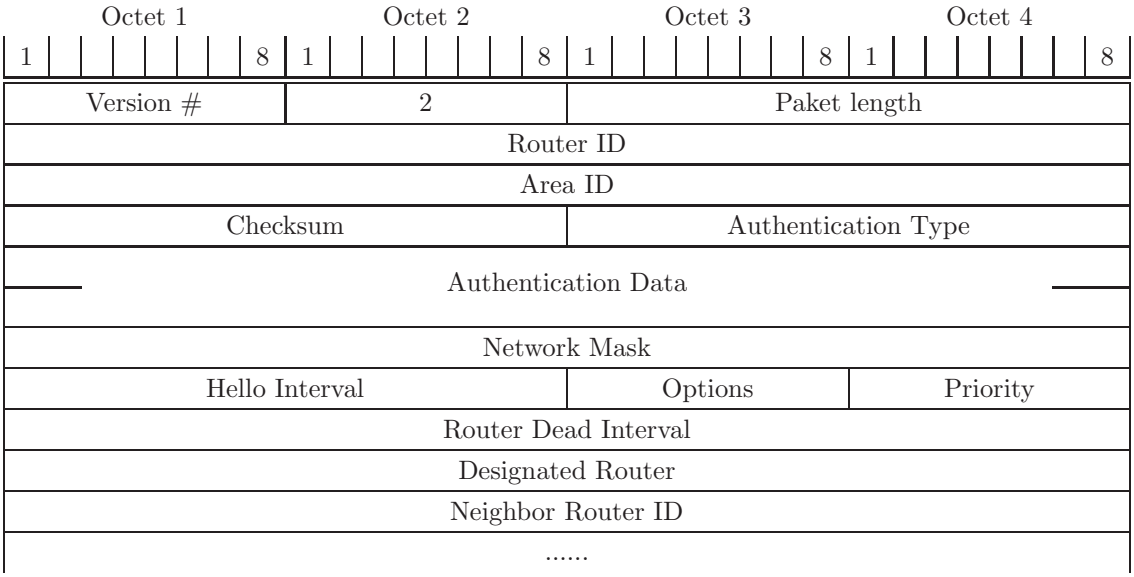


図 8.1: Hello Packet

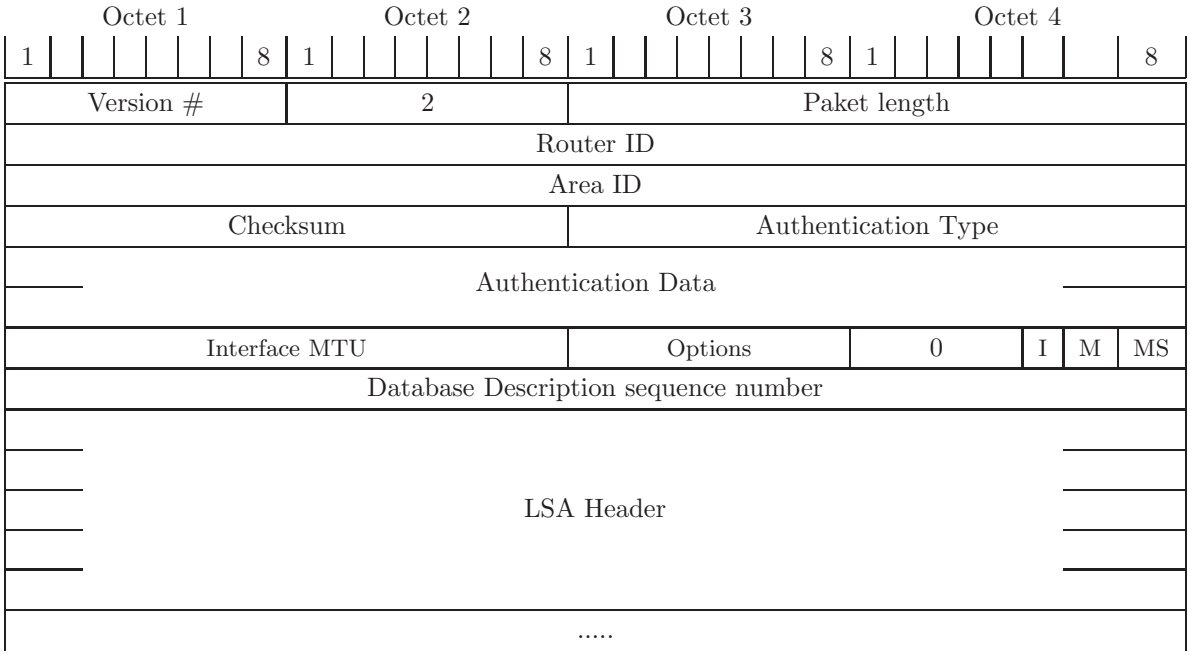


図 8.2: Database Description Packet

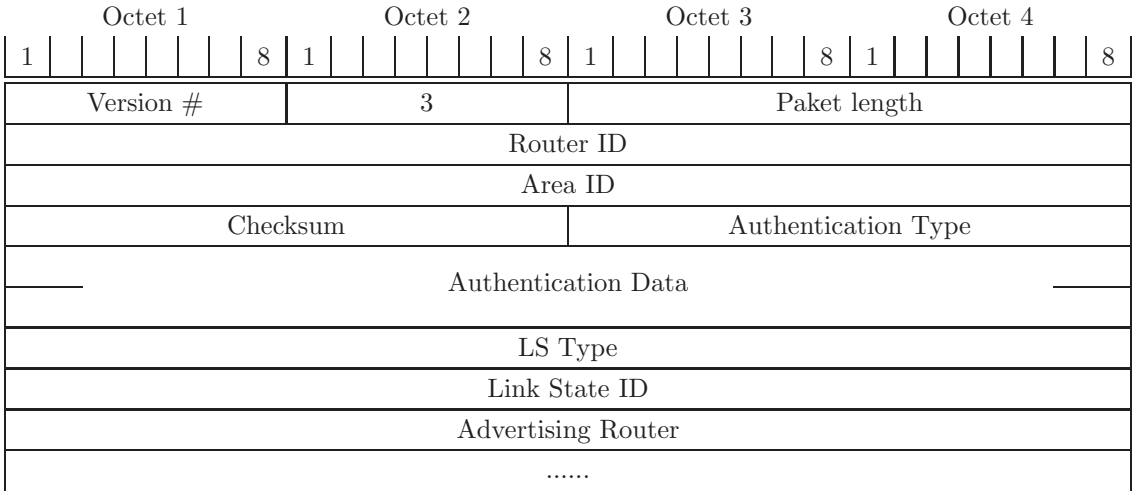


図 8.3: Link State Request Packet

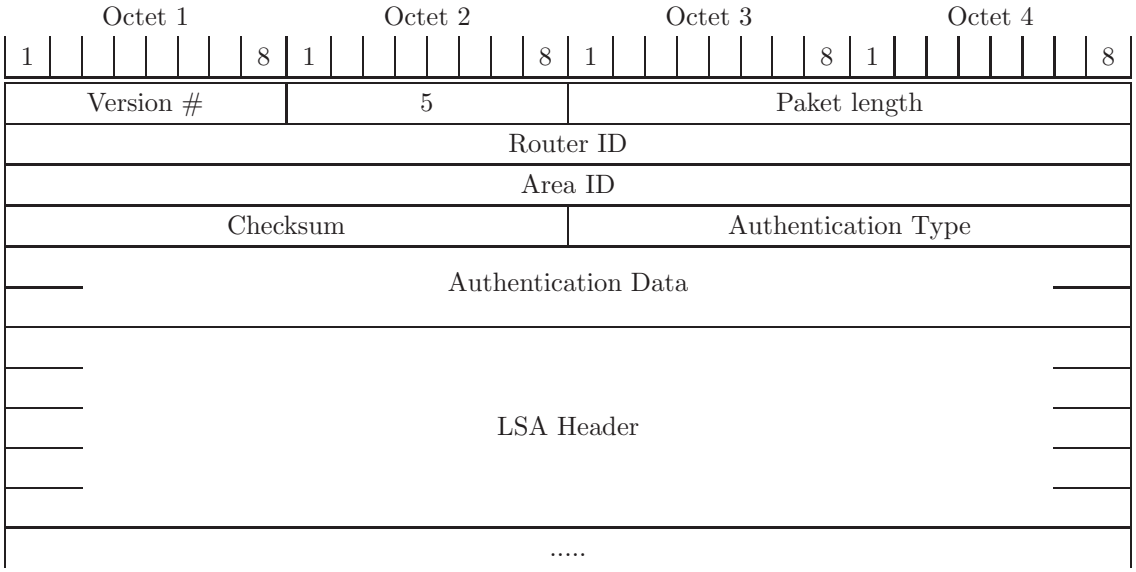


図 8.4: Link State Acknowledgement Packet

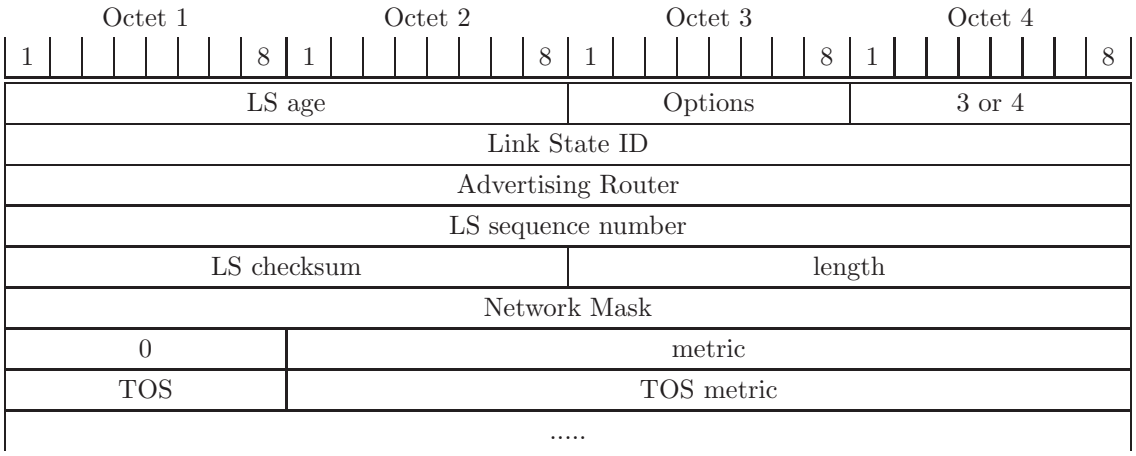


図 8.5: Summary LSA(inter-area) & Summary LSA(AS Border Router)

```
diff -c ../../gated/src/ospf_lsdb.h ./ospf_lsdb.h
*** ../../gated/src/ospf_lsdb.h Sat Feb 20 17:52:19 1999
--- ./ospf_lsdb.h Sat Feb 20 17:46:25 1999
***************
*** 179,184 ****
--- 179,188 ----
  #define DB_ASE_TAG(DB) DB_ASE((DB))->tos0.ExtRtTag
  #define DB_ASE_FORWARD(DB) DB_ASE((DB))->tos0.ForwardAddr

+ #ifdef FANPDIF
+ #include "fanpdif_edgedb.h"
+ #endif
```

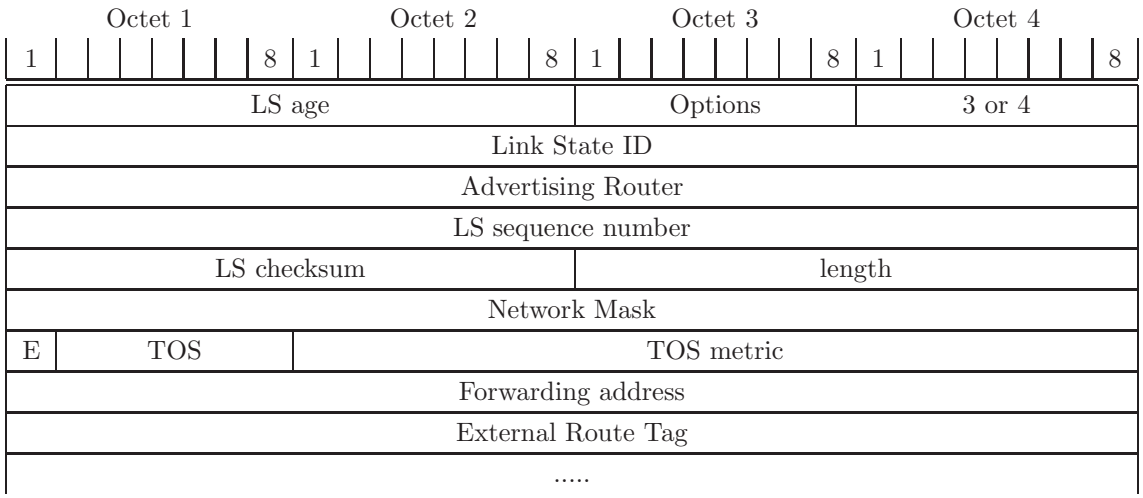


図 8.6: AS External LSA

```
+
  struct LSDB {
      struct LSDB *lsdb_forw; /* for candidate, sum and ase list */
      struct LSDB *lsdb_back;
***************
*** 223,228 ****
--- 227,235 ----
      u_int32 lsdb_dist; /* distance to root */
      time_t lsdb_time; /* for keeping age - stamped when arrived */
      struct NBR_LIST *lsdb_retrans; /* nbrs pointing to this lsdb */
+ #ifdef FANPDIF
+  EHentry *lsdb_edgenode; /* edge node */
+ #endif
  };

  #define LSDB_LIST(list, db) \
diff -c ../../gated/src/ospf_rt.c ./ospf_rt.c
*** ../../gated/src/ospf_rt.c Sat Feb 20 17:52:23 1999
--- ./ospf_rt.c Sat Feb 20 17:46:29 1999
***************
*** 125,130 ****
--- 125,137 ----
  #include "include.h"
  #include "inet.h"
  #include "ospf.h"
+ #ifdef FANPDIF
+ #include "fanpdif.h"
+ #include "fanpdif_msg.h"
```

```
+ #include "fanpdif_edgedb.h"
+ #include "fanpdif_rtdb.h"
+ #include "fanpdif_ospf.h"
+ #endif

  static block_t ospf_rtinfo_block_index;

***************
*** 133,138 ****
--- 140,150 ----
       rt, rt_entry *)
  {
      struct LSDB *db = ORT_V(rt);
+ #ifdef FANPDIF
+  u_long destaddr;
+  u_long netmask;
+  struct LSDB *ptp;
+ #endif

      fprintf(fp,
       "\t\t\tCost: %d\tArea: %A\tType: %s\tAdvRouter: %A\n",
***************
*** 152,158 ****
       rt->rt_metric,
       rt->rt_metric2);
   }
!     }
  }


--- 164,185 ----
       rt->rt_metric,
       rt->rt_metric2);
   }
!  }
! #ifdef FANPDIF
!  if(!db->lsdb_edgenode){
!  fprintf(fp, "\t\t\tno lsdb_edgenode\n");
!  if(!strncmp("ac", rt->rt_ifaps[0]->ifa_link->ifl_name, 2)){
!  if(LS_TYPE(db) != LS_ASE)
!  fprintf(fp,"\t\t\t\t-> inside node in CSR cloud\n");
!  else
!  fprintf(fp,"something wrong\n");
!  }
```

```
!  }
!  else
!  fprintf(fp, "\t\t\t0x%x EdgeRID: %A\n",
!  db->lsdb_edgenode,
!  sockbuild_in(0, db->lsdb_edgenode->ed_edgeid));
! #endif
  }


***************
*** 337,343 ****
       while (i--) {
       routers[i] = sockbuild_in(0, ORT_NH(rt, i));
       }
!      (void) rt_change(rt,
    db->lsdb_dist,
    (metric_t) -1,
    rt->rt_tag,
--- 364,371 ----
       while (i--) {
       routers[i] = sockbuild_in(0, ORT_NH(rt, i));
       }
!      (void)
!  rt_change(rt,
    db->lsdb_dist,
    (metric_t) -1,
    rt->rt_tag,
diff -c ../../gated/src/ospf_rtab.c ./ospf_rtab.c
*** ../../gated/src/ospf_rtab.c Sat Feb 20 17:52:23 1999
--- ./ospf_rtab.c Sat Feb 20 17:46:29 1999
***************
*** 126,131 ****
--- 126,137 ----
  #include "inet.h"
  #include "ospf.h"

+ #ifdef FANPDIF
+ #include "fanpdif.h"
+ #include "fanpdif_edgedb.h"
+ #endif
+
+

  /*
```

```
   * Is this route an active intra or inter area route?
***************
*** 277,282 ****
--- 283,293 ----
  {
      rt_entry *rt;
      struct OSPF_ROUTE *rr = (struct OSPF_ROUTE *) 0;
+ #ifdef FANPDIF
+  u_long advrtr;
+  struct LSDB *adv;
+ #endif
+

      assert(DB_RTR(v));

***************
*** 285,291 ****
       * and is used to check for valid entry
       */

!     switch (LS_TYPE(v)) {
      case LS_STUB:
      case LS_NET:
   /* if attached to this net, interface is the first hop */
--- 296,302 ----
       * and is used to check for valid entry
       */

!    switch (LS_TYPE(v)) {
      case LS_STUB:
      case LS_NET:
   /* if attached to this net, interface is the first hop */
***************
*** 428,433 ****
--- 439,453 ----
   break;

      case LS_ASE:
+ #ifdef FANPDIF
+         advrtr = ADV_RTR(v);
+  adv = FindLSA(v->lsdb_area, advrtr, advrtr, LS_RTR);
+  if(adv)
+  v->lsdb_edgenode = adv->lsdb_edgenode;
+  else
```

```
+  v->lsdb_edgenode = NULL;
+ #endif
+
   if (!v->lsdb_route) {
       /* see if one exists on current routing table */
       findroute(rt, DB_NETNUM(v), DB_MASK(v));
diff -c ../../gated/src/ospf_spf.c ./ospf_spf.c
*** ../../gated/src/ospf_spf.c Sat Feb 20 17:52:26 1999
--- ./ospf_spf.c Sat Feb 20 17:46:32 1999
***************
*** 126,131 ****
--- 126,137 ----
  #include "inet.h"
  #include "ospf.h"

+ #ifdef FANPDIF
+ #include "fanpdif.h"
+ #include "fanpdif_ospf.h"
+ #include "fanpdif_edgedb.h"
+ #endif
+

  /*
   * Add next hop entry to nh_block
***************
*** 526,532 ****
      return FALSE;
  }

-
  /*
   * Add parent to vertex
   * - used by all entries except for ase() who rolls its own
--- 532,537 ----
***************
*** 541,546 ****
--- 546,557 ----
         trans_area, struct AREA *)
  {
      int ret = TRUE;
+ #ifdef FANPDIF
+  u_long ifaddr;
+  EHentry *ehent;
+  u_long advrtr;
```

```
+  struct LSDB *adv;
+ #endif

      if (v == DB_FIRSTQ(a->spf))
   return ret;
***************
*** 641,646 ****
--- 652,730 ----
   }
      }

+ #ifdef FANPDIF
+
+  /*
+  keep parent edge data into this lsdb except ASE.
+  LS_ASE is handled in ospf_rtab.h:addroute()
+  */
+  switch(LS_TYPE(v)){
+  case LS_STUB:
+  case LS_SUM_NET :
+  case LS_SUM_ASB:
+   advrtr = ADV_RTR(v);
+   adv = FindLSA(v->lsdb_area, advrtr, advrtr, LS_RTR);
+   if(adv)
+  v->lsdb_edgenode = adv->lsdb_edgenode;
+   break;
+
+  case LS_NET:
+   v->lsdb_edgenode = parent->lsdb_edgenode;
+   break;
+
+  case LS_RTR:
+   /*  think OSPF all Border Router as Edge Router */
+   if(fc_ospf.fc_flags & FANPDIF_CONF_OSPF_ALLBR){
+  /* AREA Border Router */
+  if(
+   ((ntohs(DB_RTR(v)->E_B) & bit_B) ||
+    (ntohs(DB_RTR(v)->E_B) & bit_E)) &&
+    v->lsdb_area->area_id == FC_OPT_AREAID(fc_ospf)
+  ){
+  ehent = (*fc_ospf.fc_get_edgeent)(&fc_ospf, LS_ID(v));
+
+  /* this entry is still not registered */
+  if(!ehent){
```

```
+  (*fc_ospf.fc_add_edgeent)
+  (fanpd_tp, &fc_ospf, LS_ID(v),
+  FC_OPT_IFNAME(fc_ospf));
+  ehent = (*fc_ospf.fc_get_edgeent)(&fc_ospf, LS_ID(v));
+  if((ntohs(DB_RTR(v)->E_B) & bit_B))
+  ehent->ed_option = EDGE_OPT_OSPF_ABR;
+  else if((ntohs(DB_RTR(v)->E_B) & bit_E))
+  ehent->ed_option = EDGE_OPT_OSPF_ASBR;
+  }
+  v->lsdb_edgenode = ehent;
+  ifaddr = fanpdif_ospf_get_ifaddr(v, parent);
+  if(ifaddr){
+  ehent->ed_ifaddr = ifaddr;
+  ehent->ed_state = EDGE_OSPFRT_STATE_FOUNDIF;
+  }
+  ehent->ed_netaddr = 0;
+  ehent->ed_netmask = 0;
+  }
+  break;
+   }
+
+   /* FANPDIF Manual Configuration */
+   else if(!(ehent = (*fc_ospf.fc_get_edgeent)(&fc_ospf, LS_ID(v)))){
+  v->lsdb_edgenode = parent->lsdb_edgenode;
+  break;
+   }
+   else{
+  /* this LS_ID is edge node */
+  v->lsdb_edgenode = ehent;
+  ifaddr = fanpdif_ospf_get_ifaddr(v, parent);
+  if(ifaddr){
+  ehent->ed_ifaddr = ifaddr;
+  ehent->ed_state = EDGE_OSPFRT_STATE_FOUNDIF;
+  }
+  ehent->ed_netaddr = 0;
+  ehent->ed_netmask = 0;
+  break;
+   }
+  }
+ #endif
      return ret;
  }

***************
```

```
*** 959,964 ****
--- 1043,1052 ----
      v = FindLSA(a, MY_ID, MY_ID, LS_RTR);
      assert(v);

+ #ifdef FANPDIF
+  v->lsdb_edgenode = NULL;
+ #endif
+
      v->lsdb_dist = 0;
      DB_WHERE(v) = ON_RTAB;
      DB_ADDQUE(a->spf, v);
***************
*** 1334,1340 ****
  spf __PF1(area, struct AREA *)
  {
      struct AREA *sum_area = (IAmBorderRtr && ospf.backbone.ifUcnt) ? &ospf.backbone : area;
-
      area->spfcnt++;
      RTAB_REV++;

--- 1422,1427 ----
***************
*** 1347,1352 ****
--- 1434,1440 ----

      if (intra(area))
   return; /* Out of memory */
+
      /*
       * update asbr and abr routing tables for this area
       */
diff -c ../../gated/src/task.c ./task.c
*** ../../gated/src/task.c Sat Feb 20 17:53:00 1999
--- ./task.c Sat Feb 20 17:47:06 1999
***************
*** 124,131 ****
--- 124,135 ----
  #ifdef PROTO_RDISC
  #include "rdisc.h"
  #endif /* PROTO_RDISC */
+ #ifdef  FANPDIF
+ #include "fanpdif.h"
+ #endif /* FANPDIF */
```

```
  #include "parse.h"

+
  /* For systems that do not have FD_SET macros */

  #ifndef FD_SET
***************
*** 2709,2714 ****
--- 2713,2721 ----
  #ifdef PROTO_RDISC
      { rdisc_init,rdisc_var_init,"RDISC" },
  #endif /* PROTO_RDISC */
+ #ifdef  FANPDIF
+  { fanpdif_init,0,"FANPDIF"},
+ #endif /* FANPDIF */
          {0}
      };
```

```
 1 /*
 2 fanpdif.h:  hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6
 7 #ifndef  _FANPD_IF_H
 8 #define _FANPD_IF_H
 9
10 #define  FANPD_IF_SOCKET "/var/tmp/.gated"
11 /* #define FANPDIF_DEBUG */
12
13 /*
14 fanpd task is client for fanpd.
15 open socket and connect to fanpd.
16 */
17
18
19 /* PROTOTYPEs */
20 PROTOTYPE(fanpdif_init,
21           extern void,
22 (void));
23
24 PROTOTYPE(fanpdif_cleanup,
25           extern void,
26 (task *));
```

```
27
28 PROTOTYPE(fanpdif_connect,
29   extern void,
30   (task_timer *, time_t));
31
32 PROTOTYPE(fanpdif_recv,
33   extern void,
34   (task *));
35
36 PROTOTYPE(fanpdif_write,
37   extern void,
38   (task *));
39
40 PROTOTYPE(fanpdif_flash,
41   extern void,
42   (task *, rt_list *));
43 PROTOTYPE(fanpdif_terminate,
44   extern void,
45   (task *));
46
47 PROTOTYPE(fanpdif_dump,
48   extern void,
49   (task *, FILE *));
50
51 extern void_t
52 fanpdif_malloc(task *, size_t);
53
54 extern void
55 fanpdif_free(task *, void_t, size_t);
56
57 /* fanpdif_state return status of fanpdif module */
58 extern int
59 fanpdif_state(void);
60
61 #define FANPDIF_STAT_INIT 0
62 #define FANPDIF_STAT_WAITCONNECT 1
63 #define FANPDIF_STAT_CONNECT 2
64
65 void fanpdif_wait_connection(void);
66 extern task *fanpd_tp;
67
68 #endif /* _FANPD_IF_H */
```

```
 1 /*
 2 fanpdif.c: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #include "include.h"
 7 #include "krt.h"
 8 #include "ospf.h"
 9 #include <arpa/inet.h>
10
11 #include "fanpdif.h"
12 #include "fanpdif_msg.h"
13 #include "fanpdif_edgedb.h"
14 #include "fanpdif_rt.h"
15 #include "fanpdif_job.h"
16
17 /* task for communication to fanpd */
18 task *fanpd_tp = 0;     /* task for fanpd interface */
19
20 /* trace *trace; */
21 int  fanpd_soc = -1; /* socket for communicate with fanpd */
22 extern int errno;
23
24 static int recvcount = 0;
25 static int connectcount = 0;
26 static int connectfailcount = 0;
27 static int flashcount = 0;
28
29 static int allocated_memsize = 0;
30
31 static int state = FANPDIF_STAT_INIT;
32
33 int
34 fanpdif_state(void)
35 {
36 return state;
37 }
38
39 void
40 fanpdif_wait_connection(void)
41 {
42 if(fanpd_soc >= 0){
43 close(fanpd_soc);
44 fanpd_tp->task_socket = fanpd_soc = -1;
```

```
45 }
46
47 state = FANPDIF_STAT_WAITCONNECT;
48
49 /* request socket */
50 fanpd_soc =  task_get_socket(fanpd_tp, AF_UNIX, SOCK_STREAM, 0);
51 assert(fanpd_soc >= 0);
52
53 task_set_socket(fanpd_tp, fanpd_soc);
54 fanpdif_connectjob_start(fanpd_tp);
55 }
56
57 void
58 fanpdif_init(void){
59 state = FANPDIF_STAT_INIT;
60
61 if(!fanpd_tp){
62 fanpd_tp = task_alloc("FANPD_IF",
63    TASKPRI_EXTPROTO,
64    0);
65 fanpd_tp->task_flags = 0;
66 task_set_terminate(fanpd_tp, fanpdif_terminate);
67 task_set_dump(fanpd_tp, fanpdif_dump);
68
69 if(!task_create(fanpd_tp)){
70 task_quit(EINVAL);
71 }
72
73 task_set_flash(fanpd_tp, fanpdif_flash);
74 task_set_newpolicy(fanpd_tp, fanpdif_flash);
75 task_set_recv(fanpd_tp, fanpdif_recv);
76 task_set_cleanup(fanpd_tp, fanpdif_cleanup);
77
78 }
79
80 /* ignore return value from fanpdif_edgedb_init */
81 fanpdif_edgedb_init(fanpd_tp);
82 /*
83 if(!fanpdif_edgedb_init(fanpd_tp)){
84 fanpdif_terminate(fanpd_tp);
85 return;
86 }
87 */
88 /* these routine must be called at "gcc reconfig" */
```

```
 89 fanpdif_ospf_init(fanpd_tp);
 90 fanpdif_wait_connection();
 91
 92 flashcount = 0;
 93 return;
 94 }
 95
 96 void
 97 fanpdif_connect(task_timer *tip, time_t intval)
 98 {
 99 int err;
100 struct sockaddr_un sun;
101 int len;
102
103 bzero(&sun, sizeof(sun));
104 sun.sun_family = AF_LOCAL;
105 strcpy(sun.sun_path, FANPD_IF_SOCKET);
106 len = SUN_LEN(&sun);
107
108 err = connect(fanpd_soc, (struct sockaddr *)&sun, len);
109 if(err < 0){
110 connectfailcount++;
111 return;
112 }
113 connectcount++;
114 task_set_socket(fanpd_tp, fanpd_soc);
115 fanpdif_connectjob_term(fanpd_tp);
116 fanpdif_msg_init();
117
118 state = FANPDIF_STAT_CONNECT;
119 return;
120 }
121
122 void
123 fanpdif_recv(task *tsp)
124 {
125 int n;
126 fanpdif_msg fmsg;
127
128 if((n = fanpdif_get_msg(tsp->task_socket, &fmsg)) == 0){
129 fanpdif_wait_connection();
130 return;
131 }
132 else if(n < 0){
```

```
133 assert(n > 0);
134 }
135
136 recvcount++;
137 switch(fmsg.fm_opcode){
138 case GRT_REQUEST:
139 if(!fanpdif_ospf_putall(tsp)){
140 /* peer server down or socket is broken. */
141 fanpdif_wait_connection();
142 return;
143 }
144
145 break;
146 default:
147 /*
148 ignore other opcode and
149 unknown operation message.
150 */
151 return;
152 }
153 }
154
155 void
156 fanpdif_flash(task *tp, rt_list *rlt)
157 {
158 flashcount++;
159
160 rt_open(tp);
161 fanpdif_ospf_flash(tp, rlt);
162 rt_close(tp, (gw_entry *)0, 0, NULL);
163 }
164
165 void
166 fanpdif_dump(task *tsp, FILE *fp)
167 {
168 fprintf(fp, "\tsocket %d \n", fanpd_soc);
169 fprintf(fp, "\tallocated memory size %d bytes\n", allocated_memsize);
170
171 fprintf(fp, "\tconnect %d times \n", connectcount);
172 fprintf(fp, "\tconnect fail %d times \n", connectfailcount);
173 fprintf(fp, "\trecv %d times \n", recvcount);
174 fprintf(fp, "\tflash %d times \n", flashcount);
175 fanpdif_edgedb_dump(fanpd_tp, fp);
176 fanpdif_msg_dump(fp);
```

```
177 fanpdif_ospf_dump(fp);
178 return;
179 }
180
181 /*
182 this routine will be called
183 at "gdc reconfig" before fanpdif_init.
184 */
185 void
186 fanpdif_cleanup(task *tsp)
187 {
188 if(fanpd_soc >= 0)
189 fanpdif_msg_delall(fanpd_soc, AF_INET, FANPDIF_MSGPROTO_OSPF);
190
191 fanpdif_ospf_terminate(fanpd_tp);
192 fanpdif_edgedb_term(fanpd_tp);
193
194 return;
195 }
196
197 void
198 fanpdif_terminate(task *tsp)
199 {
200 if(fanpd_soc >= 0)
201 fanpdif_msg_delall(fanpd_soc, AF_INET, FANPDIF_MSGPROTO_OSPF);
202 /*
203 close(fanpd_soc);
204 fanpd_tp->task_socket = fanpd_soc = -1;
205 */
206
207 fanpdif_connectjob_term(tsp);
208
209 fanpdif_ospf_terminate(tsp);
210 fanpdif_edgedb_term(tsp);
211
212 task_set_terminate(tsp, 0);
213 task_set_dump(tsp, 0);
214 task_set_recv(tsp, 0);
215 task_set_flash(tsp, 0);
216 task_set_cleanup(tsp, 0);
217
218 task_delete(tsp);
219 }
220
```

```
221 void_t
222 fanpdif_malloc(task *tp, size_t size)
223 {
224 allocated_memsize += size;
225
226 return task_mem_malloc(tp, size);
227 }
228
229 void
230 fanpdif_free(task *tp, void_t p, size_t size)
231 {
232 allocated_memsize -= size;
233
234 task_mem_free(tp, p);
235 }
```

```
 1 /*
 2 fanpdif_msg.h: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #ifndef _FANPD_IF_MSG_H
 7 #define _FANPD_IF_MSG_H
 8
 9 /* interface between FANPD & GATED,
10    version 1 */
11
12 typedef struct _fanpdif_message {
13 u_char fm_version; /* version 1 */
14 u_char fm_opcode; /* see below */
15 u_short fm_len; /* length of this message */
16 u_char fm_family; /* support only AF_INET */
17 u_char fm_rtproto; /* OSPF, BGP  */
18 u_short fm_prelen; /* VLSM length */
19 union {
20 /* for AF_INET */
21 struct {
22 u_long _dest; /* destination address */
23 u_long _egress; /* OSPF edge node or BGP nexthop */
24 u_short _ifindex; /* next hop ifindex */
25 u_short _egrsmask;  /* egress network mask */
26 u_long _next; /* next hop address */
27 u_long _egnet; /* egress network */
```

```
28 } _addr;
29 /* for AF_INET6 */
30 struct {
31 char  _dest6[128];
32 char  _egress6[128]; /* OSPF edge node or BGP nexthop */
33 u_short _ifindex; /* next hop ifindex */
34 u_short _egrsmask;  /* egress network mask */
35 char  _next6[128];
36 char  _egnet6[128]; /* OSPF edge node or BGP nexthop */
37 } _addr6;
38 } _addresses;
39 } fanpdif_msg;
40
41 #define  fm_dstaddr _addresses._addr._dest
42 #define  fm_egrsaddr _addresses._addr._egress
43 #define  fm_ifindex  _addresses._addr._ifindex
44 #define  fm_egrsmask _addresses._addr._egrsmask
45 #define  fm_nexthop _addresses._addr._next
46 #define  fm_egrsnet _addresses._addr._egnet
47 #define  fm_dstaddr6 _addresses._addr6._dest6
48 #define  fm_egrsaddr6 _addresses._addr6._egress6
49 #define  fm_nexthop6 _addresses._addr6._next6
50 #define  fm_egrsnet6 _addresses._addr6._egnet6
51 #define  fm_egrsmask6 _addresses._addr6._egrsmask
52
53 /* version number */
54 #define FANPDIF_VERSION1  1
55
56 /* operation codes */
57 #define GRT_ADD 0x1 /* gated -> fanpd */
58 #define GRT_DEL 0x2 /* gated -> fanpd */
59 #define GRT_CHANGE 0x3 /* gated -> fanpd */
60 #define GRT_REQUEST 0x4 /* fanpd -> gated */
61 #define GRT_DELALL 0x5 /* gated -> fanpd */
62
63 /*  routing PROTO field */
64 #define FANPDIF_MSGPROTO_ALL 0
65 #define FANPDIF_MSGPROTO_OSPF 1
66 #define FANPDIF_MSGPROTO_BGP 2
67
68 extern const char *fanpdif_opnames[];
69
70 /*  three length patern */
71 #define FANPDIF_AC_LEN 28 /* use ADD, CHANGE */
```

```
 72 #define FANPDIF_AC_LEN6 (128*4+12) /* use ADD, CHANGE */
 73
 74 #define FANPDIF_D_LEN 12  /* use DEL */
 75 #define FANPDIF_D_LEN6 (128+8) /* use DEL */
 76
 77 #define FANPDIF_RDA_LEN  6  /* use REQUEST, DELALL */
 78 #define FANPDIF_RDA_LEN6  6  /* use REQUEST, DELALL */
 79
 80 /* functions  */
 81 /* only v4 address */
 82 /*
 83 RETURN VALUE
 84 same value returned by read/write.
 85 */
 86 int fanpdif_msg_add(int s, u_char af, u_char rtproto, u_long dst, u_short mask,
 87 u_long egress, u_long next, u_short ifidx,  u_long egrnet, u_short egrmask);
 88 int fanpdif_msg_del(int s, u_char af, u_char rtproto, u_long dst, u_short mask);
 89 int fanpdif_msg_chg(int s, u_char af, u_char rtproto, u_long dst, u_short mask,
 90 u_long egress, u_long next, u_short ifidx,  u_long egrnet, u_short egrmask);
 91 int fanpdif_msg_req(int s, u_char af, u_char rtproto);
 92 int fanpdif_msg_delall(int s, u_char af, u_char rtproto);
 93
 94 int fanpdif_get_msg(int, fanpdif_msg *);
 95
 96 /* function for gated */
 97 void fanpdif_msg_init(void);
 98 void fanpdif_msg_dump(FILE *);
 99
100 #endif /*_FANPD_IF_MSG_H */
```

```
 1 /*
 2 fanpdif_msg.c: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #include <sys/types.h>
 7 #include <fcntl.h>
 8 #include <sys/socket.h>
 9 #include <sys/errno.h>
10 #include "include.h"
11 #include "fanpdif.h"
12 #include "fanpdif_msg.h"
13
```

```
14 #define GRT_UNKNOWN 0
15 static u_long put_count[GRT_DELALL+1];
16 static u_long get_count[GRT_DELALL+1];
17
18 const char *fanpdif_opnames[] = {
19 "unknown",
20 "GRT_ADD",
21 "GRT_DEL",
22 "GRT_CHANGE",
23 "GRT_REQUEST",
24 "GRT_DELALL",
25 0
26 };
27
28 static int
29 putmsg(int s, char *data, int len)
30 {
31 int m, n;
32 fanpdif_msg *fmsg = (fanpdif_msg *)data;
33
34     m = 0;
35
36 retry1:
37 while((n = write(s, (char *)&data[m], len - m)) > 0){
38 if(len - m != n){
39 m += n;
40 continue;
41 }
42 else{
43 put_count[fmsg->fm_opcode]++;
44 return fmsg->fm_len;
45 }
46 }
47 if(n <= 0 && errno == EINTR)
48 goto retry1;
49
50 return n;
51 }
52
53 static int
54 fanpdif_put_msg(int s, fanpdif_msg *fmsg)
55 {
56 int m, n;
57 char *data;
```

```
 58
 59 fmsg->fm_version = FANPDIF_VERSION1;
 60
 61 switch(fmsg->fm_opcode){
 62 case GRT_ADD:
 63 case GRT_CHANGE:
 64 fmsg->fm_len = htons(FANPDIF_AC_LEN);
 65 break;
 66 case GRT_DEL:
 67 fmsg->fm_len = htons(FANPDIF_D_LEN);
 68 break;
 69 case GRT_REQUEST:
 70 case GRT_DELALL:
 71 fmsg->fm_len = htons(FANPDIF_RDA_LEN);
 72 break;
 73 defalut:
 74 abort();
 75 }
 76
 77 return putmsg(s, (char *)fmsg, ntohs(fmsg->fm_len));
 78 }
 79
 80 static int
 81 getmsg(int s, char *data, int len)
 82 {
 83 int m,n;
 84
 85     m = 0;
 86
 87 retry2:
 88 while((n = read(s, (char *)&data[m], len - m)) > 0){
 89 if(len - m != n){
 90 m += n;
 91 continue;
 92 }
 93 else
 94 return len;
 95 }
 96 if(n <= 0 && errno == EINTR)
 97 goto retry2;
 98 return n;
 99 }
100
101 int
```

```
102 fanpdif_get_msg(int s, fanpdif_msg *fmsg)
103 {
104 int n, len;
105 if((n = getmsg(s, (char *)fmsg, FANPDIF_RDA_LEN))
106 != FANPDIF_RDA_LEN)
107 return n;
108
109 if(fmsg->fm_opcode > 0 && fmsg->fm_opcode < GRT_DELALL + 1)
110 get_count[fmsg->fm_opcode]++;
111 else {
112 get_count[GRT_UNKNOWN]++;
113 return n;
114 }
115
116 switch(fmsg->fm_opcode){
117 case GRT_ADD:
118 case GRT_CHANGE:
119 len = FANPDIF_AC_LEN - FANPDIF_RDA_LEN;
120 break;
121 case GRT_DEL:
122 len = FANPDIF_D_LEN - FANPDIF_RDA_LEN;
123 break;
124 case GRT_REQUEST:
125 case GRT_DELALL:
126 return n;
127 }
128
129 n = getmsg(s, (char *)&fmsg->fm_prelen, len);
130 if(n <= 0)
131 return n;
132 else
133 return (n + FANPDIF_RDA_LEN);
134 }
135
136
137 int
138 fanpdif_msg_add(int s, u_char af, u_char rtproto, u_long dst, u_short mask,
139 u_long egress, u_long next, u_short ifidx, u_long egrnet, u_short egrmask)
140 {
141 fanpdif_msg fmsg;
142
143 fmsg.fm_version = FANPDIF_VERSION1;
144 fmsg.fm_family = af;
145 fmsg.fm_rtproto = rtproto;
```

```
146 fmsg.fm_opcode = GRT_ADD;
147 fmsg.fm_dstaddr = dst;
148 fmsg.fm_prelen = htons(mask);
149 fmsg.fm_egrsaddr = egress;
150 fmsg.fm_nexthop = next;
151 fmsg.fm_egrsnet = egrnet;
152 fmsg.fm_egrsmask = htons(egrmask);
153 fmsg.fm_ifindex = htons(ifidx);
154
155 return fanpdif_put_msg(s, &fmsg);
156 }
157
158 int
159 fanpdif_msg_del(int s, u_char af, u_char rtproto, u_long dst, u_short mask)
160 {
161 fanpdif_msg fmsg;
162
163 fmsg.fm_version = FANPDIF_VERSION1;
164 fmsg.fm_family = AF_INET;
165 fmsg.fm_family = af;
166 fmsg.fm_rtproto = rtproto;
167 fmsg.fm_opcode = GRT_DEL;
168 fmsg.fm_dstaddr = dst;
169 fmsg.fm_prelen = htons(mask);
170
171 return fanpdif_put_msg(s, &fmsg);
172 }
173
174
175 int
176 fanpdif_msg_chg(int s, u_char af, u_char rtproto, u_long dst, u_short mask,
177 u_long egress, u_long next, u_short ifidx, u_long egrnet, u_short egrmask)
178 {
179 fanpdif_msg fmsg;
180
181 fmsg.fm_version = FANPDIF_VERSION1;
182 fmsg.fm_family = af;
183 fmsg.fm_rtproto = rtproto;
184 fmsg.fm_opcode = GRT_CHANGE;
185 fmsg.fm_dstaddr = dst;
186 fmsg.fm_prelen = htons(mask);
187 fmsg.fm_egrsaddr = egress;
188 fmsg.fm_nexthop = next;
189 fmsg.fm_egrsnet = egrnet;
```

```
190 fmsg.fm_egrsmask = htons(egrmask);
191 fmsg.fm_ifindex = htons(ifidx);
192
193 return fanpdif_put_msg(s, &fmsg);
194 }
195
196 int
197 fanpdif_msg_req(int s, u_char af, u_char rtproto)
198 {
199 fanpdif_msg fmsg;
200
201 fmsg.fm_version = FANPDIF_VERSION1;
202 fmsg.fm_family = af;
203 fmsg.fm_rtproto = rtproto;
204 fmsg.fm_opcode = GRT_REQUEST;
205
206 return fanpdif_put_msg(s, &fmsg);
207 }
208
209 int
210 fanpdif_msg_delall(int s, u_char af, u_char rtproto)
211 {
212 fanpdif_msg fmsg;
213
214 fmsg.fm_version = FANPDIF_VERSION1;
215 fmsg.fm_family = af;
216 fmsg.fm_rtproto = rtproto;
217 fmsg.fm_opcode = GRT_DELALL;
218
219 return fanpdif_put_msg(s, &fmsg);
220 }
221
222 void
223 fanpdif_msg_dump(FILE *fp)
224 {
225 int i;
226 fprintf(fp, "fanpdif_msg_dump:\n");
227
228 for(i = GRT_UNKNOWN; i < GRT_DELALL + 1; i++)
229 fprintf(fp, "\t%s: put %d, get %d\n",
230 fanpdif_opnames[i], put_count[i], get_count[i]);
231
232 }
233
```

```
234 void
235 fanpdif_msg_init(void)
236 {
237 bzero(put_count, sizeof(put_count));
238 bzero(get_count, sizeof(get_count));
239 }
```

```
 1 /*
 2 fanpdif_job.h: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #ifndef _FANPDIF_JOB_H_
 7 #define _FANPDIF_JOB_H_
 8
 9 /*
10 this file defines a job, which works
11 every FANPDIF_JOB_T_UPDATE.
12 */
13
14 #define  FANPDIF_JOB_T_UPDATE 15
15 #define  FANPDIF_JOB_T_OFFSET 15
16
17 extern void fanpdif_connectjob_start(task *);
18 extern void fanpdif_connectjob_term(task *);
19
20 #endif /* _FANPDIF_JOB_H_ */
```

```
 1 /*
 2 fanpdif_job.c: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #include "include.h"
 7 #include "inet.h"
 8 #include "krt.h"
 9 #include "fanpdif.h"
10 #include "fanpdif_edgedb.h"
11 #include "fanpdif_msg.h"
12 #include "fanpdif_job.h"
13 #include "fanpdif_ospf.h"
14
```

```
15 static task_timer *job_timer = 0;
16
17 /* timer job for connect to fanpd */
18 static task_timer *connect_timer = 0;
19
20 void
21 fanpdif_connectjob_start(task *tp)
22 {
23         if(connect_timer)
24 return;
25 connect_timer = task_timer_create(tp, "CONNECT", (flag_t)0, (time_t)5,
26 (time_t)5, fanpdif_connect, (void_t)0);
27 }
28
29 void
30 fanpdif_connectjob_term(task *tp)
31 {
32      if(connect_timer){
33  task_timer_delete(connect_timer);
34  connect_timer = 0;
35  }
36 }
37
38
```

```
 1 /*
 2 fanpdif_rt.h: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #ifndef _FANPDIF_RT_H_
 7 #define _FANPDIF_RT_H_
 8
 9 rt_entry *fanpdif_activert_lookup(task *, u_long, u_long, u_long);
10
11 /*
12 255.255.255.0 -> return 24
13 255.255.255.224 -> return 27
14 */
15 extern u_short masktomasklen(u_long);
16 #endif /* _FANPDIF_RT_H_ */
```

```
 1 /*
 2 fanpdif_rt.c: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #include "include.h"
 7 #include "inet.h"
 8 #include "krt.h"
 9 #include "fanpdif_rt.h"
10 #include "fanpdif_ospf.h"
11
12 u_short
13 masktomasklen(u_long mask)
14 {
15 register u_short i;
16 register u_long x;
17 i = 32;
18 x = 0x1;
19
20 mask = ntohl(mask);
21
22 while(1){
23 if((x & mask) == x)
24 break;
25 else{
26 x <<= 1;
27 i--;
28 }
29 }
30
31 return i;
32 }
33
34 #define  FANPDIF_SINADR(x)     ((struct sockaddr_in *)(x))->sin_addr.s_addr
35
36 rt_entry *
37 fanpdif_activert_lookup(task *tp, u_long dst, u_long dstmask, u_long proto)
38 {
39 register int i;
40 register rt_head *rth;
41 register rt_list *rtl = rthlist_match(sockbuild_in(0, dst));
42 register rt_entry *rt = (rt_entry *)NULL;
43 int found;
44
```

```
45 if(!rtl)
46 return 0;
47
48 found = 0;
49     RT_LIST(rth, rtl, rt_head) {
50 rt = rth->rth_active;
51 if(!rt)
52 continue;
53 if(rt->rt_gwp->gw_proto != proto)
54 continue;
55 if(FANPDIF_SINADR(rt->rt_dest) == dst &&
56    FANPDIF_SINADR(rt->rt_dest_mask) == dstmask){
57 found = 1;
58 break;
59 }
60 }RT_LIST_END(rth, rtl, rt_head);
61
62 RTLIST_RESET(rtl);
63 if(found)
64 return rt;
65 return NULL;
66 }
```

```
 1 /*
 2 fanpdif_edgedb.h: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #ifndef  _FANPDIF_EDGEDB_H
 7 #define  _FANPDIF_EDGEDB_H
 8
 9 #define  _EDGEROUTER_FILE  "/etc/csr/egressnodes"
10 #define  DEFAULT_EDGE_IFNAME "ac"
11
12 /* hash table for edge router */
13 #define  EDGE_HTSIZE  7
14 #define  EHASH(x) ((x) % EDGE_HTSIZE)
15
16 typedef struct _edgedb_hentry {
17 u_long ed_edgeid; /* edge OSPF router id or BGP peer addr*/
18 char   ed_ifname[8]; /* interface name for this edge router */
19 u_long ed_ifindex; /* Interface Index */
20 u_long ed_ref; /* reference count */
```

```
21 u_long ed_ifaddr;   /* interface address in this network*/
22 u_long ed_netaddr; /* network address on routing table */
23 u_short ed_netmask; /* network mask on routing table */
24 u_char ed_option; /* optional data; ospf abr/asbr */
25 u_char ed_state; /* status */
26 struct _edgedb_hentry *ed_next;
27 } EHentry;
28
29 /* for OSPF route state */
30 #define EDGE_OSPFRT_STATE_INIT 0   /* init */
31 #define EDGE_OSPFRT_STATE_FOUNDIF 1   /* interface address found */
32 #define EDGE_OSPFRT_STATE_FOUNDRT 2   /* found rtinfomation for the if */
33
34 #define EDGE_OPT_OSPF_ABR 1
35 #define EDGE_OPT_OSPF_ASBR 2
36
37 /*  fanpdif configuration & Edge CSR DataBase */
38 typedef struct _fanpdif_config_t{
39 u_short fc_flags; /* see blow */
40 u_short fc_nedge_entry; /* number of edge entry */
41 char   *fc_optdata; /* optional data for each protocol use,
42    I don't free this entry */
43 EHentry *fc_edgedb[EDGE_HTSIZE];
44 int  (*fc_add_edgeent)
45 (task *, struct _fanpdif_config_t *, u_long, char *);
46 int  (*fc_del_edgeent)
47 (task *, struct _fanpdif_config_t *, u_long);
48 EHentry *(*fc_get_edgeent)
49 (struct _fanpdif_config_t *, u_long);
50 void (*fc_varinit_edgedb)
51 (struct _fanpdif_config_t *);
52 void (*fc_free_edgedb)
53 (task *, struct _fanpdif_config_t *);
54 void  (*fc_dump_edgedb)
55 (task *, FILE *, struct _fanpdif_config_t *);
56 /* EHentry *(*fc_get_edgeent_bynetaddr)
57 (struct _fanpdif_config_t *, u_long, u_long); */
58 } fanpdif_config_t;
59
60 /* for member fc_flags o fanpdif_config_t  */
61 #define FANPDIF_CONF_OSPF 0x1
62 #define FANPDIF_CONF_BGP 0x2
63
64 #define FANPDIF_CONF_OSPF_ALLBR 0x10
```

```
65
66 /* using for fc_ospf.fc_optdata */
67 typedef struct {
68         u_int areaid;
69         char  ifname[8];
70 } fc_optdata_ospf_t;
71 #define FC_OPT_AREAID(x)  (((fc_optdata_ospf_t *)((x).fc_optdata))->areaid)
72 #define FC_OPT_IFNAME(x)  (((fc_optdata_ospf_t *)((x).fc_optdata))->ifname)
73
74 extern fanpdif_config_t fc_ospf;
75 /*extern fanpdif_config_t fc_bgp;*/
76
77 /* external functions */
78 /* if fail, return 0, if success return 1 */
79 extern int fanpdif_edgedb_init(task *);
80 extern void fanpdif_edgedb_dump(task *, FILE *);
81 extern int fanpdif_edgedb_term(task *);
82
83 /* fanpdif command list in config file */
84 #define FC_OSPF_EDGENODE "OSPF_EDGENODE"
85 #define FC_OSPF_ALLBR "OSPF_ALLBR"
86
87 #endif /* _FANPDIF_EDGEDB_H */
```

```
 1 /*
 2 fanpdif_edgedb.c: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #include <stdio.h>
 7 #include <sys/types.h>
 8 #include <sys/socket.h>
 9 #include <netinet/in.h>
10 #include <arpa/inet.h>
11 #include <sys/errno.h>
12 #include "include.h"
13 #include "fanpdif_edgedb.h"
14
15 #define FAIL   0
16 #define SUCCESS 1
17
18 /* configuration for FANPDIF_OSPF */
19 fanpdif_config_t fc_ospf;
```

```
20 static fc_optdata_ospf_t  fc_optdata_ospf;
21 /*fanpdif_config_t fc_bgp; */
22
23 static void    fc_init(task *, fanpdif_config_t *);
24 static int     add_edgeent
25                         (task *, fanpdif_config_t *, u_long, char *);
26 static int     del_edgeent
27                         (task *, fanpdif_config_t *, u_long);
28 static EHentry *get_edgeent
29                         (fanpdif_config_t *, u_long);
30 static void    varinit_edgedb
31                         (fanpdif_config_t *);
32 static void    free_edgedb
33                         (task *, fanpdif_config_t *);
34 static void    dump_edgedb
35                         (task *, FILE *, fanpdif_config_t *);
36
37 static int     fc_parse_and_doit(task *, char *);
38
39 static int
40 add_entry(task *tp, EHentry *ep, EHentry **egdb)
41 {
42 int hashkey;
43 EHentry *hashent;
44
45 if(!tp || !ep || !egdb)
46 return FAIL;
47
48 hashkey = EHASH(ep->ed_edgeid);
49 hashent = egdb[hashkey];
50
51 if(!hashent)
52 egdb[hashkey] = ep;
53 else {
54 for(;; hashent = hashent->ed_next){
55 if(hashent->ed_edgeid == ep->ed_edgeid)
56 return FAIL;
57 if(!hashent->ed_next){
58 hashent->ed_next = ep;
59 break;
60 }
61 }
62 }
63 return SUCCESS;
```

```
 64 }
 65
 66 static int
 67 del_entry(task *tp, u_long edgeid, EHentry **egdb)
 68 {
 69 int hashkey;
 70 EHentry *hashent;
 71 EHentry *eent;
 72
 73 if(!tp || !egdb)
 74 return FAIL;
 75 hashkey = EHASH(edgeid);
 76 hashent = egdb[hashkey];
 77
 78 for(; hashent; hashent = hashent->ed_next)
 79 if(hashent->ed_next &&
 80    hashent->ed_next->ed_edgeid == edgeid){
 81 eent = hashent->ed_next;
 82 hashent->ed_next = eent->ed_next;
 83 fanpdif_free(tp, eent, sizeof(EHentry));
 84 bzero(eent, sizeof(EHentry));
 85 /* I think I should do something for RTDBentrys
 86    which this entry referenced here.  */
 87 return SUCCESS;
 88 }
 89 return FAIL;
 90 }
 91
 92 static int
 93 del_listed_entrys(task *tp, EHentry *hp)
 94 {
 95 if(hp->ed_next)
 96 del_listed_entrys(tp, hp->ed_next);
 97 fanpdif_free(tp, hp, sizeof(EHentry));
 98 bzero(hp, sizeof(EHentry));
 99 return SUCCESS;
100 }
101
102 static void
103 fc_init(task *tp, fanpdif_config_t *fc)
104 {
105 if(!fc)
106 return;
107
```

```
108 bzero((char *)fc, sizeof(fc));
109 fc->fc_add_edgeent = add_edgeent;
110 fc->fc_del_edgeent = del_edgeent;
111 fc->fc_get_edgeent = get_edgeent;
112 fc->fc_varinit_edgedb = varinit_edgedb;
113 fc->fc_free_edgedb = free_edgedb;
114 fc->fc_dump_edgedb = dump_edgedb;
115 }
116
117
118 static int
119 add_edgeent(task *tp, fanpdif_config_t *fc, u_long edgeid, char *ifname)
120 {
121 EHentry *adent;
122 int  n, hashkey;
123
124 if(!tp || !fc)
125 return FAIL;
126
127 adent =(EHentry *)fanpdif_malloc(tp, sizeof(EHentry));
128 if(!adent){
129 fprintf(stderr, "add_edgeent: malloc failed\n");
130 fflush(stderr);
131 task_quit(ENOMEM);
132 }
133
134 adent->ed_edgeid = edgeid;
135 if(ifname && *ifname != '\0')
136 strcpy(adent->ed_ifname, ifname);
137 else
138 strcpy(adent->ed_ifname, DEFAULT_EDGE_IFNAME);
139
140 adent->ed_ref = 0;
141 adent->ed_netaddr = 0;
142 adent->ed_netmask = 0;
143 adent->ed_state = EDGE_OSPFRT_STATE_INIT;
144 adent->ed_next = (EHentry *)NULL;
145
146 if(n = add_entry(tp, adent, fc->fc_edgedb))
147 fc->fc_nedge_entry++;
148 return n;
149 }
150
151 static int
```

```
152 del_edgeent(task *tp, fanpdif_config_t *fc, u_long edgeid)
153 {
154 int n;
155 if(!tp || !fc)
156 return FAIL;
157
158 if(n = del_entry(tp, edgeid, fc->fc_edgedb))
159 fc->fc_nedge_entry--;
160 return n;
161 }
162
163 static EHentry *
164 get_edgeent(fanpdif_config_t *fc, u_long edgeid)
165 {
166 EHentry *hp;
167
168 for(hp = fc->fc_edgedb[EHASH(edgeid)]; hp; hp = hp->ed_next)
169 if(hp->ed_edgeid == edgeid)
170 return hp;
171 return NULL;
172 }
173
174 static void
175 varinit_edgedb(fanpdif_config_t *fc)
176 {
177 int i;
178 EHentry *hp;
179
180 for(i = 0; i < EDGE_HTSIZE; i++)
181 for(hp = fc->fc_edgedb[i]; hp ; hp = hp->ed_next){
182 hp->ed_state = EDGE_OSPFRT_STATE_INIT;
183 hp->ed_ref = 0;
184 hp->ed_ifaddr = 0;
185 hp->ed_ifindex = 0;
186 hp->ed_netaddr = 0;
187 hp->ed_netmask = 0;
188 }
189 }
190
191 static void
192 free_edgedb(task *tp, fanpdif_config_t *fc)
193 {
194 int i;
195 for(i = 0; i < EDGE_HTSIZE; i ++){
```

```
196 if(fc->fc_edgedb[i]){
197 del_listed_entrys(tp, fc->fc_edgedb[i]);
198 fc->fc_edgedb[i] = (EHentry *)NULL;
199 }
200 }
201 fc->fc_nedge_entry = 0;
202 }
203
204 void
205 dump_edgedb(task *tp, FILE *fp, fanpdif_config_t *fc)
206 {
207 int i;
208 EHentry *hp;
209
210 if(fc->fc_flags & FANPDIF_CONF_OSPF){
211 fprintf(fp, "OSPF Edgedb: ");
212 if(fc->fc_flags & FANPDIF_CONF_OSPF_ALLBR)
213 fprintf(fp, "all Area Border Router configuration\n");
214 else
215 fprintf(fp, "Manual configuration\n");
216 }
217 else if(fc->fc_flags & FANPDIF_CONF_BGP)
218 fprintf(fp, "BGP Edgedb:\n");
219
220 else
221 fprintf(fp, "something wrong\n");
222
223 fprintf(fp, "\t%d entrys\n", fc->fc_nedge_entry);
224 fprintf(fp, "\trouter id: reference count on ifaddr(rt_entry)\n");
225 for(i = 0; i < EDGE_HTSIZE; i ++){
226 if(!fc->fc_edgedb[i])
227 continue;
228 for(hp = fc->fc_edgedb[i]; hp ; hp = hp->ed_next){
229 fprintf(fp, "\t%A(%d): state %d(%d), reference %d on %s(%d):%A(%A/%d)\n",
230 sockbuild_in(0, hp->ed_edgeid),
231 i,
232 hp->ed_state,
233 hp->ed_option,
234 hp->ed_ref,
235 hp->ed_ifname,
236 hp->ed_ifindex,
237 sockbuild_in(0, hp->ed_ifaddr),
238 sockbuild_in(0, hp->ed_netaddr),
239 hp->ed_netmask
```

```
240 );
241 }
242 }
243 }
244
245 int
246 fanpdif_edgedb_term(task *tp)
247 {
248 (*fc_ospf.fc_free_edgedb)(tp, &fc_ospf);
249 /* (*fc_bgp.fc_free_edgedb)(tp, fc_bgp); */
250 return SUCCESS;
251 }
252
253 void
254 fanpdif_edgedb_dump(task *tp, FILE *fp)
255 {
256 if(!fp)
257 return;
258
259 fprintf(fp, "fanpdif_edgedb_dump:\n");
260 (*fc_ospf.fc_dump_edgedb)(tp, fp, &fc_ospf);
261 /* (*fc_bgp.fc_dump_edgedb)(tp, fp, &fc_bgp); */
262 }
263
264 int
265 fanpdif_edgedb_init(task *tp)
266 {
267 FILE *ep = 0;
268 char *p = 0, buf[124];
269 int line = 0;
270
271 fc_init(tp, &fc_ospf);
272 fc_ospf.fc_flags |= FANPDIF_CONF_OSPF;
273
274 ep = fopen(_EDGEROUTER_FILE, "r");
275 if(!ep)
276 return SUCCESS;
277
278 while(p = (char *)fgets(buf, 124, ep)){
279 char *sharp;
280 line++;
281 while(*p == ' ' || *p == '\t')
282 p++;
283 if(*p == '#' || *p == '\n')
```

```
284 continue;
285
286 if(sharp = (char *)index(p, '#'))
287 *sharp = '\0';
288
289 if(!fc_parse_and_doit(tp, p)){
290 fprintf(stderr, "error was occured at line %d\n",line);
291 fflush(stderr);
292 fclose(ep);
293 abort();
294 /* return FAIL; */
295 }
296 }
297 fclose(ep);
298 return SUCCESS;
299 }
300
301 static int
302 fc_get_param_ospf(char *p, u_long *edgeid, char *ifname)
303 {
304 struct in_addr addr;
305 char *q, *r;
306
307 if(!p || !edgeid)
308 return FAIL;
309
310 /*  get ifname first */
311 if(q = (char *)index(p, '(')){
312 char *s;
313 s = q;
314 s--;
315 while(*s == ' ' || *s == '\t')
316 s--;
317 *(s+1) = '\0';
318
319 q++;
320 while(*q == ' ' || *q == '\t')
321 q++;
322 r = (char *)index(q, ')');
323 if(!r)
324 return FAIL;
325 r--;
326 while(*r == ' ' || *r == '\t')
327 r--;
```

```
328 *(r + 1) = '\0';
329 if(r < q)
330 return FAIL;
331 strncpy(ifname, q, strlen(q));
332 }
333 else if(ifname)
334 *ifname = 0;
335
336 while(*p == ' ' || *p == '\t')
337 p++;
338
339 if(*p == '\n' || *p == '\0')
340 return FAIL;
341
342 if(!inet_aton(p, &addr))
343 return FAIL;
344 *edgeid = addr.s_addr;
345
346 return SUCCESS;
347 }
348 static int
349 fc_parse_and_doit(task *tp, char *command)
350 {
351 char *p, *q;
352 u_long edgeid;
353 char ifname[8];
354
355 bzero(ifname, sizeof(ifname));
356
357 /* "OSPF_EDGENODE: 203.178.141.42(ac0)" */
358 if(!strncmp(p, FC_OSPF_EDGENODE, strlen(FC_OSPF_EDGENODE))){
359 if(fc_ospf.fc_flags & FANPDIF_CONF_OSPF_ALLBR){
360 fprintf(stderr, "OSPF_ALLBR config can't \
361 be used with OSPF_EDGENODE config\n");
362 return FAIL;
363 }
364 if(!(q = (char *)index(p, ':'))){
365 fprintf(stderr, "Usage:\n%s: \
366 <edge routerid>[(ifname)]\n", FC_OSPF_EDGENODE);
367 return FAIL;
368 }
369 q++;
370 if(!fc_get_param_ospf(q, &edgeid, ifname)){
371 fprintf(stderr, "Usage:\n%s: \
```

```
372 <edge routerid>[(ifname)]\n", FC_OSPF_EDGENODE);
373 return FAIL;
374 }
375 if(!(*fc_ospf.fc_add_edgeent)(tp, &fc_ospf, edgeid, ifname)){
376 fprintf(stderr, "same edgeid(%A) is \
377 already registered!\n",sockbuild_in(0, edgeid));
378 return FAIL;
379 }
380
381 return SUCCESS;
382 }
383
384 /* "OSPF_ALLBR: 0.0.0.0" */
385 else if(!strncmp(p, FC_OSPF_ALLBR, strlen(FC_OSPF_ALLBR))){
386 if(fc_ospf.fc_nedge_entry){
387 fprintf(stderr, "OSPF_ALLBR config can't \
388 be used with OSPF_EDGENODE config\n");
389 return FAIL;
390 }
391 if(!(q = (char *)index(p, ':'))){
392 fprintf(stderr, "Usage:\n%s: \
393 <area id>[(ifname)]\n", FC_OSPF_ALLBR);
394 return FAIL;
395 }
396 q++;
397
398 if(!fc_get_param_ospf(q, &edgeid, ifname)){
399 fprintf(stderr, "Usage:\n%s: \
400 <area id>[(ifname)]\n", FC_OSPF_ALLBR);
401 return FAIL;
402 }
403 fc_optdata_ospf.areaid = edgeid;
404 strcpy(fc_optdata_ospf.ifname, ifname);
405 fc_ospf.fc_optdata = &fc_optdata_ospf;
406 fc_ospf.fc_flags |= FANPDIF_CONF_OSPF_ALLBR;
407 return SUCCESS;
408 }
409
410 /* unknown command */
411 fprintf(stderr, "Unknown command\n");
412 return FAIL;
413 }
```

```
 1 /*
 2 fanpdif_rtdb.h: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #ifndef _FANPDIF_RTDB_H_
 7 #define _FANPDIF_RTDB_H_
 8
 9 /*
10 routing table cache database
11 */
12 typedef struct _rtdb {
13 struct _rtdb *re_next; /* double linked */
14 struct _rtdb *re_prev; /* list  */
15 struct _rtdb *re_hnext; /* for using hashtable */
16 u_long re_hkey; /* hash key */
17 u_long re_daddr; /* destination address */
18 u_long re_dmask; /* destination netmask */
19 u_long re_nhop; /* next hop */
20 u_long re_advrt; /* advertize router */
21 union {
22 EHentry *_edgertr; /*  OSPF transit router id */
23 u_long  _bgppeer; /*  BGPPEER */
24 } _transrtr;
25 char *re_ifname; /* interface name */
26 } RTDBentry;
27
28 #define re_edgertr _transrtr._edgertr
29 #define re_bgppeer _transrtr._bgppeer
30
31 #define RTDB_HTSIZE 33
32 #define RTDB_HASH(x) ((x) % RTDB_HTSIZE)
33 typedef struct {
34 RTDBentry *rtdb_top;
35 RTDBentry *rtdb_last;
36 RTDBentry **rtdb_ht; /* hash table */
37 int  rtdb_entrys; /* number of entrys */
38 } RTDB;
39
40 void fanpdif_rtdb_init(task *);
41 void rtdb_init(task *, RTDB *);
42 void rtdb_alloc(task *, RTDB *);
43 void rtdb_free(task *, RTDB *);
44 void rtdb_copy(RTDB *, RTDB *);
```

```
45 void rtdb_dump(FILE *, RTDB *);
46
47 RTDBentry *rtdbent_alloc(task *);
48 void rtdbent_free(task *, RTDBentry *);
49 void rtdbent_release(task *, RTDBentry *);
50
51 /*
52 if same entry already addred, just return 0.
53 else if success, return 1.
54 */
55 int rtdbent_add(RTDB *,RTDBentry *);
56 RTDBentry *rtdbent_get(RTDB *, u_long, u_long);
57 /*
58 take out rtdbentry from hash table & linked list in RTDB.
59 */
60 void rtdbent_del(RTDB *, RTDBentry *);
61
62 /*
63 if same entry, return 0, else -1.
64 */
65 int rtdbent_cmp(RTDBentry *, RTDBentry *);
66
67 #endif /* _FANPDIF_RTDB_H_ */
```

```
 1 /*
 2 fanpdif_rtdb.c: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #include "include.h"
 7 #include "fanpdif.h"
 8 #include "fanpdif_edgedb.h"
 9 #include "fanpdif_rt.h"
10 #include "fanpdif_rtdb.h"
11 #include "arpa/inet.h"
12
13 static void rtdbent_insque(RTDB *, RTDBentry *);
14 static void rtdbent_remque(RTDB *, RTDBentry *);
15
16 static RTDB free_entrys;
17 static int allocated_rtdbentrys = 0;
18
19 static int duplicate_count = 0;
```

```
20
21 /* for fanpdif task */
22 void
23 fanpdif_rtdb_init(task *tp)
24 {
25 if(!tp)
26 return;
27 if(free_entrys.rtdb_top)
28 rtdb_free(tp, &free_entrys);
29
30 bzero(&free_entrys,  sizeof(RTDB));
31 }
32
33 /* for initialize rtdb */
34 void
35 rtdb_init(task *tp, RTDB *rtdb)
36 {
37 if(!tp || !rtdb)
38 return;
39
40 rtdb->rtdb_top = rtdb->rtdb_last = (RTDBentry *)NULL;
41 rtdb->rtdb_entrys = 0;
42 if(rtdb->rtdb_ht)
43 bzero(rtdb->rtdb_ht, sizeof(RTDBentry *) * RTDB_HTSIZE);
44 }
45
46
47 void
48 rtdb_alloc(task *tp, RTDB *rp)
49 {
50 if(!tp || !rp)
51 return;
52 rp->rtdb_top = (RTDBentry *)NULL;
53 rp->rtdb_last = (RTDBentry *)NULL;
54 rp->rtdb_ht = (RTDBentry **)fanpdif_malloc(tp,
55 sizeof(RTDBentry *) * RTDB_HTSIZE);
56 rp->rtdb_entrys = 0;
57 assert(rp->rtdb_ht != (RTDBentry **)NULL);
58 bzero(rp->rtdb_ht, sizeof(RTDBentry *) * RTDB_HTSIZE);
59 }
60
61 void rtdb_free(task *tp, RTDB *rp)
62 {
63 RTDBentry *rtp, *rq;
```

```
 64 if(!rp)
 65 return;
 66
 67 for(rtp = rp->rtdb_top; rtp ; rtp = rq){
 68 rq = rtp->re_next;
 69 rtdbent_del(rp, rtp);
 70 rtdbent_free(tp, rtp);
 71 }
 72
 73 if(rp->rtdb_ht)
 74 fanpdif_free(tp, rp->rtdb_ht, sizeof(RTDBentry *) * RTDB_HTSIZE);
 75
 76 bzero(rp, sizeof(RTDB));
 77 }
 78
 79 /*  put all entry into freelist */
 80 void rtdb_release_entrys(task *tp, RTDB *rp)
 81 {
 82 RTDBentry *rtp, *rq;
 83 if(!rp)
 84 return;
 85
 86 for(rtp = rp->rtdb_top; rtp ; rtp = rq){
 87 rq = rtp->re_next;
 88 rtdbent_del(rp, rtp);
 89 rtdbent_release(tp, rtp);
 90 }
 91
 92 rtdb_init(tp, rp);
 93 }
 94
 95 void rtdb_copy(RTDB *rp, RTDB *rq)
 96 {
 97 if(!rp || !rq)
 98 return;
 99
100 rq->rtdb_top = rp->rtdb_top;
101 rq->rtdb_last = rp->rtdb_last;
102 rq->rtdb_entrys = rp->rtdb_entrys;
103
104 if(rp->rtdb_ht && rq->rtdb_ht)
105 bcopy(rp->rtdb_ht, rq->rtdb_ht, sizeof(RTDBentry *) * RTDB_HTSIZE);
106 }
107
```

```
108 void rtdb_dump(FILE *fp, RTDB *rtdb)
109 {
110 RTDBentry *rtp;
111
112 if(!fp || !rtdb)
113 return;
114 fprintf(fp, "\trtdb_dump::\t%d/%d entrys, %d free, dup %d times\n", rtdb->rtdb_entrys, allocated
115
116 fprintf(fp, "destaddr/mask(hashkey)\t\t-> egress addr(advrouter, nexthop)\n");
117 for(rtp = rtdb->rtdb_top; rtp; rtp = rtp->re_next){
118 fprintf(fp, "%A/%d(%d)\t\t-> %A/%s(%A, %A)\n",
119 sockbuild_in(0, rtp->re_daddr),
120 masktomasklen(rtp->re_dmask),
121 rtp->re_hkey,
122 sockbuild_in(0, rtp->re_edgertr?
123 rtp->re_edgertr->ed_edgeid
124 : 0),
125 rtp->re_ifname,
126 sockbuild_in(0, rtp->re_advrt),
127 sockbuild_in(0, rtp->re_nhop)
128 );
129 }
130 }
131
132 /*
133 if free_entrys has free entry, return it, else malloc called
134 */
135 RTDBentry *
136 rtdbent_alloc(task *tp)
137 {
138 RTDBentry *pp;
139
140 if(!free_entrys.rtdb_top){
141 pp = fanpdif_malloc(tp, sizeof(RTDBentry));
142 if(!pp)
143 return pp;
144 allocated_rtdbentrys++;
145 }
146 else {
147 pp = free_entrys.rtdb_top;
148 rtdbent_del(&free_entrys, free_entrys.rtdb_top);
149 free_entrys.rtdb_entrys--;
150 }
151
```

```
152 bzero(pp, sizeof(RTDBentry));
153 return pp;
154 }
155
156 /*
157 free this entry
158 */
159 void
160 rtdbent_free(task *tp, RTDBentry *rent)
161 {
162 allocated_rtdbentrys--;
163 fanpdif_free(tp, rent, sizeof(RTDBentry));
164 }
165
166 /*
167 put rent into free_entrys as unused entry
168 */
169 void
170 rtdbent_release(task *tp, RTDBentry *rent)
171 {
172 rtdbent_insque(&free_entrys, rent);
173 free_entrys.rtdb_entrys++;
174 }
175
176 /*
177 just insert dlink queue
178 */
179 static void
180 rtdbent_insque(RTDB *rrp, RTDBentry *rent)
181 {
182 if(!rrp->rtdb_top){
183 rrp->rtdb_top = rrp->rtdb_last = rent;
184 rent->re_next = rent->re_prev = (RTDBentry *)NULL;
185 return;
186 }
187
188 rrp->rtdb_last->re_next = rent;
189 rent->re_prev = rrp->rtdb_last;
190 rent->re_next = (RTDBentry *)NULL;
191 rrp->rtdb_last = rent;
192 return;
193 }
194
195 /*
```

```
196 just remove entry from dlink queue
197 */
198 static void
199 rtdbent_remque(RTDB *rrp, RTDBentry *rent)
200 {
201 if(!rrp || !rent)
202 return;
203 /* take out rent from queue */
204 if(rent == rrp->rtdb_top)
205 rrp->rtdb_top = rrp->rtdb_top->re_next;
206 else
207 rent->re_prev->re_next = rent->re_next;
208
209 if(rent == rrp->rtdb_last)
210 rrp->rtdb_last = rrp->rtdb_last->re_prev;
211 else
212 rent->re_next->re_prev = rent->re_prev;
213
214 rent->re_next = rent->re_prev = (RTDBentry *)NULL;
215 }
216
217 /*
218 - insert rent into dlink queue
219 - put into hash table
220 - increment rtdb->rtdb_entrys
221 */
222 int
223 rtdbent_add(RTDB *rrp, RTDBentry *rent)
224 {
225 RTDBentry *rep, *req;
226
227 if(!rent || !rrp ||!rrp->rtdb_ht)
228 return 0;
229
230 rent->re_hkey = RTDB_HASH(rent->re_daddr+rent->re_dmask);
231 rep = rrp->rtdb_ht[rent->re_hkey];
232
233 if(!rep){
234 rrp->rtdb_ht[rent->re_hkey] = rent;
235 rtdbent_insque(rrp, rent);
236 rrp->rtdb_entrys++;
237 return 1;
238 }
239
```

```
240 for(req = 0;; req = rep, rep = rep->re_hnext){
241 if(rep->re_daddr == rent->re_daddr &&
242    rep->re_dmask == rent->re_dmask){
243    /* if same rent is exist, do nothing */
244 #if 0
245 /* if same rent is exist, insque before old entry */
246     if(!req){
247 rrp->rtdb_ht[rent->re_hkey] = rent;
248 rent->re_hnext = rep;
249 rtdbent_insque(rrp, rent);
250 rrp->rtdb_entrys++;
251 return 1;
252 }
253 else {
254 req->re_hnext = rent;
255 rent->re_hnext = rep;
256 rtdbent_insque(rrp, rent);
257 rrp->rtdb_entrys++;
258 return 1;
259 }
260 #endif
261 /*abort(); */
262 duplicate_count++;
263 return 0;
264 }
265
266 if(!rep->re_hnext)
267 break;
268 }
269
270 rep->re_hnext = rent;
271 rent->re_hnext = NULL;
272 rtdbent_insque(rrp, rent);
273 rrp->rtdb_entrys++;
274 return 1;
275 }
276
277
278 /* return RTDBentry whitch is matched  */
279 RTDBentry *
280 rtdbent_get(RTDB *rrp, u_long daddr, u_long dmask)
281 {
282 RTDBentry *rep;
283
```

```
284 if(!rrp || !rrp->rtdb_top || !rrp->rtdb_ht)
285 return NULL;
286
287 rep = rrp->rtdb_ht[RTDB_HASH(daddr+dmask)];
288
289 if(!rep)
290 return NULL;
291
292 for(; rep ; rep = rep->re_hnext)
293 if(rep->re_daddr == daddr &&
294    rep->re_dmask == dmask)
295 return rep;
296 return NULL;
297 }
298
299 /*
300 - remove rent from dlink queue
301 - take out rent from hash table
302 - rtdb->rtdb_entrys decrement
303 */
304 void
305 rtdbent_del(RTDB *rrp, RTDBentry *rent)
306 {
307 RTDBentry *rep;
308
309 if(!rent || !rrp || !rrp->rtdb_top)
310 return;
311
312 /* take out rent from linked list */
313 rtdbent_remque(rrp, rent);
314
315 if(!rrp->rtdb_ht)
316 return;
317
318 /* take out rent from hash table */
319 rep = rrp->rtdb_ht[RTDB_HASH(rent->re_daddr+rent->re_dmask)];
320 if(!rep)
321 return;
322
323 if(rep == rent){
324 rrp->rtdb_ht[RTDB_HASH(rent->re_daddr+rent->re_dmask)] =
325 rep->re_hnext;
326 rrp->rtdb_entrys--;
327 return;
```

```
328 }
329
330 for(; rep ; rep = rep->re_hnext)
331 if(rep->re_hnext == rent){
332 rep->re_hnext = rent->re_hnext;
333 rrp->rtdb_entrys--;
334 return;
335 }
336 return;
337 }
338
339 /*
340 compare two RTDBentrys
341 */
342 int
343 rtdbent_cmp(RTDBentry *r1, RTDBentry *r2)
344 {
345 if(!r1 || !r2)
346 return -1;
347 if(r1->re_daddr == r2->re_daddr &&
348    r1->re_dmask == r2->re_dmask &&
349    r1->re_nhop == r2->re_nhop &&
350    r1->re_edgertr == r2->re_edgertr)
351 return 0;
352 return 1;
353 }
```

```
 1 /*
 2 fanpdif_ospf.h: hiddy@sfc.wdie.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #ifndef _FANPDIF_OSPF_H_
 7 #define _FANPDIF_OSPF_H_
 8
 9 #define FANPDIF_OSPF
10
11 /*
12 these functions provide interface for
13 using OSPF Lisk State Database
14 */
15
16 void fanpdif_ospf_init(task *);
```

```
17 void fanpdif_ospf_dump(FILE *);
18 void fanpdif_ospf_terminate(task *);
19 int fanpdif_ospf_putall(task *);
20 void fanpdif_ospf_flash(task *, rt_list *);
21
22 /* using in ospf */
23 struct LSDB *fanpdif_ospf_ptp(u_long, struct LSDB *);
24 u_long fanpdif_ospf_get_ifaddr(struct LSDB *, struct LSDB *);
25
26 #endif /* _FANPDIF_OSPF_H_ */
```

```
 1 /*
 2 fanpdif_ospf.c: hiddy@sfc.wide.ad.jp
 3 $Id: appendix.tex,v 1.5 1999/02/21 23:59:18 hiddy Exp $
 4 */
 5
 6 #include "include.h"
 7 #include "inet.h"
 8 #include "ospf.h"
 9 #include "fanpdif.h"
10 #include "fanpdif_edgedb.h"
11 #include "fanpdif_msg.h"
12 #include "fanpdif_rt.h"
13 #include "fanpdif_rtdb.h"
14 #include "fanpdif_ospf.h"
15
16 #define SUCCESS 1
17 #define FAIL  0
18
19 #define EDGE_IFNAME_OK(e, r)  (!strncmp((r)->rt_ifaps[0]->ifa_link->ifl_name, \
20 e->ed_ifname, strlen(e->ed_ifname)))
21
22 /* statistics */
23 static int edge_routes = 0; /* total routes for transite edge route */
24 static u_long rtdb_ctl[GRT_CHANGE + 1];
25
26 static RTDB rtdb_ospf; /* all data entrys which send to fanpd */
27 static RTDB rtdb_candidate; /* all data entrys which send to fanpd */
28 static void candidate_add(task *, RTDBentry *);
29 static void candidate_del(task *, u_long dst, u_long dmask);
30 static void candidate_sync(task *);
31
32 static int fanpdif_ospf_edge_rtlookup(void);
```

```
33 static int fmsg_commit(task *tp, RTDBentry *rep, u_char mflag);
34 static int make_rtdbentry_rt(task * tp, RTDBentry *rnew, rt_entry *rt);
35
36 void
37 fanpdif_ospf_init(task *tp)
38 {
39 fanpdif_rtdb_init(tp);
40 edge_routes = 0;
41 bzero(rtdb_ctl, sizeof(rtdb_ctl));
42 bzero(&rtdb_ospf, sizeof(RTDB));
43 rtdb_alloc(tp, &rtdb_ospf);
44 bzero(&rtdb_candidate, sizeof(RTDB));
45 rtdb_alloc(tp, &rtdb_candidate);
46 (*fc_ospf.fc_varinit_edgedb)(&fc_ospf);
47 }
48
49 void
50 fanpdif_ospf_terminate(task *tp)
51 {
52 if(!tp)
53 return;
54 (*fc_ospf.fc_varinit_edgedb)(&fc_ospf);
55 rtdb_free(tp, &rtdb_ospf);
56 rtdb_free(tp, &rtdb_candidate);
57 }
58
59 void
60 fanpdif_ospf_dump(FILE *fp)
61 {
62 fprintf(fp, "fanpdif_ospf_dump:\n\ttotal edge transit routes: %d\n",
63 edge_routes);
64 fprintf(fp, "\trtdb control:\n");
65 fprintf(fp, "\t\tGRT_ADD: %d\n", rtdb_ctl[GRT_ADD]);
66 fprintf(fp, "\t\tGRT_DEL: %d\n", rtdb_ctl[GRT_DEL]);
67 fprintf(fp, "\t\tGRT_CHANGE: %d\n", rtdb_ctl[GRT_CHANGE]);
68 fprintf(fp, "\tospf rtdb entrys: %d\n", rtdb_ospf.rtdb_entrys);
69 rtdb_dump(fp, &rtdb_ospf);
70 fprintf(fp, "\n\tospf candidate entrys\n");
71 rtdb_dump(fp, &rtdb_candidate);
72 }
73
74
75 /* put all entry on RTDB to fanpd */
76 int
```

```
 77 fanpdif_ospf_putall(task *tp)
 78 {
 79 int n;
 80 RTDBentry *rep = rtdb_ospf.rtdb_top;
 81
 82 if(!tp || tp->task_socket < 0)
 83 return FAIL;
 84
 85 for(; rep ; rep = rep->re_next){
 86 n = fanpdif_msg_add(tp->task_socket,
 87 AF_INET,
 88 FANPDIF_MSGPROTO_OSPF,
 89 rep->re_daddr,
 90 masktomasklen(rep->re_dmask),
 91 rep->re_edgertr->ed_ifaddr,
 92 rep->re_nhop,
 93 rep->re_edgertr->ed_ifindex,
 94 rep->re_edgertr->ed_netaddr,
 95 rep->re_edgertr->ed_netmask
 96 );
 97
 98 if(n <= 0)
 99 return FAIL;
100 }
101
102 return SUCCESS;
103 }
104
105 /*
106 when routing table is updated by some protocol,
107 this function will be called
108 */
109 void
110 fanpdif_ospf_flash(task *tp, rt_list *rtl)
111 {
112 int n;
113 register rt_head *rth;
114 register RTDBentry *rep = 0, *rent = 0;
115
116 if(!tp || !rtl)
117 return ;
118
119 fanpdif_ospf_edge_rtlookup();
120 candidate_sync(tp);
```

```
121
122     RT_LIST(rth, rtl, rt_head) {
123 register rt_entry *rt;
124 register rt_entry *rtf = rth->rt_forw;
125 register RTDBentry *crep = 0;
126
127 rep = rtdbent_get(&rtdb_ospf, sock2ip(rth->rth_dest),
128 sock2ip(rth->rth_dest_mask));
129 n = 1;
130
131 /* exist acitve route */
132 if(rtf != (rt_entry *)&rth->rt_forw &&
133   (rt = rth->rth_active) &&
134   (rt->rt_gwp->gw_proto == RTPROTO_OSPF ||
135    rt->rt_gwp->gw_proto == RTPROTO_OSPF_ASE)){
136 rent = rtdbent_alloc(tp);
137
138 /* not found old entry */
139 if(!rep){
140 /* transit edge router? */
141 if(make_rtdbentry_rt(tp, rent, rt) &&
142    rent->re_edgertr->ed_state == EDGE_OSPFRT_STATE_FOUNDRT){
143 candidate_del(tp, rent->re_daddr, rent->re_dmask);
144 n = fmsg_commit(tp, rent, GRT_ADD);
145 continue;
146 }
147
148 else{ /* no transit edge router */
149 candidate_add(tp, rent);
150 continue;
151 }
152 }
153 else{
154 if(make_rtdbentry_rt(tp, rent, rt) &&
155    rent->re_edgertr->ed_state == EDGE_OSPFRT_STATE_FOUNDRT){
156 /* same rtdbentry exist? */
157 if(!rtdbent_cmp(rep, rent)){
158 rtdbent_release(tp, rent);
159 continue;
160 }
161 else{
162 candidate_del(tp, rent->re_daddr, rent->re_dmask);
163 n = fmsg_commit(tp, rent, GRT_CHANGE);
164 continue;
```

```
165 }
166 }
167 else{
168 n = fmsg_commit(tp, rep, GRT_DEL);
169 candidate_add(tp, rent);
170 continue;
171 }
172 }
173 }
174 else if(rep){
175 n = fmsg_commit(tp, rep, GRT_DEL);
176 }
177 }RT_LIST_END(rth, rtl, rt_head);
178
179 return;
180 }
181
182 static void
183 candidate_add(task *tp, RTDBentry *rent)
184 {
185 RTDBentry *rep;
186
187 if(!tp || !rent)
188 return;
189 rep = rtdbent_get(&rtdb_candidate, rent->re_daddr, rent->re_dmask);
190 if(rep){
191 rtdbent_del(&rtdb_candidate, rep);
192 rtdbent_release(tp, rep);
193 }
194 rtdbent_add(&rtdb_candidate, rent);
195 }
196
197 static void
198 candidate_del(task *tp, u_long dst, u_long dmask)
199 {
200 RTDBentry *rep;
201
202 rep = rtdbent_get(&rtdb_candidate, dst, dmask);
203
204 if(rep){
205 rtdbent_del(&rtdb_candidate, rep);
206 rtdbent_release(tp, rep);
207 }
208 }
```

```
209
210 static void
211 candidate_sync(task *tp)
212 {
213 int n;
214 RTDBentry *rep, *req, *rent, *active;
215 rt_entry *rt;
216
217 for(rep = rtdb_candidate.rtdb_top; rep ; rep = req){
218 req = rep->re_next;
219 rt = fanpdif_activert_lookup(tp, rep->re_daddr, rep->re_dmask, RTPROTO_OSPF);
220 if(!rt)
221 rt = fanpdif_activert_lookup(tp, rep->re_daddr, rep->re_dmask, RTPROTO_OSPF_ASE);
222 if(!rt)
223 continue;
224 rent = rtdbent_alloc(tp);
225
226 if(make_rtdbentry_rt(tp, rent, rt) &&
227    rent->re_edgertr->ed_state == EDGE_OSPFRT_STATE_FOUNDRT){
228 rtdbent_del(&rtdb_candidate, rep);
229 rtdbent_release(tp, rep);
230 n = fmsg_commit(tp, rent, GRT_ADD);
231 continue;
232 }
233
234 /*
235   here we can only release.
236   if we add new entry after release, may be loop forever
237  */
238 rep->re_nhop = rent->re_nhop;
239 rep->re_advrt = rent->re_advrt;
240 rep->re_edgertr = rent->re_edgertr;
241 rep->re_ifname = rent->re_ifname;
242 rtdbent_release(tp, rent);
243
244 #if 0
245 /* ok, install this rtdbentry to rtdb_ospf */
246 if(active = rtdbent_get(&rtdb_ospf, rent->re_daddr, rent->re_daddr)){
247 if(!rtdbent_cmp(rent, active)){
248 rtdbent_release(tp, rent);
249 continue;
250 }
251
252 n = fmsg_commit(tp, rent, GRT_CHANGE);
```

```
253 }
254 else{
255 n = fmsg_commit(tp, rent, GRT_ADD);
256 }
257 #endif
258 }
259 }
260
261 static int
262 fanpdif_ospf_edge_rtlookup(void)
263 {
264 int i;
265 EHentry *hp;
266 rt_entry *rtent;
267 int found = 0;
268
269 for(i = 0; i < EDGE_HTSIZE; i ++){
270 if(!fc_ospf.fc_edgedb[i])
271 continue;
272 for(hp = fc_ospf.fc_edgedb[i]; hp ; hp = hp->ed_next){
273 if(!hp->ed_ifaddr)
274 continue;
275
276 /* this edgenode interface addr is'nt still found */
277 if(hp->ed_state == EDGE_OSPFRT_STATE_INIT)
278 continue;
279
280 /* this edgenode interface is fixed */
281 if(hp->ed_state == EDGE_OSPFRT_STATE_FOUNDRT)
282 continue;
283
284 rtent = rt_lookup(0xffffffff,
285   0,
286   sockbuild_in(0, hp->ed_ifaddr),
287   RTPROTO_BIT(RTPROTO_OSPF));
288     if(!rtent){
289     findroute(rtent, hp->ed_ifaddr, 0xffffffff);
290     if(!rtent)
291 continue;
292
293 }
294
295 /* check the interface of edgerouter */
296 if(!EDGE_IFNAME_OK(hp, rtent)){
```

```
297 hp->ed_state = EDGE_OSPFRT_STATE_INIT;
298 hp->ed_netaddr = 0;
299 hp->ed_netmask = 0;
300 }
301 /* this is correct interface address & network */
302 else {
303 hp->ed_state = EDGE_OSPFRT_STATE_FOUNDRT;
304 hp->ed_netaddr = sock2ip(rtent->rt_dest);
305 hp->ed_netmask = masktomasklen(sock2ip(rtent->rt_dest_mask));
306 hp->ed_ifindex = rtent->rt_ifaps[0]->ifa_link->ifl_index;
307 found = 1;
308 }
309 }
310 }
311 return found;
312 }
313
314 /*
315 if daddr is ptp link interface address,
316 route is advertised by the neighbor of host who
317 has interface whitch numbered 'daddr'.
318 return lsdb for router which has interface numbered 'daddr'.
319 */
320 static int fanpdif_ospf_have_ptplink(u_long lnk, struct LSDB *v);
321 struct LSDB *
322 fanpdif_ospf_ptp(u_long daddr, struct LSDB *v)
323 {
324 int i, cnt;
325 struct LSDB *pv;
326 struct RTR_LA_PIECES *linkp, *nextlinkp;
327
328 cnt = ntohs(DB_RTR(v)->lnk_cnt);
329
330 for(i = 1, linkp = (struct RTR_LA_PIECES *)&(DB_RTR(v)->link);
331 i <= cnt; i++, linkp = nextlinkp){
332 nextlinkp = (i < cnt)?
333 (struct RTR_LA_PIECES *)((long)linkp + RTR_LA_PIECES_SIZE +
334 ((linkp->metric_cnt)*RTR_LA_METRIC_SIZE)):
335 (struct RTR_LA_PIECES *)0;
336
337 if(linkp->con_type == RTR_IF_TYPE_RTR){
338 pv = FindLSA(v->lsdb_area, linkp->lnk_id, linkp->lnk_id, LS_RTR);
339 if(!pv)
340 continue;
```

```
341 if(fanpdif_ospf_have_ptplink(daddr, pv))
342 return pv;
343 }
344 }
345
346 return 0;
347 }
348
349 /* called by only fanpdif_ospf_ptp */
350 static int fanpdif_ospf_have_ptplink(u_long lnk, struct LSDB *v)
351 {
352 int i, cnt;
353 struct RTR_LA_PIECES *linkp, *nextlinkp;
354
355 cnt = ntohs(DB_RTR(v)->lnk_cnt);
356
357 for(i = 1, linkp = (struct RTR_LA_PIECES *)&(DB_RTR(v)->link);
358 i <= cnt; i++, linkp = nextlinkp){
359 nextlinkp = (i < cnt)?
360 (struct RTR_LA_PIECES *)((long)linkp + RTR_LA_PIECES_SIZE +
361 ((linkp->metric_cnt)*RTR_LA_METRIC_SIZE)):
362 (struct RTR_LA_PIECES *)0;
363
364 if(linkp->con_type == RTR_IF_TYPE_RTR){
365 if(lnk == linkp->lnk_data)
366 return lnk;
367 }
368 }
369 return 0;
370 }
371
372
373 static u_long
374 find_net_linkp(u_long edgenet, u_long netmask, struct LSDB *v);
375 u_long
376 fanpdif_ospf_get_ifaddr(struct LSDB *edgev,struct LSDB *parentv)
377 {
378 int i, cnt;
379 struct RTR_LA_PIECES *linkp, *nextlinkp;
380 rt_entry *rtent;
381 u_long intface = 0;
382
383 if(!edgev || !parentv)
384 return 0;
```

```
385
386 /* XXXX */
387 if(LS_TYPE(parentv) != LS_RTR && LS_TYPE(parentv) != LS_NET)
388  return 0;
389
390 else if(LS_TYPE(parentv) == LS_NET){
391 intface = find_net_linkp(DB_NET(parentv)->ls_hdr.ls_id,
392  DB_NET(parentv)->net_mask,
393  edgev);
394 if(!intface)
395   return 0;
396 }
397
398 else { /* LS_RTR */
399 cnt = ntohs(DB_RTR(edgev)->lnk_cnt);
400
401 for(i = 1, linkp = (struct RTR_LA_PIECES *)&(DB_RTR(edgev)->link);
402 i <= cnt; i++, linkp = nextlinkp){
403 nextlinkp = (i < cnt)?
404 (struct RTR_LA_PIECES *)((long)linkp + RTR_LA_PIECES_SIZE +
405 ((linkp->metric_cnt)*RTR_LA_METRIC_SIZE)):
406 (struct RTR_LA_PIECES *)0;
407
408 if(linkp->lnk_id == LS_ID(parentv)){
409 intface = linkp->lnk_data;
410 break;
411 }
412 }
413
414 if(!linkp || !intface)
415   return 0;
416 }
417
418 return intface;
419 }
420
421 /* find out same network link in LSDB v. */
422 static u_long
423 find_net_linkp(u_long edgenet, u_long netmask, struct LSDB *v)
424 {
425 int i, cnt;
426 struct RTR_LA_PIECES *linkp, *nextlinkp;
427
428 edgenet = edgenet & netmask;
```

```
429
430 cnt = ntohs(DB_RTR(v)->lnk_cnt);
431
432 for(i = 1, linkp = (struct RTR_LA_PIECES *)&(DB_RTR(v)->link);
433 i <= cnt; i++, linkp = nextlinkp){
434 nextlinkp = (i < cnt)?
435 (struct RTR_LA_PIECES *)((long)linkp + RTR_LA_PIECES_SIZE +
436 ((linkp->metric_cnt)*RTR_LA_METRIC_SIZE)):
437 (struct RTR_LA_PIECES *)0;
438
439 if(linkp->con_type == RTR_IF_TYPE_TRANS_NET){
440 if(edgenet == (linkp->lnk_id & netmask))
441 return linkp->lnk_data;
442 }
443 }
444
445 return 0;
446 }
447
448 /*
449 build RTDBentry from rt_entry
450 */
451 static int
452 make_rtdbentry_rt(task * tp, RTDBentry *rnew, rt_entry *rt)
453 {
454 rt_entry *rtent;
455 struct LSDB *v, *rv, *ptp;
456 EHentry *ehent;
457
458 v = rv = ptp = (struct LSDB *)NULL;
459
460 if(!rnew || !rt)
461 return FAIL;
462
463 if((rt->rt_gwp && rt->rt_gwp->gw_proto != RTPROTO_OSPF &&
464    rt->rt_gwp->gw_proto != RTPROTO_OSPF_ASE) ||
465    !rt->rt_data)
466 return FAIL;
467
468 /*  get transit edge router */
469 rnew->re_daddr = sock2ip(rt->rt_dest);
470 rnew->re_dmask = sock2ip(rt->rt_dest_mask);
471 rnew->re_nhop = sock2ip(rt->rt_routers[0]);
472 rnew->re_advrt = ORT_ADVRTR(rt);
```

```
473 rnew->re_ifname = rt->rt_ifaps[0]->ifa_link->ifl_name;
474
475 switch(rt->rt_gwp->gw_proto){
476 case RTPROTO_OSPF:
477 v = ORT_V(rt);
478 /* case of point to point link */
479 if(v && (LS_TYPE(v) == LS_RTR || LS_TYPE(v) == LS_STUB) &&
480    rnew->re_dmask == 0xffffffff){
481    (LS_TYPE(v) == LS_STUB)?
482 v = FindLSA(v->lsdb_area, rnew->re_advrt, rnew->re_advrt, LS_RTR):
483 v;
484 ptp = fanpdif_ospf_ptp(rnew->re_daddr, v);
485 if(ptp)
486 v = ptp;
487 }
488 break;
489 case RTPROTO_OSPF_ASE:
490 rv = ORT_V(rt);
491 if(rv->lsdb_edgenode)
492 v = rv;
493 else {
494 v = rv->lsdb_border;
495 rv->lsdb_edgenode = v->lsdb_edgenode;
496 }
497 break;
498 default:
499 return FAIL;
500 }
501
502 /*
503 FILTER
504 this route does'nt transit edge router
505 or not throgh CSR cloud
506 */
507 if(v && v->lsdb_edgenode){
508 rnew->re_edgertr = v->lsdb_edgenode;
509 return SUCCESS;
510 }
511 else{
512 rnew->re_edgertr = 0;
513 return FAIL;
514 }
515 }
516
```

```
517
518 static int
519 fmsg_commit(task *tp, RTDBentry *rep, u_char mflag)
520 {
521 RTDBentry *rent;
522 int n;
523
524 if(!tp || !rep)
525 abort();
526
527 rtdb_ctl[mflag]++;
528 n = 1;
529 switch(mflag){
530 case GRT_ADD:
531 rtdbent_add(&rtdb_ospf, rep);
532 rep->re_edgertr->ed_ref++;
533 edge_routes++;
534
535 if(fanpdif_state() == FANPDIF_STAT_CONNECT){
536 n = fanpdif_msg_add(tp->task_socket,
537 AF_INET,
538 FANPDIF_MSGPROTO_OSPF,
539 rep->re_daddr,
540 masktomasklen(rep->re_dmask),
541 rep->re_edgertr->ed_ifaddr,
542 rep->re_nhop,
543 rep->re_edgertr->ed_ifindex,
544 rep->re_edgertr->ed_netaddr,
545 rep->re_edgertr->ed_netmask
546 );
547 }
548 break;
549
550 case GRT_DEL:
551 if(fanpdif_state() == FANPDIF_STAT_CONNECT){
552 n = fanpdif_msg_del(tp->task_socket,
553 AF_INET,
554 FANPDIF_MSGPROTO_OSPF,
555 rep->re_daddr,
556 masktomasklen(rep->re_dmask)
557 );
558 }
559 rtdbent_del(&rtdb_ospf, rep);
560 rtdbent_release(tp, rep);
```

```
561 rep->re_edgertr->ed_ref--;
562 edge_routes--;
563 break;
564
565 case GRT_CHANGE:
566 rent = rtdbent_get(&rtdb_ospf, rep->re_daddr, rep->re_dmask);
567 if(!rent)
568 abort();
569 if(!rtdbent_cmp(rent, rep)){
570 rtdbent_release(tp, rep);
571 break;
572 }
573 if(rep->re_edgertr != rent->re_edgertr){
574 rent->re_edgertr->ed_ref--;
575 rep->re_edgertr->ed_ref++;
576 }
577
578 rtdbent_del(&rtdb_ospf, rent);
579 rtdbent_release(tp, rent);
580 rtdbent_add(&rtdb_ospf, rep);
581 if(fanpdif_state() == FANPDIF_STAT_CONNECT){
582 n = fanpdif_msg_chg(tp->task_socket,
583 AF_INET,
584 FANPDIF_MSGPROTO_OSPF,
585 rep->re_daddr,
586 masktomasklen(rep->re_dmask),
587 rep->re_edgertr->ed_ifaddr,
588 rep->re_nhop,
589 rep->re_edgertr->ed_ifindex,
590 rep->re_edgertr->ed_netaddr,
591 rep->re_edgertr->ed_netmask
592 );
593 }
594 break;
595 default:
596 abort();
597 }
598
599 if(n <= 0)
600 fanpdif_wait_connection();
601 return n;
602 }
```