通信経路におけるボトルネック回線容量の推定

2004MT014 夫馬　修平　　2004MT079 大崎　将也
指導教員　後藤　邦夫

1　はじめに

現在，動画や音声などのコンテンツの需要が盛んになり，回線が混雑しサービスを安定して提供，使用できないおそれがでてきた．そこで，事前にホスト間で利用可能な帯域幅や遅延，経路上でもっとも狭い回線の容量などの回線状況を把握する必要がでてきた．今研究では PathTester [5] [6] を用いて，それらの値を計測する．

PathTester とは既存の帯域幅計測ツール [1] [4] で用いられているワンパケット，パケットトレイン，パケットペアの 3 つの手法を応用し，片道遅延，可用帯域，ボトルネック回線容量の推定を目的とするツールである．昨年度卒論では計測パスの両端に専用アプリケーションを配置し、NTP サーバで時刻補正した結果，片道遅延，可用帯域計測に成功した．

しかし，このツールには下記の 2 つの問題点がある．

1. ボトルネック回線容量の計測結果の精度が安定しない．
2. Windows で実行できないため，実験環境が限られてしまう．

第 1 の問題は，パケットペアのデータ長に比例する到着時間の差の増加に着目し，安定した推定値を得られる新たなデータ処理方法の提案をする．

第 2 の問題は，クライアント側のプログラムを Windows と Linux のどちらでも実行できる Java で再現し，解決する．

夫馬修平は主にモデルの作成を，大崎将也は主にプログラムの作成を担当した．

2　新 PathTester のアプリケーションの概要

新 PathTester で使う新たなアプリケーションの概要を述べる．

2.1　新 PathTester の起動時の環境

新 PathTester の起動時の環境は，サーバ側の PC で常にプログラムが起動している状態になっており，クライアント側からポート指定をしてリクエストすることによって NAPT を通過し，実ネットワークを介して一般家庭からでも片道遅延，可用帯域，ボトルネック回線容量の推定を行うことができるようした．新 PathTester の起動時のプログラムの流れは，図 1 のようになっている．

まず，待機中のサーバ側 HostS にクライアント側 HostR が PathTester の使用リクエストをだし，それに対して返事を送信する．次に，各々のホストの最寄りの NTP サーバに対して時刻同期をし，片道遅延，可用帯域，ボトルネック回線容量の推定を行う．時刻同期とは各ホストのローカルクロックとの差を保存し，ローカルクロックで記録したサンプル中の時刻を補正することである．さらに正確さを高めるためここでまた NTP サーバと時刻同期する．HostS で統計処理をして結果を HostR に送信し，受信した HostR は結果を出力し実行を終了する．結果を出力した後，HostS の PC は待機状態に戻るようにしてある．なお，NTP サーバとの通信の遅延を少なくするため，最寄りの NTP サーバを使うことにする．

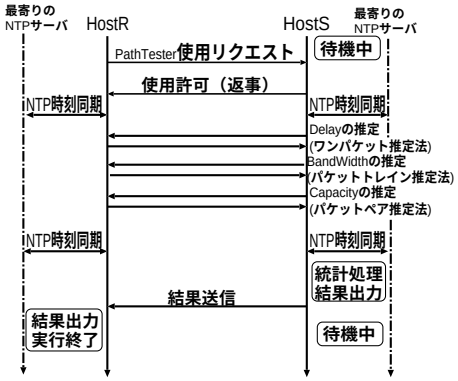

図 1　新 PathTester の処理の流れ

2.2　ボトルネック回線容量の推定方法

ボトルネックとは，コンピュータの処理速度やネットワークの通信速度の向上を阻むもので経路上で一番遅い（狭い）回線の容量である．

図 2 に示すように，両端の HostS と HostR の間にいくつかの速い回線があり，途中 Router1 から Router2 への回線がボトルネックとなる最も遅い回線であるとする．両端のホストの NIC 速度が計測対象パスのボトルネック回線容量より小さければ，ボトルネックはホストの NIC になるので，NIC より速いボトルネックは計測

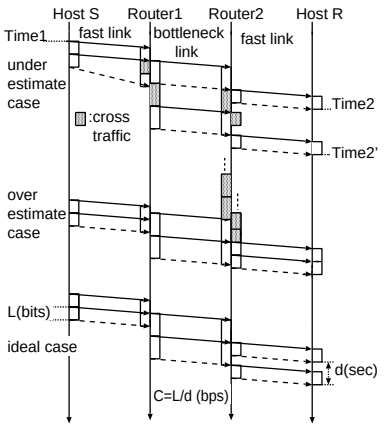

図 2　ボトルネック回線容量の推定

できないので，NIC の速度は bottleneck link より速いとする．既存 PathTester でも，パケットペアの推定法が用いられており，CapProbe[4] では探査パケットをパケットペアで送信し，受信の際に遅れが最小のサンプルが得られるまで探査パケットの送信を繰り返す．また，往復で計測する必要があるために，サンプルが短時間で得られず，計測に長時間かかる問題がある．新 PathTester では，計測する時刻は HostS におけるパケット送信開始時刻と，HostR におけるパケット受信完了時刻である．パケットペアの先頭パケットの送信時刻，その受信完了時刻，2 つめのパケットの受信完了時刻をそれぞれ，$Time_1$，$Time_2$，$Time'_2$ とする．クロストラフィックが全くなく，全てのルータで待ち行列遅延がない場合は，図 2 の最後の例のように，パケットサイズ，L(bits)，に対して到着時のパケットペアの間隔，$d = Time'_2 - Time_2$(sec) はボトルネック回線容量，C，で決定され，式 1 が成立する．

$$C = \frac{L}{d} \tag{1}$$

パケットペアの間にクロストラフィックが割り込んだ場合は，最初の例のように，d が大きくなり，$\frac{L}{d}$ はボトルネック回線容量として過小評価になる．ペア前にクロストラフィックが割り込んだ場合は，2 つめの例のように d が大きくなり，$\frac{L}{d}$ は過大評価となる．理論的には，L が小さいほどパケット割り込み確率が小さいので，L が小さいパケットペアの計測に適当時間かければ，パケット割り込みも，ボトルネック後の高速リンクの待ち行列もないサンプルが得られる．しかし，実際は L が小さいパケットを PC から送信すると少し間隔が空いてしまう．

2.3 改良ボトルネック回線容量の推定方法

今回の改良したボトルネック回線容量の推定手法の方法では，同じ長さのパケットペアでその長さをかえて送信する．例えば，100+100，...，1400+1400 オクテットのように送信し，そのサンプルを統計処理することにした．算出方法は，C=回帰直線の傾きの逆数× 8 × 10^{-6}(Mbps) で求められる．クロストラフィックがない状況では，図 3 のようなパケットが増えるたびに増加傾向のあるという結果がみられ，回帰直線とサンプルとのズレがないので計測は成功していると考えられる．このように増加傾向になる理由としては，オーバーヘッド時間，パケット長に比例する送信時間，バッファでの待ち時間，他のトラフィックの割り込みなどであると考えられる．そして，バッファの待ち時間とトラフィックの割り込みは 0 が理想である．なお，探査パケットが分割送信されるとすべての計測が不可能となるので，PPPoE が用いられるのを想定して，探査パケットの最大長は 1450 オクテットとしている．

これより，改良した推定方法について述べる．推定結果の精度の向上ために，まずパケットペアをパケットサイズごとに複数回送信することにした．多くのサンプルを得ることができるが，外れ値を含んでおり，バラつきが生じるためサンプルの選別も必要になる．そこで，新たな計測手法として考えたのは，クラスタ分析を用いて，

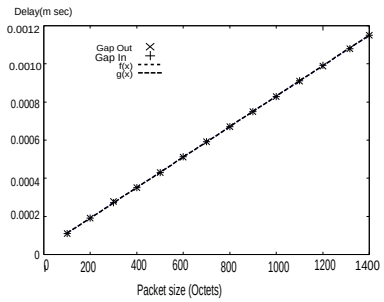

図 3　実行例:ボトルネック回線容量の推定

外れ値を除くという方法である．プロット図を目視で確かめながら実験を繰り返した結果，以下の 2 つのルールで外れ値とし，除いていくことにする．

1. パケットペアの到着時刻がそれぞれ遅いもの．
2. 計測結果を過大評価してしまうパケットペアの到着時刻の差の値．

1 つめに関しては，図 2 に示したように，パケットペア間にクロストラフィックが混入してしまい，図 4 のような外れ値となる場合が多かったためである．そこで，パケットペアの到着時間が早いものが欲しいので，それぞれの到着時間を 2 乗し，その和を求め，その値でサイズ長ごとクラスタリングし，平均値が一番小さいグループを採用することで解決した．

次に 2 つめであるが，1 つめのルールの外れ値を除いたとしても，過大評価してしまう値が残ってしまうことが分かっている．過大評価してしまう値は，図 2 の over estimate case のように到着時刻の差の値が極端に小さく，この値は図 4 の下方にある傾きが小さい回帰直線を引こうとしているものである．これらの値は，到着時刻の差の値を全体でクラスタリングして，平均値が小さいものを除くことで解決した．しかし，このクラスタ分析

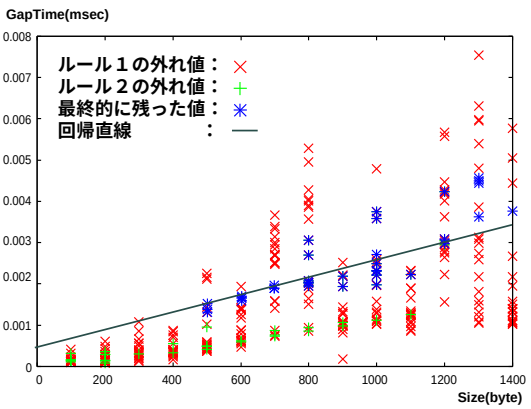

図 4　外れ値を含むプロット図

だけではまだ外れ値がある状態となっている．これを解決するために，ロバスト回帰分析も用いた．ロバスト回帰分析の回帰直線を引くことによって，さらに外れ値を

除いた回帰直線が引けるということになる．なお，実験結果については 4 節で示す．

3 PathTester のシステムの実現

この節では，システムを実現するために必要なパケットデータ形式やプログラミングの工夫などを述べる．

3.1 探査パケットの形式

システムを実現するにあたって、サーバ側とクライアント側の間で送受信される探査パケットについて説明する．PathTester は，計測したいネットワークのそれぞれで起動する sender プログラム,receiver プログラムから構成される．サーバ側のプログラムでは推定計算を含む処理をし，クライアント側のプログラムでは単に，送信されてきた探査パケットに対し て応答している．ペイロードは図 5 のような独自の形式である．

version/control	res1	key	
interval			
sequence		length	
phase	subseq	tsq	tsize
sendttl1	recvttl1	sendttl2	recvttl2
time1(sec,usec)(8 octets)			
time2(sec,usec)(8 octets)			
time3(sec,usec)(8 octets)			
time4(sec,usec)(8 octets)			
ntpoffset(sec,usec)(8 octets)			
ntpdelay(sec,usec)(8 octets)			
ntpserver(48 octet char string)			
reserved (48 octets)			

図 5　探査データの形式

3.2 2 つのプログラム全体構造

sender.cc と receiver.cc の構造は，図 1 の流れでパケットの送受信を繰り返す．特徴としては図 5 にあるように，パケットの順番を sequence，subseq，tsq に書き込むというところ，パケットを受信した際と送信する際にサーバとクライアントで時間を記録するため，time1，time2，time3，time4，の 4 箇所書き込むというところである．パケットの送受信が全て終わったら，データファイルを生成し，パケットペアのサイズと 1 つ目と 2 つ目の時間やそれらの差，それらの 2 乗の和を書き込む．そして，統計処理ソフト R で，先に作ったデータファイルからクラスタリングして値を除去し，さらに新しいデータファイルを作る．その新しいデータファイルより R で計算し，ボトルネック回線容量値を求める．最後に，サーバ側で結果を出力し，クライアント側に結果を送信して実行を終わる．

3.3 Windows で使える Java 版クライアントの工夫

C++ でプログラミングされている PathTester を，Java で再現するために以下の問題点がある．

1. Linux のシステムコール関数の gettimeofday() 相当のものが Java には無い．
2. C 言語の構造体のような UDP datagram のデータをパケットで定義する．
3. Hops 数を示すための TTL の取得方法．

1 つ目の問題であるが，絶対時間ミリ秒を取得する System.currentTimeMillis() とシステム時間のナノ秒を取得する System.nanoTime() の 2 つを組み合わせることによって解決した．

2 つ目の問題については，int[] 型を定義して 16 進数で代入する．なお，8 ビットを超えるものはシフト演算を使い，いくつかに分けて代入する．送信時にはこの int[] 型を byte 型にキャストし，byte[全体のデータ長] 型に代入してから送信する．受信した datagram パケットは byte[] 型から int[] 型に戻して扱うという処理をして，この問題を解決した．

TTL の取得問題に関しては，C 言語で関数を呼び出す JNI を使用すれば可能であるが，TTL は実行に影響しないうえ，本研究では Windows での実行を目的としているため，これを省略した．

3.4 GUI の導入

GUI の実現に java.awt を利用した．そして，JSmooth[3] を使って Windows のアイコン付き実行ファイルにした．GUI を実行すると，アイコンをクリックするだけで実行でき，コマンドプロンプトを開く必要がなく扱いやすくなった．また PathTester のツールをインストールすれば，そのままツールを使えるようになっている．もし PC に jdk が入ってない場合でも，jdk のインストール手順の説明が出る．これで，Windows でも Linux と同等の実行が可能になり，さらに GUI 機能を追加することでエンドユーザにも扱え，より多くの環境で実行できるようになった．

4 新 PathTester の実験結果

本節では，新 PathTester が旧 PathTester より正確に推定できるかを確認する実験について述べる．

4.1 エミュレータ GINE での実験

PathTester の実験を User Mode Linux を利用し，エミュレータ GINE [2] で実験した．図 6 で示めすような，実験環境モデルを作成した．ルーター 4 つに対して，In 方向のボトルネックを 10Mbps にしてクロストラフィックを徐々に投入し，どれほどの影響がでるかを計測した．In 方向を計測した実験結果は表 1 に示す．表 1 は NewIn を 2 節で示した新しい推定法を使った値，OldIn は旧 PathTester での値である．Out 方向はクロストラフィックは影響していないので省略する．

表 1 から，クロストラフィックがない場合では，OldIn でも 10Mbps に近い値が得られる．しかし，5Mbps，7Mbps のクラストラフィックを投入した場合は，OldIn では 10Mbps に近い値は得られなくなっているが，NewIn では 10Mbps に近い推定結果が得られている．つまり，新たな計測手法はクロストラフィックによる影響を軽減できている．よって，新 PathTester の計測手

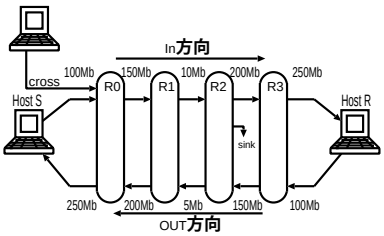

図 6　エミュレーションモデル

表 1　エミュレータでの実験結果

Cross(Mbps)	CP(Mbps)		Hops
	NewIn	OldIn	O/I
0.0	9.999	9.784	4/4
5.0	10.221	7.434	4/4
7.0	10.457	6.349	4/4

法の精度が，旧 PathTester よりも向上していることが分かる．

4.2　実ネットワーク上での実験

次に実際のネットワークで実験した．実験環境は研究生夫馬宅と研究室間，研究生大崎宅と研究室間の 2 通りで実行した．研究生宅のネットワーク環境は，夫馬宅：下り 30Mbps ケーブルテレビ回線，大崎宅：下り 10Mbps ケーブルテレビ回線である．上りの数字は不明であるので今回は省略する．また，実ネットワーク上で任意のクロストラフィックを投入するのは難しいために，ネットトラフィックが一番混んでいる時間帯が 21 時から 23 時，一番すいている早朝の 2 つの時間帯に実験をすることにした．結果は表 2 に示す．NewOut は新

表 2　自宅と研究室間での実験結果

Place	Time	CP(Mbps)		Hops
		NewOut	OldOut	O/I
夫馬	23:00	10.696	263.771	12/12
宅	5:00	24.390	224.892	12/12
大崎	23:00	9.754	22.812	10/10
宅	5:00	10.342	7.655	10/10

PathTester での結果，OldOut は旧 PathTester での結果となっている．

大崎宅と研究室間の実験結果は，新 PathTester の実行結果の値が 10Mbps に近似しており，新 PathTester がきちんとデータの外れ値をはずし，ボトルネック回線容量を推定していることが分かる．また，夫馬宅と研究室間では，以前までは明らかな過大評価の結果で安定してしまっていたが，新たな手法では 30Mbps 付近の値を計測することができた．またトラフィックが混んでいる時間帯，そうでない時間帯ともに良い結果が得られることから，実ネットワークでも，ボトルネック回線容量に近似した値を計測できると考えられる．

実験結果をまとめると，エミュレータの実験から，新 PathTester は旧 PasthTester よりも計測の精度は向上していると考えられ, 実ネットワーク環境では, 新 PathTester は過小評価, 過大評価になるデータを除去し, 必要なデータだけを採用することができればボトルネック回線容量を計測できることが分かった．

5　おわりに

本研究は，ボトルネック回線容量のクライアント側のプログラムを書き直し，GUI 機能も追加することで Windows でも利用可能になり，エンドユーザでも容易に使用できるようにした．また，ボトルネック回線容量の推定手法を 2.3 節で述べた多くの手法を用い，GINE や実ネットワーク上で実験を繰り返すことで，PathTester の推定精度は向上できた．だが，下記のような解決すべき点が残されている．

- エミュレータであっても，実ネットワーク上のような実験結果を得られる環境モデルの作成．
- 実ネットワーク上のいろいろな経路で実験してデータを集め，経路におけるボトルネック回線容量の特徴を調べる．
- 取得できたボトルネック回線容量値が本当に正しいかの判断ができるための基準を提案する．
- パケットサイズの範囲縮小，パケット数の削減をしても，期待する実験結果が得られるか調べる．

謝辞

統計関連の質問に丁寧に答えてくださった松田 准教授に深く感謝いたします．

参考文献

[1] Downey, B. A.: *Clink* (accessed August 2007). (http://allendowney.com/research/clink/).

[2] Ihara, A., Murase, S. and Goto, K.: IPv4/v6 Network Emulator using Divert Socket., *Proc. of 11th International Conference on Systems Engineering (ICSE2006).*, Coventry, UK, pp. 159–166 (Sep. 2006).

[3] JSmooth: *Java Executable Wrapper* (accessed August 2007). (http://jsmooth.sourceforge.net/).

[4] Kapoor R., Chen L., Lao L., Gerla M., and Sanadidi. Y. M.: *CapProbe: a simple and accurate capacity estimation technique.* SIGCOMM 2004:67-78(2004). (http://doi.acm.org/10.1145/1015467.1015476/).

[5] 吉田秀考：インターネットにおける可用帯域幅の推定と伝送遅延の評価，修士論文，南山大学数理情報学部情報通信学科 (Apr. 2005).

[6] 片岡哲哉，森真一郎：片道遅延、可用帯域とボトルネック回線容量推定ツールの改良とその評価，卒業論文，南山大学数理情報学部情報通信学科 (Apr. 2006).