受信状況のフィードバックを用いた IP データグラムの複数経路伝送の設計と評価

M2006MM013 川本 高弘
指導教員　河野 浩之

1　はじめに

現在のデータ通信は単一経路で伝送されるので, 帯域幅が小さい経路にトラフィックが集中し, 安定した帯域の確保が難しくなっている. また, ひとつのトラフィックが経路の可用帯域を使い切ってしまうと, バックグラウンドで流れていたその他のトラフィックを圧迫する.

そこで本研究では, 1 つのデータストリームを複数の経路で伝送することにより, それぞれの経路にかかる負荷を分散し, ネットワーク全体の帯域幅を効率良く利用することができるアーキテクチャを設計する. オーバレイネットワークにおける複数経路伝送の研究には [1] がある. ネットワークの両端にゲートウェイを配置し, 各経路の受信状況をフィードバックすることで, 振り分けに反映させる. それによって, 輻輳を抑制し, 常に安定した帯域幅を確保できるようにする. 振り分けはパケット毎に行い, 各経路に重みを設定することで, 経路の比重に応じた割合でトラフィックを振り分ける. [1] でも同様の研究がなされており, ゲートウェイが経路の輻輳状況をフィードバックすることで効率的な伝送を実現する.

システムは実ネットワークで運用可能なように設計したが, 実験環境の条件が自由に設定できるのでネットワークエミュレータを用いて実験した. フィードバックの更新間隔や更新係数を調整することで, 帯域幅の急激な変化にも対応できた. いくつかの振り分けアルゴリズムを試してクロストラフィックが発生してもスループットを維持できることを確認した. また, ゲートウェイで受信時に順序を補正することで TCP のスループットを向上させることができた.

2　複数経路伝送モデルの概要

本研究で提案するシステムは個人で所有する小規模なネットワークから組織が管理する中規模なネットワークにおいて, 特に遠隔地にある拠点間をインターネット経由で通信する場合に用いることを想定する. ゲートウェイをネットワークの両端に設置し, その間に 2 つ以上の中継ノードを設置することによってオーバレイネットワークを構成し複数経路を実現する. また, 各経路につき 2 つ以上の中継ノードを設置することもできる. ゲートウェイ, 中継ノードは各ネットワークの管理者によって運用されるため, エンドユーザからはゲートウェイや中継ノードの存在は隠蔽され, 一般の利用者は複数経路で通信していることを意識することなくネットワークを利用できる. 実装するさい, 実ネットワーク上においても実現可能となるようにアーキテクチャを設計するが, エミュレータを用いた方が自由にネットワーク構成を設定できるので, 実際にはネットワークエミュレータを用いて実装する. エミュレータには Goto'sIP NetworkEmulator (以下 GINE 略)[2] を用い, ゲートウェイ, 中継ノード間の構成をエミュレータ上で実装する. この節ではまず, 片方向フローだけの単純なモデルで本研究で設計するシステムの中でのパケットの流れとトンネリングの概要を説明し, その後受信状況のフィードバックで用いる情報を説明する.

2.1　片方向フローにおけるトンネリングの概要

ある一組のゲートウェイ間を流れるパケットを考えると**図 1** はその片方向のパケットの流れを示している. ゲートウェイは送信するパケットを振り分ける機能と受信したパケットを順序通りに補正する機能という大きく分けて 2 つの構成から成る. 本来, この 2 つの機能は同時に動作し, ゲートウェイ間での双方向通信を可能にするが, 片側のフローだけに着目した場合, 送信側のゲートウェイでは振り分け機能, 受信側のゲートウェイでは順序補正機能のみが動作することになる. 以後は送信側のゲートウェイを EnterNode, 受信側のゲートウェイを ExitNode, 中継ノードを Relay と呼ぶ.

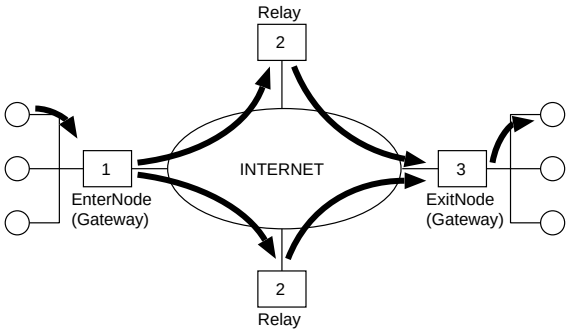

図 1　片方向フローの複数経路伝送

本研究では EnterNode と ExitNode 間を Relay を経由してトンネリングすることによってオーバレイネットワーク上での複数経路を実現する. 以下でトンネリングの概要と片方向フローにおけるパケットの流れを説明する.

1. 送信ホストから送信されたパケットが EnterNode へ到着すると, 経由先となる Relay が振り分けアルゴリズムで決定され, そのパケットにトンネリングのための制御情報を付加したものをペイロードとし, UDP で Relay へ送信する
2. Relay では受信した UDP セグメントのペイロードの中にあるパケットの宛先 IP アドレスを読み取り, そのアドレス宛へ同様に UDP で送信する.
3. ExitNode では受信した UDP セグメントからペ

イロードを取り出し, EnterNode で付加された制御情報を用いてパケット順を補正し, 受信ホストへ送信する

2.2 フィードバックする情報

本研究では経路状況を受動的に計測する. 調査パケットを出さずに計測できる情報として各経路の遅延がある. そこで ExitNode において, パケットが通過する時の各経路の状況を集計し, EnterNode へフィードバックすることで Relay にどの程度の量のパケットを振り分けるかの判断材料として用いる. 受信状況に関する情報として以下の値を用いる.

- 平均遅延時間
- 最大遅延時間
- 最小遅延時間
- 遅延時間の分散
- 各経路の平均ジッタ

上記の値を測定するために EnterNode でパケットにタイムスタンプをつける. それらから ExitNode での受信時間から経路の遅延時間を計算できる. また, 経路毎に各パケットの遅延時間を一定時間集計してくことで上記の値を算出できる. ジッタとはその経路での遅延のゆらぎを示し, 各パケットの遅延時間から算出できる. 平均や分散などのフィードバック情報の集計は一定時間毎に行う. ExitNode は受信したパケットの遅延時間を保存しておき, 集計のさいに保存されたデータはクリアされ, 再度パケットの遅延の記録を開始する.

2.3 提案システムの構成

本研究で提案する複数経路伝送システムはクラス間通信によってパケットの伝送をエミュレートしている. 複数経路を実現している各クラスの間には PacketQueue クラスを置き, そのクラスの enqueue() と dequeue() のメソッドを呼び出すことでクラス間通信を行う.

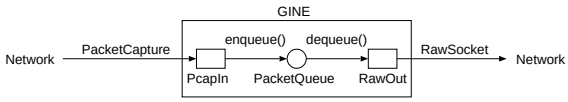

図 2 GIEN でのソケット通信

図 2 は GINE 内部でのパケットのやりとりを示す. enqueue() を呼び出すと PacketQueue クラスにパケットが送信され, dequeue() を呼び出すことで PacketQueue クラス中に蓄積されているパケットが取り出される. クラス間の通信はすべてこれらのメソッドを呼び出すことで行われる. 実ネットワークと GINE とのパケットのやりとりには RawSocket と PacketCapture を用いる. 外部からやって来たパケットは PacketCapture によって PcapIn クラスへキャプチャされ, GINE 内部に取り込まれる. GINE から実ネットワークにパケットを送信する場合は, RawOut クラスにおいて RawSocket でパケットが作成され, 外部ネットワークへ送信される. 以上のようにクラス間での通信は PacketQueue クラスの enqueue, dequeue メソッドによって実現され, 外部ネットワークとネットワークエミュレータ GINE とのやりとりには RawSocket, PacketCapture を利用する. **図 2** では PcapIn と RawOut クラスの間に PacketQueue クラスのひとつしか存在しないが, 実際には EnterNode, ExitNode, Router, Relay などの複数のクラスを配置して複数経路のネットワークを構成する.

3 振り分けアルゴリズム

単一経路でデータを伝送する場合, 経路中のボトルネック回線にトラフィックが集中し, ひとつのトラフィックが可用帯域を使い切ってしまう. そこで本研究では同一のデータを複数の経路で伝送し, 経路にかかる負荷を分散する. 振り分けアルゴリズムの目的はフィードバックされた情報から経路の輻輳を推定し, 輻輳が発生した経路に流すトラフィックを制御することである. トラフィック分配の基準として本研究では遅延の増加と遅延のゆらぎに着目する. 輻輳が発生するとその経路の遅延は増大する傾向にあるので, 遅延の増加を基準にすることで輻輳を推定する. また, 輻輳の発生によって経路が不安定になるため, 遅延のゆらぎを検出することで輻輳を推定する.

3.1 遅延を利用した振り分けアルゴリズム

経路が混んでいるかどうかを推定するさい, 経路の遅延を利用することが考えられる. その経路が空いている状態の時と比べて遅延がどれだけ増加するかを求めることで混雑具合を推定できる. そこで各パケットの遅延を用いたアルゴリズムをいくつか提案した. 以下に詳細を示す.

1. 平均遅延をそのまま重みにする
2. (平均遅延 − 最小遅延) を重みにする
3. (平均遅延 − 最小遅延) $+N\times$ 平均遅延の標準偏差を重みにする

1. は平均遅延をそのまま重みとすることで遅延の小さな経路により多くのパケットを振り分けるアルゴリズムである. 遅延をもとに各経路の重みを求めているので, このアルゴリズムを利用すれば遅延の小さい経路が優先され, 全体としての平均遅延を小さく抑えられると期待できる. このアルゴリズムの問題点は経路固有の伝搬遅延を考慮していないことである. 最大遅延のみを基準として経路を選択するので衛星回線での通信のような遅延は大きいが回線容量も大きな経路が存在する場合, その経路にはあまりパケットが振り分けられない. 各経路の遅延が異なる場合, 2. の方法を取ることで是正できる. 最小遅延が伝搬遅延に近い値であると仮定すると, 平均遅延から最小遅延を引くことで, 遅延の増加分のみを重みとすることができ, 遅延の違いによる振り分け配分の不公平を減らすことができる. 3. は計算から経路の推定最大遅延を求め, 重みにする方法である. ExitNodo に到着するパケット遅延増加の分布が正規分布と仮定すると平均と分散を用いて遅延の最大値を推定できる. 遅延

表 1　ネットワーク構成

経路	回線容量	固定遅延
Path0	10Mbps	100ms
Path1	5Mbps	10ms
Path2	5Mbps	10ms

表 2　Path0 へのクロストラフィック

発生時間	平均間隔	平均ペイロード長
20s - 40s	3ms	1400Bytes
40s - 90s	1ms	1300Bytes
100s - 130s	5ms	1200Bytes

増加の平均 μ, 分散 σ^2 とし, $\mu + 2\sigma(N = 2)$ の値を推定最大遅延とすることで, 90% 以上の出現確率で経路の最大遅延を推定できる. その値を振り分けの基準とし, 各経路への振り分けの確率を求める.

3.2　遅延のゆらぎを利用した振り分けアルゴリズム

実験の結果, 輻輳が発生した経路は遅延が不安定になる傾向があった. 輻輳が起こると送りきれなかったパケットはルータのバッファにたまり, 待ち時間が発生する. バッファでの待ち時間のため, その経路の遅延は不安定になると推測される. 以上のような理由から遅延のゆらぎを推定できれば輻輳が起こっている経路の選択を回避できると考えられる. そこで遅延のばらつきの度合から経路の輻輳状態を推測し, パケットを振り分ける方法を提案する. 遅延のゆらぎを利用したアルゴリズムを以下に示す.

1. 遅延の分散を重みにする
2. 各パケットのジッタの平均を重みにする

本研究では遅延の分散, パケットのジッタという 2 つの尺度から輻輳を推定する. 1. は遅延の分散を重みにし, 遅延のゆらぎの少ない経路を優先的に選択するようにした. 2. の方法は遅延のゆらぎをジッタを用いて評価した. ジッタの算出方法は RFC3550 の RTP: A Transport Protocol for Real-Time Applications にある計算方法を採用した [3]. RFC ではジッタは受信した 2 つのパケットの間隔から計算する. 遅延のゆらぎがあると, パケット間隔にばらつきが生じるため, そのばらつきから輻輳を推定する. そして, 算出されたジッタから各経路の重みを計算する.

平均遅延から輻輳を推定する以外の振り分けアルゴリズムは遅延のゆらぎを基準として経路を選択するため, 遅延の大きさは選択基準に含まない. そのため, それぞれのアルゴリズムの振り分け比率は同様の傾向を示すと考えられる.

4　実験と考察

この複数経路システムに流す通信として FTP によるファイル転送, ストリーミングや電子会議システムなどを想定する. FTP による通信は TCP を, ストリーミングや電子会議システムには UDP を使って模倣する. インターネットでのストリーミングには 2Mbps から 5Mbps くらいが用いられる. 今回の実験ではもう少し余裕をもたせて UDP10Mbps を流す.

実験環境は GINE を用いて**表 1** のように構築した. 実験 1 では平均遅延をもとにした場合と遅延の増加をもとにした場合での振り分けアルゴリズムを用いた. 固定遅延は大きいが回線速度も大きい経路と逆に遅延は小さいが回線容量も小さい経路を用意し, これらの経路を有効に活用できるかアルゴリズム毎に比較する. 遅延差がある複数経路伝送での公平性が振り分け基準に遅延の増加を用いることで改善されるか実験した. 実験 2 では順序補正した場合としない場合に TCP スループットに違いが出るかを調べた. また, クロストラフィックを流すことで, 故意に輻輳を発生し, 輻輳度を推定してアルゴリズムに反映できるか, 可用帯域の急な変化に振り分けアルゴリズムが対応できるかなどを比較した. 振り分けアルゴリズムにはいくつか実験した中でもっとも結果の良かった遅延の増加を用いた場合のアルゴリズムを使う. クロストラッフィックは UDP を用いて**表 2** のように Path0 へ流す. 更新係数 α は 0.03, 更新間隔 T は 3 秒で実験する.

図 3 は平均遅延をもとに輻輳を推定する振り分けアルゴリズムを用いてスループットを測定した結果である. UDP の 10Mbps を複数経路に流した結果, Path1 と Path2 に対して重点的に配分された. たとえ回線容量が大きくてもその経路の遅延が大きい場合にはその経路への比重が少なくなることを示している. これに対して, 遅延の増加をもとに輻輳を推定するアルゴリズムで同様の実験をすると, **図 4** のように遅延の大きい Path0 へ重点的に振り分けられた. 複数経路全体での総合スループットはどのアルゴリズムを用いた場合でもほとんど違いは見られなかったが, 振り分け方法によって経路への分配比率に差が見られた. 平均遅延をもとに輻輳を推定すると固定遅延の大きい経路は不利に扱われるが, 遅延の増加から輻輳を推定すれば固定遅延の違いによる振り分けの不公平は発生しない. 同様に分散やジッタから輻輳推定する振り分けアルゴリズムでも公平性を改善できた.

図 5 **図 6** は遅延の増加をもとに輻輳を推定する振り分けアルゴリズムでの TCP と UDP のスループットの測定結果である. ネットワーク構成と回線容量は実験 1 と同じである. クロストラフィックによる輻輳で遅延が自然に発生するため, 3 経路とも固定遅延は設定しない. UDP は 10Mbps のトラフィックを流した. **図 5** は TCP で順序補正ありと順序補正なしの場合で総合スループットを比較した. **図 6** は UDP で順序補正なしの場合の測定結果である. TCP では順序補正をした場合の方が全

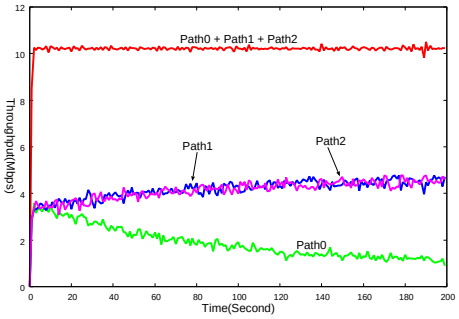

図 3　実験 1　UDP 平均遅延を用いたアルゴリズム

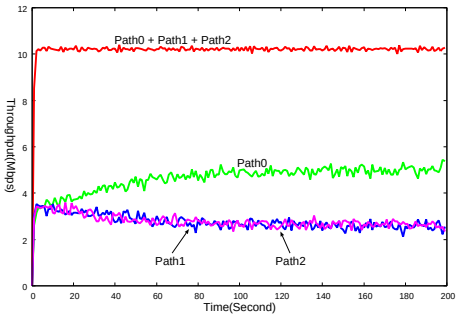

図 4　実験 1　UDP 遅延の増加を用いたアルゴリズム

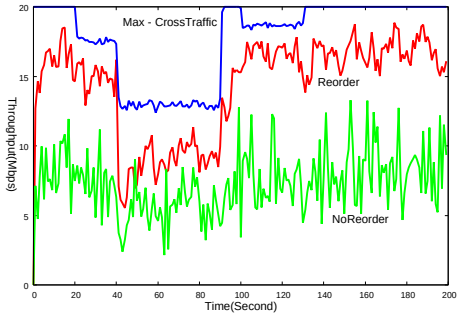

図 5　実験 2　TCP 遅延の増加を用いたアルゴリズム

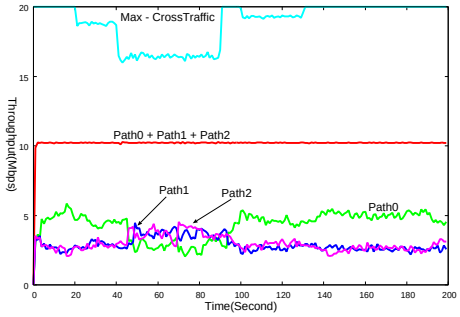

図 6　実験 2　UDP 遅延の増加を用いたアルゴリズム

体的に高いスループットが得られ, 安定している. クロストラフィックが流れて経路の輻輳状況が変化すると, スループットが大きく低下するが, フィードバック情報によって振り分け配分が変更されてすぐにスループットが上昇した. 一方, UDP ではクロストラフィックの発生によって複数経路のスループットは低下せず, 経路の分配比率を切り替えることでクロストラフィック発生前のスループットを維持できた. クロストラフィックが発生しても複数経路上の通信への影響を最小限にすることができる.

5　おわりに

実験 1 では遅延のゆらぎから輻輳を推定することで可用帯域の大きな経路に多くのトラフィックを振り分けることができた. 平均遅延をもとに輻輳を推定するアルゴリズムでは遅延差のある経路で不公平な振り分けが起こったが, 遅延のゆらぎを振り分けの基準にすることで公平性を改善できた. 実験 2 では順序補正によって TCP スループットが大幅に改善できた. また, クロストラフィックが流れ, 輻輳が発生した場合も本研究のシステムは振り分けの比重を変更することでスループットを維持できた. 複数経路を用いることでクロストラフィックの影響を抑えている. また, 影響が出てもフィードバックされた情報を使って分配比率を調整し, クロストラフィック発生前のスループットをできるだけ維持する. これは TCP, UDP どちらの場合も有効であり, これらの機能によって, ファイル転送, ストリーミングなどの TCP 通信や電子会議などの RTP 通信のコンテンツをより安定的にユーザへ提供できる. また, バックグラウンドにおける通信トラフィックへの影響を低減できる.

今後の課題として本研究で提案した複数経路伝送システムを実ネットワークで実現させることが必要である. [4] にある TUN/TAP を使って仮想的なトンネリングデバイスを構成し, そのデバイスを通して通信することで比較的容易に複数経路を構成することができるようになると思われる.

参考文献

[1] Kokku , R., Bohra, A., Ganguly, S., Venkataramani, A.: “A Multipath Background Network Architecture,” IEEE INFOCOM, pp. 1352-1360, May 2007.

[2] Ihara, A., Murase, S. and Goto, K.: “IPv4/v6 Network Emulator using Divert Socket,” Proc. of 18th International Conference on Systems Engineering(ICSE2006), Coventry, UK, pp. 159-166(Sep. 2006).

[3] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.,: “RFC 3550 RTP: A Transport Protocol for Real-Time Applications,” RFC 3550, July 2003.

[4] Krasnyansky, M., Yevmenkin, M.: “Universal driver TUN TAP,” http://vtun.sourceforge.net/tun/ (accessed 2008).