修　　士　　論　　文

大規模ネットワークにおける効率的なバンド幅マップ構築アルゴリズム

An Efficient Algorithm for Building Bandwidth Map of Large-Scale Networks

指導教員　**田浦 健次朗 准教授**

東京大学情報理工学系研究科
電子情報学専攻

氏　名　**48-086423 長沼 翔**

提　出　日　**平成 22 年 2 月 9 日**

概 要

本論文では大規模なネットワークにも適応可能なバンド幅マップ構築アルゴリズムを提案する．バンド幅マップとはネットワークトポロジーの全てのエッジにそのバンド幅の値を振った情報である．この情報は，集合通信や局所性を考慮した負荷分散などの，並列分散アプリケーションで一般的に行われる処理を行う際に有用である．

ところで，ある環境におけるバンド幅の別の表現法としてよく用いられるものにバンド幅行列がある．バンド幅行列は，単にシステム内の全ホスト対全ホスト間の End-to-End バンド幅を表に列挙したものであり，容易に取得できる．しかしこれだけの情報では実際に通信を行う時の通信同士の衝突や中間のボトルネックリンクを知ることができない．

本手法では，複数のホストペアを協調させて複数のネットワークストリームを流すことにより，一対一のバンド幅測定では知ることのできなかった中間ボトルネックリンクを測定可能にする．更に我々の手法では，互いに通信の衝突が起こらないような領域にネットワークトポロジーを分割して考え，多数のエッジのバンド幅測定を並列に実行できるようにスケジューリングを行う．本研究の手法は WAN をまたぐ 15 クラスタ 311 ホスト，640 エッジの環境のバンド幅マップを 50 秒程度で構築した．構築したバンド幅マップの中には中間ボトルネックリンク以外の値も確かめられ，またホスト数の増加に対するスケーラビリティも実現された．

目 次

図 目 次

アルゴリズム，擬似コード

第1章　序論

1.1　背景

昨今のネットワーク性能の向上や計算機の価格の下落により，多数の計算機をネットワーク接続して計算処理を分散させて並列に行う並列分散処理が盛んに行われるようになっている．並列分散処理により安価で莫大な計算資源を得ることができ，これまでできなかった計算が可能になったことで，様々な分野からその有用性が期待されている．中でも，大容量データに対する大規模な計算を必要とする言語処理や画像処理などの分野では，並列分散処理は必須の技術となっている．また，計算資源の利用という以外にも，負荷分散やデータの冗長化など，分散システムはコンピュータシステムの根幹を支える重要な技術になりつつある．

並列分散処理の流行により，そのシステムのネットワークリンクのバンド幅の情報を何らかの手法で知るということが，システム管理者だけでなくユーザにとっても重要なこととなった．上に挙げたような大規模データを扱う処理をデータインテンシブなアプリケーションといい，このようなアプリケーションは大容量のデータ通信を計算機間で行う場合が多い．この際，何の意識もせずに通信することが全体のパフォーマンスを大きく低下させてしまうことにつながる．なぜならば，大規模な分散環境では一般的にネットワーク構成が不均質であり，複雑な接続形態や大小のバンド幅が混在することが普通であるからである．このような環境上で計算機間のデータ通信を無意識に行っていては，通信同士の衝突によるバンド幅の低下やコンテンションの発生，またはバンド幅の小さい経路を頻繁に使用してしまい処理を滞らせるなど，処理自体にかかる時間に対するデータ転送にかかる時間が大きくなってしまう．よってユーザはシステムのネットワークの接続形態やバンド幅を把握し、これらによる遅延が発生しないようにデータ転送の手順を考えて通信を行う必要性が出てきた．ここで挙げたようなデータインテンシブアプリケーション以外にも，局所性を考慮した負荷分散や集合通信などの，並列分散計算における最も基本的な操作であっても，バンド幅を考慮して最適化すべきだという研究も多くなされている [28][13][20].

しかし，ネットワーク構成が不均質な環境上において，個々のユーザが既存のバンド幅測定プログラムで特定のリンクのバンド幅の値を知ることは容易ではない．それは，ネットワークには通常複数のスイッチやルーターが存在し，それらは通信の末端になることができないためである．通信の末端とは，それにログインしているユーザがセンダやレシーバなどの通信プログラムを走らせることができ，パケットを送出したり受け止めたりすることができるエンドホストのようなノードの

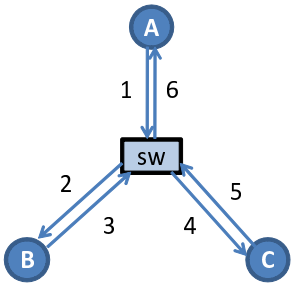
ことをいう．現在広く用いられているバンド幅測定手法は，二つのエンドホスト間でデータ転送を行い，かかった時間で割るというものであるが，二つのエンドホスト間には一つ以上のスイッチ，すなわち二つ以上のエッジが存在することがほとんどである．既存手法のような End-to-End の手法では，複数の測定すべきエッジがその End-to-End 間にあるにもかかわらず，結果は一つしか得ることはできない．さらに，その一つの結果とは経路中の複数のエッジのうちバンド幅が最小のものの値に律速されて出ている結果であるため，どのエッジがそのバンド幅を持っているかは不明である．エンドホストに直接繋がっていないエッジ，つまりそのエッジの両端がスイッチやルータといったネットワーク機器であるエッジのことを，中間エッジという．通常、中間エッジの両端のネットワーク機器にユーザが自由にログインできるということはなく，よって既存の End-to-End のバンド幅測定手法では中間の特定のエッジのバンド幅を測ることはできない．

さらに，システムの規模が大きい場合，測定すべき箇所が多いということも問題である．仮に，平均 m 分岐のスイッチで構成される深さ d のツリーのネットワークを考えた時，そのシステムに存在するエッジの本数は $O(m^d)$ となる．例えば，全国の大学に設置したクラスタを広帯域バックボーンで相互接続した広域分散環境である *InTrigger*[1] では，少なくとも 600 以上のエッジが存在する．この本数を，現在広く使われているバンド幅測定プログラム *Iperf*[22] で仮に測定できたとしても，全て測り終えるのに 100 分近くの時間がかかり，その作業の手間も膨大であることは容易に想像できる．従って，システムの大規模化に伴ってエッジの数も急速に増加していくという問題にも何らかの対策を打たなければ，実用的なバンド幅測定手法と言うことはできない．

1.2 目的

我々の研究は，1.1 節の後半に挙げた二つの要求を主に満足させるバンド幅測定手法を実現することが目的である．最終的な実装では，ネットワークトポロジー情報を入力とし，そのバンド幅マップを出力するものを目指す．本手法で得られたバンド幅マップは，通信を行うアプリケーションの性能を最適化する為の参照情報のような位置づけで利用されることを想定している．ひいては，複雑なネットワーク上で通信を行う並列分散計算や分散アルゴリズムといった，一つ上のレイヤのあらゆるアプリケーションの最適化の為の下地となることが，本研究を含むバンド幅測定の研究全体が目標とし，貢献を目指すものである．さて，ここでバンド幅マップがどのようなものでなぜ必要かということを，分散環境におけるバンド幅測定では出力の形式が主に二つに分類されることを交えて，以下で説明する．

分散環境上で複数測定したバンド幅の表現方法には，バンド幅マップとバンド幅行列の二つがある．バンド幅マップとはネットワークトポロジーの全てのエッジにそのバンド幅の値を振った情報である．より一般的に言うと，測定対象としているネットワークのグラフの全てのエッジにバンド幅の値を重みとして付加した，重み付き有向グラフである．一方バンド幅行列とは，システム中の全エンドホスト対全エンドホストの組で End-to-End のバンド幅測定を実行し，その測定結果を表にま



Bandwidth map (Mbps)

Snd ＼ Rcv	A	B	C
A	-	1	1
B	3	-	3
C	5	2	-

Bandwidth matrix (Mbps)

図 1.1: 簡単なバンド幅マップとバンド幅行列の例

とめたものである．言い換えると，システム中のエンドホストの数を N としたとき，バンド幅行列は $N \times N$ の行列となり，行列の (i, j) 成分とはそれぞれに対応するエンドホストがセンダ，レシーバとなったときのバンド幅測定の値となる．例として，三つのエンドホストが一つのスイッチでスター型に接続された最も単純なネットワークを，バンド幅マップとバンド幅行列の両方で表現したものを図 1.1 に示す．図中の数字はバンド幅の値を表している．

バンド幅マップとバンド幅行列を比べると，前者の方が表現できる情報量は多く，取得しておく価値は高い．図 1.1 からも明らかに，同じ測定対象のネットワークでもバンド幅マップとバンド幅行列とでは，前者の方が情報量が多いことが見て取れる．例えば図 1.1 中のバンド幅行列では，スイッチからエンドホスト A に向かうエッジのバンド幅 6 Mbps と，スイッチからエンドホスト C に向かうエッジのバンド幅 4 Mbps が見えていないが，バンド幅マップではそれらの値を表現することができている．更に別の例として，ホスト B, C の二つからホスト A へ同時にネットワークストリームを流すような場合を考えると，バンド幅マップから読み取るにスイッチからホスト A へ向かうエッジのバンド幅が 6 Mbps であるために，二つのストリームの合計バンド幅は 6 Mbps に律速されることが予め分かる．このようなことはバンド幅行列からでは判断することができず，並列分散計算を行うユーザが誤ってネットワークに適さないデータ転送手順を踏ませてしまいパフォーマンスの低下が起きてしまう可能性も生じる．よってバンド幅行列ではなくバンド幅マップを得ることができなければ，分散アプリケーションのより詳細な最適化を実現することができず，故に我々の研究の目標を達成することはできない．

しかし，バンド幅マップを得ることはバンド幅行列を得ることよりも一般的に難しい．主な原因は，1.1 節でも述べたように，バンド幅の値を特定することが難しい中間エッジが存在するためである．このためか現在広く使用されている表現方法はバンド幅行列である．ある環境のバンド行列を構築したい場合には，単に $N \times (N-1)$ 回の一対一バンド幅測定を行えばよい．

そこで本研究では中間エッジを含む全てのエッジを測定でき，高速にバンド幅マップを構築する手法を提案する．本手法では，多ノード協調参加のバンド幅測定を行うことによって中間エッジの測

定を行い，さらにネットワークトポロジーをいくつかの部分グラフに分割して考え，並列に測定を進めることによって高速化を実現している．

1.3 本論文の構成

2 章 関連手法 バンド幅測定に関する既存の手法・研究を紹介し，解決すべき問題を明らかにする．

3 章 提案手法 本研究のアルゴリズムを述べる．

4 章 評価 本手法を実装し，その実験と評価を行う．

5 章 結論 本論文のまとめと今後の課題について述べる．

第2章　関連手法

2.1　End-to-End のバンド幅測定手法

Iperf[22] は現在最もよく使われているバンド幅測定プログラムの一つである．Iperf は二つのエンドホストでセンダとレシーバに分かれて起動され，片方からもう片方へ大きいデータを送りつけ，送受信できたバイト数をかかった時間で割るという原理でバンド幅を求める．原理は単純であるが，このようにして算出した値をバンド幅とする方法は他の多くのバンド幅測定手法で採用されており，本手法のバンド幅を算出する基本的な部分もこの原理を使用している．

しかし，この原理では中間エッジのバンド幅については知ることができないということを 1.1 節でも述べた．Iperf を含む多くの手法も同様で，センダから送出されたネットワーク流はレシーバに到達するまでに経由するリンクのうち，最小のバンド幅に転送速度が律速される．プログラムとしてはその値しか報告することができず，結局，ホスト間のボトルネックリンクの値しか知ることができず，またそのボトルネックリンクがホスト間のどのリンクによるものなのかも判断することができない．例えば，図 1.1 で挙げたようなネットワークトポロジーでは，任意の 2 つのエンドホスト間には中間エッジが 2 つ存在する．このような最も単純なネットワークトポロジーでさえもこれらのバンド幅測定手法では我々の求めたい正しい答えを得ることができず，よってこれらの手法では本研究の目標を達成することはできない．

Iperf と類似した測定手法に *Netperf*[2] や *NetPIPE*[19] がある．NetPIPE では，socket による単純な send/recv バンド幅だけでなく，ssh 通信トンネルや MPI 通信などの様々なアプリケーション，様々なプロトコルのスループットの測定を提供している [23][24].

Pathrate[8] や *Nettimer*[14] は，センダとレシーバ間のネットワークストリームを一つ一つのパケットレベルのミクロな視点でそれらの挙動を細かく解析し，バンド幅を求める手法である．この種の研究ではスイッチやルータのキュー待ち時間やフォワーディング処理遅延などのネットワークパラメータをモデル化し，2 ホスト間で多数やりとりしたパケットの到着時間間隔の変動から，バンド幅を算出するという原理である．このような観点からバンド幅を求める基本原理は Jacobson によって初めて導入された [12]．以下にその内容を簡単に説明する．Jacobson は，送信元のホストから n ホップ目のルータにサイズ s のパケットが到着するまでにかかる時間 $T_n(s)$ を，式（2.1）の様に定式化した．Q_k，d_k，b_k はそれぞれ k ホップ目のルータのキュー待ち時間，機器固有の遅れ，次のルータへ向かうリンクのバンド幅とする．ここで d_k，b_k は各 k について固有の定数であり，Q_k は変動する

値である．

$$T_n(s) = \sum_{k=0}^{n}(Q_k + d_k + \frac{s}{b_k}) \tag{2.1}$$

もしあるルータ k でキュー待ちの時間無くパケットが通り抜けた場合，Q_k の値は 0 となる．n ホップ先までのルータ全てにおいて待ち時間無くパケットが通り抜けた場合，全てにおいて $Q_k = 0$ とおくことができ，このとき $T_n(s)$ は最小値をとる．すなわち

$$min\{T_n(s)\} = \sum_{k=0}^{n}(d_k + \frac{s}{b_k}) \tag{2.2}$$

となる．従って次の式が導かれる．

$$min\{T_n(s)\} - min\{T_{n-1}(s)\} = d_n + \frac{s}{b_n} \tag{2.3}$$

この式により，なんらかの方法で $min\{T_n(s)\}$，$min\{T_{n-1}(s)\}$ を求めることができれば，b_n を算出することができる．以上が Jacobson の提案した手法である．

Jacobson ら自身が実装した *Pathchar*[12] ではパケットの TTL を 1〜n まで変化させ，返ってきた ICMP Time Exceeded Message の到着時間を計測する．各ルータに到着する時間 $T_n(s)$ を，返ってくる ICMP メッセージの到着時間 $\tilde{T}_n(s)$ に置き換えても式（2.3）は成り立つ．Pathchar では n，s を固定して何度もパケットを投げ，返ってきた ICMP メッセージのうち最短時間であったものにおいて $Q_k = 0$，すなわち $min\{\tilde{T}_n(s)\}$ とみなして計算を行う．そして各 n について様々な s で $min\{\tilde{T}_n(s)\}$ を求め，s と式（2.3）の関係をプロットしたときの直線の傾きの逆数をバンド幅として算出することができる．

ところが，この原理は一つ一つのパケットの到着間隔という非常に短い時間の計測をプログラムで行わなければならないので，実環境では正しいバンド幅の値を出しにくいという問題点がある．例えば 1Gbps のリンクのバンド幅をこの計算で求めることを考える．1Gbps のリンクは，現在最も普及し一般のクラスタの LAN などにも通常で備わっているものである．式（2.1）の $\frac{s}{b_k}$ の項を支配的にすれば結果の分散は相対的に小さくなるので，一つのパケットに許される最大サイズである 65535 バイトを s として考える．すると $\frac{s}{b_k}$ はおよそ 500μ 秒と観測されることになる．ところで 1Gbps の LAN で構成されるクラスタ内通信の遅延は一般的に 1500μ 秒程度はあり，また $\pm 200\mu$ 秒程度のぶれがある．よってもし $min\{T_n(s)\}$ の取得を誤ってしまえば，$\frac{s}{b_k}$ の値は他の項の誤差が無視できないほどの大きさになるので，そこから算出した結果は大きく外れてしまう．さらにこのネットワークモデルで考えている遅延以外にも，エンドホストやルータにおける突発的な CPU 負荷の変動やクロストラフィックによるキュー待ち遅延の増加などで，ミリ秒単位のノイズが入り込んでしまう可能性は十分にある．今後，ネットワーク性能の向上により 10Gbps 回線が普及すれば，さらに $\frac{s}{b_k}$ の項は他のノイズに埋もれてしまうことになり，正しいバンド幅を計算することができなくなってしまう．特に WAN をまたぐ通信であればその遅延の誤差は一般的に大きくなり，WAN をまたぐような大

規模な分散環境を想定している我々の研究ではこれらの手法は適さない．他にも ICMP メッセージを返さない機器があると測定ができないという問題や，TTL を操作するために root 権限が必要であるという問題もあって，これらの手法では我々の目指す貢献を実現することはできない．

ここに挙げた他にも [12] のアイデアに基づいたバンド幅測定は様々研究されている [15][27][7].

2.2 バンド幅行列構築手法

Network Weather Service [26] [25] は，グリッド環境などで広く用いられている分散システム向けの統合ネットワーク監視システムである．以降これを NWS と記述する．NWS はネットワークの接続性の監視や各エンドホスト間での遅延時間の測定など，ネットワークに関連する様々なモニタリングサービスを提供しているが，ここではその中のバンド幅測定に関連するサービスモジュールを紹介する．NWS がバンド幅行列を取得する原理は以下の様である．まずシステム内の全てのエンドホストに NWS デーモンを起動する．次に参加しているエンドホストの全ての組み合わせ $N \times (N-1)$ ペアについて Iperf のような End-to-End バンド幅測定を逐一実行し，システムのバンド幅行列を構築する．ここで測定を逐一実行しなければならない理由は，NWS はネットワークトポロジーを考慮しないでバンド幅測定を進めているためである．測定するために流すネットワークストリームは，衝突するという性質を持つ．Iperf を原理とした End-to-End バンド幅測定では，一つのエッジに同時に二つ以上のネットワークストリームが流されると，それらの通信が競合し両測定の結果が誤ってしまう．NWS ではバンド幅行列の測定に際してこの問題を避けるために，一度に一つのペアについてしか測定できないという制約を設けている．実際にはシステム内の全ホストで唯一のトークンをホスト同士で受け渡しするトークンリングを生成し，トークンを所持しているホストのみが測定ペアの一方となることができるという機構を用いている．

このような原理では $O(N^2)$ の計算時間がかかってしまうのは明らかで，エンドホスト数の増加に対してスケーラブルとは言えず，また単に $N \times (N-1)$ 回の測定を自動化しているというだけではスマートな手法とも言い難い．我々の研究は，並列分散アプリケーションの最適化の為にユーザ自身が高速にバンド幅マップを取得できるようにすることが目的の一つであった．これを達成するためには，この原理に加えて測定の並列化などの工夫を加えて，高速かつスケーラブルな手法にする必要がある．またこの手法では，$N \times N$ のバンド幅行列しか得ることができないのは明らかであり，我々が目標としているバンド幅マップの取得をするには不十分なものである．

2.3 バンド幅マップ構築手法

バンド幅マップを構築する初期の研究の一つに *TopoMon* [6] がある．TopoMon は NWS 上の拡張モジュールの一つとして提供され，バンド幅行列の情報とネットワークトポロジーの情報からバンド幅マップを構築する．手順として，TopoMon はまず NWS を使用してシステムのバンド幅行列

を取得し，同じく NWS で取得したネットワークトポロジーの各エッジにバンド幅の値を振っていき，バンド幅マップを完成させる．NWS がバンド幅行列を構築する方法は 2.2 節 に記述したとおりで，ネットワークトポロジーは全ホスト対全ホストの traceroute の結果から取得される．各エッジへのバンド幅の振り方は以下のようにする．ホスト A, B 間のバンド幅が X だとすると，そのパスに沿った全てのエッジのバンド幅は “X かそれ以上” と言うことができる．これを全てのホストペアについて繰り返し，エッジのバンド幅の値を更新していく．

こうしてバンド幅マップを構築していくが，TopoMon の出力するバンド幅は “それ以上” という情報でしかなく，エッジの確定的なバンド幅の値を報告しているわけではない．これはそもそものNWS が出力するバンド幅行列が End-to-End の測定で得られたものであるために，2.1 節で言及した中間エッジのバンド幅を測定できていないためである．[6] の論文中ではこの問題への対処を Future Work としており，我々の研究はこれも考慮に入れたバンド幅マップを対象とする．また TopoMon のバンド幅行列の取得には NWS を用いており，$N \times (N-1)$ の時間がかかるのは変わらない話であるので，我々の手法では測定の高速化とスケーラビリティの話も取り入れる必要がある．

bhtree[16] は我々が過去に行った研究である．背景や目的は本研究同じとして大規模ネットワークのバンド幅マップを構築するアルゴリズムの提案をした．この手法では End-to-End のバンド幅測定を一つのペア間だけで行うのではなく，一斉に多対多のバンド幅測定を行う．これによって，一対一では測ることのできなかった，ボトルネック以外の中間エッジのバンド幅測定を可能にしている．また，与えられたネットワークトポロジーを細かく分解し，測定の競合しない部分トポロジーに分割してからそれらを並列に測定することで，高速性と高いスケーラビリティを得ることができた．

しかし，この研究では問題や実装を単純にするため，ホスト間のルーティングトポロジーはツリーであると仮定し，また各リンクのバンド幅は対称でなければならないという制限を設けていた．すなわち，実環境のネットワークには多く存在するはずのバンド幅非対称なエッジや循環構造をもつネットワークに対して，正しい答えを出すことができなかった．また，アルゴリズムの特性により，特定のトポロジーを入力すると中間エッジの測定を誤るという問題も残されている．本研究では，ネットワークトポロジーやエッジの向きなどにそのような制約を課すことなく，一般のネットワークに適用可能なバンド幅マップ構築アルゴリズムを目指す．また，bhtree の実装ではネットワークの直径やスイッチの分岐数に応じてアドホックな処理を行う部分もあり，アルゴリズムとして見通しの良いものではなく，一般性に欠けていた．本研究ではより見通しの良いアルゴリズムを考案し，あらゆるネットワークを統一的な視点から取り扱うことのできるものを提案する．

ここで参考までに，bhtree の実験で得られたバンド幅マップを可視化したものを図 2.1 に示す．得られた結果としては，300 ホスト程がネットワーク接続された広域分散環境において，およそ 2 分でバンド幅マップの構築ができた．図中の灰色のノードがスイッチでオレンジ色のノードがエンドホストを表す．また，各所に記入してある数字は各スイッチまたはホスト間を結ぶエッジのバンド幅（Mbps）となっている．

これまで述べてきた手法とはややアプローチは違うが，*SNMP* や *BWCTL* を用いたバンド幅測定

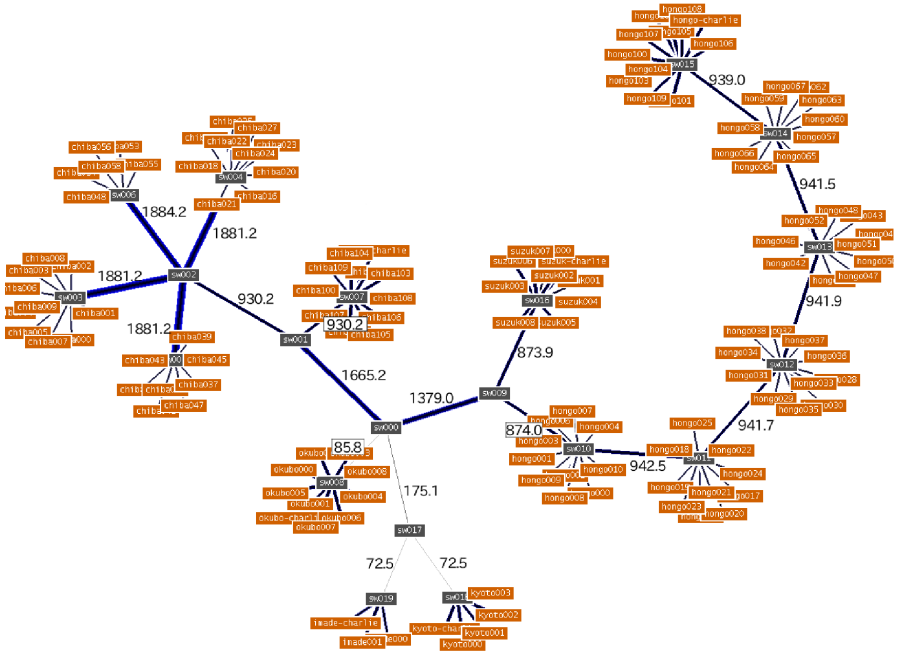


図 2.1: ***bhtree*** の出力

手法やネットワークモニタリングフレームワークも近年考案されている [3][4]．SNMP はスイッチやルータなどのネットワーク機器にアクセスし，様々な統計情報を取得するプロトコルである．あるポートにどれ程のパケットが通過したかの情報を SNMP によって取得すれば，適当な時間間隔の傾きがそのポートに繋がるリンクのバンド幅ということができる．BWCTL は，Iperf や *Thrulay*[17] といった基本的な End-to-End バンド幅測定ツールのラッパーで，BWCTL のデーモンが動いているノードに対してそれらのバンド幅測定ツールを実行するよう命令することができるフレームワークである．中間エッジの両端のルータにそのデーモンが走っていれば，BWCTL によってその中間エッジのみに対して直接バンド幅測定ツールを適用することができる．こういった仕組みによって中間エッジを測定していくことで，環境全体のバンド幅マップを生成することができるのではないかと考えられる．

しかし，SNMP であれ BWCTL であれ，スイッチを含む全てのネットワーク機器に設定がなされていなければ，バンド幅マップを生成することはできない．この設定を利用ネットワーク内の全ての機器に導入するのは非現実的であり，またそれ自体に対応していない機器も多く存在する．また，バンド幅マップ構築における高速化やスケーラビリティの話はここでは一切されておらず，たとえ中間エッジのバンド幅を測定できたとしても，全てのエッジに対してそれを実行するだけでは実用的なバンド幅マップ構築アルゴリズムということはできない．加えて，SNMP の管理情報データベー

スや BWCTL のデーモンプログラムにはその管理者権限を要求したり，または特定のユーザのみ受け付ける認証や IP アドレスによるアクセス制限をかけることができ，実際そうなっている場合も多い．我々の研究が想定している実行環境は，複数拠点が WAN で結ばれたような大規模分散環境であり，そのような状況において全てのネットワーク機器にアクセスできる権限を持っているということは考えにくい．

第3章　提案手法

3.1　問題の仕分け

提案手法の詳細に入る前に，我々がとりかかるべき問題を大きく二つに分類する．2 章で挙げた既存手法の問題点を総合的にとらえた結果，本研究で取り扱うべき問題のキーワードは以下の二つであると考えた．

- 中間エッジのバンド幅測定
- 並列化

後者は単なる高速化をするということだけでなく，適用環境の拡大に対してスケールするものにしなくてはならない．これら二つを満足するものでないと実用的なバンド幅マップ構築アルゴリズムとは言えないことは今まで述べてきた通りであり，我々の研究の目的を達成するためにも必要である．本研究では多対多協調のバンド幅測定と，それらの並列化によってこれらの課題を解決する．

この章は以下の様な構成になっている．3.2 節で，求めるべき正しいバンド幅は何で得られるかという定義をした後，中間エッジに対する多対多協調のバンド幅測定方法を述べる．3.3 節では手法の並列化について述べる．3.2 節と 3.3 節はそれぞれ独立した話であるので，それら二つを統合した最終的なアルゴリズムの形を 3.4 節で述べる．

本手法はネットワークトポロジーとそのルーティング情報を入力として，バンド幅マップを出力する．ネットワークトポロジーとそのルーティング情報は与えられていることを前提としている．これら情報は traceroute や既存のトポロジー推定手法 [18]，またはそれらの組み合わせによって容易に取得することができる．

以降，入力トポロジーを G，ルーティング情報を $\boldsymbol{R}$，そしてバンド幅を W で表す．トポロジー G は，エッジ $\boldsymbol{E}$ とノード $\boldsymbol{V}$ で構成されるグラフ $G(\boldsymbol{E}, \boldsymbol{V})$ である．本研究で言及するエッジは，物理的には一つのリンクであっても非対称なバンド幅を持つ二つエッジとして扱い，それぞれ $\boldsymbol{E}$ の別の元として取り扱う．ルーティング情報 $\boldsymbol{R}$ は，Source ノードと Destination ノードのペアからエッジの行ベクトルへのマップであり，$\boldsymbol{R} : (s, d) \mapsto (e_0, e_1, \ldots, e_m), \quad s \in \boldsymbol{V}, d \in \boldsymbol{V}, e_k \in \boldsymbol{E}$ と表せる．またこのエッジの行ベクトルをパスといい $\boldsymbol{p}$ で表し，このネットワークに存在する全てのパス，すなわち $\boldsymbol{R}$ の像を $\boldsymbol{P}$ で表す．バンド幅 W はエッジからそのバンド幅の値への変換であり，エッジ e のバンド幅を $W(e)$ と表す．

Let $\boldsymbol{P}_{e_t}$ denote the set of paths that traverse e_t, that is:

$$\boldsymbol{P}_{e_t} := \{\boldsymbol{p} \mid e_t \in \boldsymbol{p}\} \quad (1)$$

Then we get the bandwidth of e_t as follows:

$$W(e_t) := Sum\_of\_Simultaneous\_Measurements\_of(\boldsymbol{P}_{e_t}) \quad (2)$$

図 3.1: バンド幅の定義

3.2 バンド幅測定

3.2.1 定義

本研究では一つのエッジについてそのバンド幅を以下のように定める．今，バンド幅を知りたい一つのエッジの一方向に着目しているとする．まずシステム中の $N \times (N-1)$ 組あるホストホストペアのうち，そのルーティング経路中にそのエッジを含むようなホストペアを列挙する．ただし N はシステムに存在するエンドホスト数である．次にそれらのホストペア間でそれぞれの End-to-End のバンド幅測定を一斉に開始する．求めたいバンド幅とはそれらの観測値の合計である．言い換えると図 3.1 のようになる．

このようにしてバンド幅を決定してよい理由を説明する．まず図 3.1 の (1) の意味は，今 e_t のバンド幅を測定したいのであるから，全てのルーティングパス $\boldsymbol{P}$ のうち e_t をその経路に含むパス $\boldsymbol{P}_{\boldsymbol{e_t}}$ のみを考えればいいことは明らかである．$\boldsymbol{P}_{\boldsymbol{e_t}}$ 以外のパスを通るようなネットワークストリームを発生させるバンド幅測定を行っても e_t には流れ込むことがなく観測バンド幅には何の影響もない．こうすると逆に，ネットワークグラフ上の別の箇所でのストリームの衝突などにより e_t の本来のバンド幅より小さく報告されてしまうこともありうる．

次に図 3.1 の (2) の意味は，一対一のバンド幅測定では使い切ることのできなかったエッジのバンド幅を，多数のバンド幅測定によるネットワークストリームで飽和させ，利用できるバンド幅の最大値をそれらの和によって表現している．もしここで $\boldsymbol{P}_{e_t}$ を流れる全てのストリームを使わずに測定したのでは e_t のバンド幅を飽和させることができず，本来あるバンド幅より小さい値が報告されてしまう可能性があるので，$\boldsymbol{P}_{e_t}$ の全てのパスに同時にネットワークストリームを発生させ合計を取る必要がある．

図 3.1(2) で e_t のバンド幅を飽和させることができたと判断できた場合，それを e_t のバンド幅だと決定できる．一方で $\boldsymbol{P}_{e_t}$ 全てのパスにネットワークストリームを発生させてもなお e_t のバンド幅を飽和させることができない場合も考えられる．しかし我々の研究ではたとえそうであったとしてもそれを e_t のバンド幅だと決定してもよい．なぜならば，それはシステム全体を導引してでさえ e_t のバンド幅は使い切ることができないほど大きいということを意味するので，e_t のバンド幅は上と同

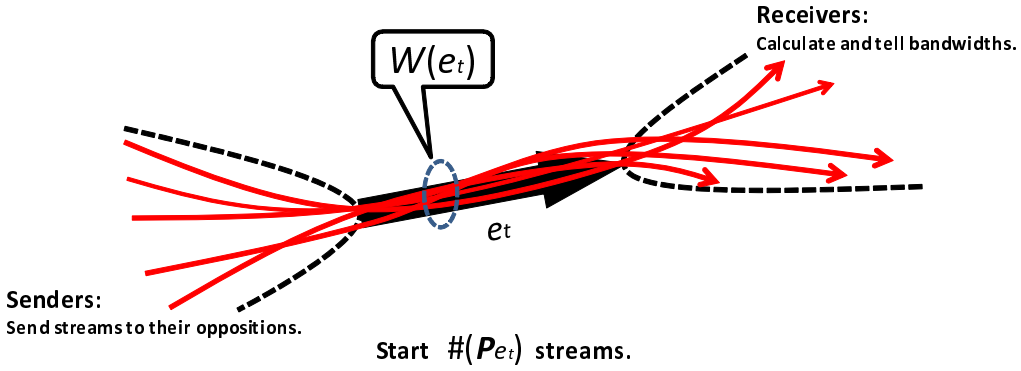


図 3.2: 最大本数ストリームによるバンド幅測定

様に定義しても実用上問題はないと言えるからである．もしくは出力の段階で “この値以上” と注釈を付けることは容易にでき，実用上誤った答えを出すことは無い．いずれにせよ本研究の貢献は分散アプリケーションの通信最適化であるため，実用上の最大可用帯域を報告するバンド幅測定手法であれば我々の目的は達成される．

以上によって，いかなる場合でも図 3.1 で求めたバンド幅 $W(e_t)$ はエッジ e_t のバンド幅であると言ってもよい．図 3.1 を図式化すると図 3.2 のようになる．なお実装上では図 3.1 中の $\boldsymbol{P}_{e_t}$ の選別や，パス $\boldsymbol{p}$ にストリームを発生させるにはどのエンドホストのペア間で通信をすればいいかなどの判断は，入力の $\boldsymbol{R}$ から求めることができる．

バンド幅測定を一斉に開始しそれらの和をとる操作を図 3.1(2) では *Sum of Simultaneous Measurements of* と表現しているが，この部分は 3.2.1 節の定義の意味を崩さない限り様々な実装が考えられる．以下の 3.2.2 節では naïve な実装例を紹介し，3.2.3 節ではなるべくネットワークに負荷をかけない *m*oderate な実装例を紹介する．

3.2.2 naïve な実装

3.2.1 節の定義通りの実装を行えば中間エッジのバンド幅を測定可能な一つのプログラムが完成する．本稿ではこれを naïve なバンド幅測定手法と呼ぶ．その擬似コードを Algorithm 3.1 に示す．

コードの意味を説明する．まず Algorithm 3.1 ではある一つのエッジ e_t のバンド幅を測定しようとしている．行 1 の $\boldsymbol{P}_{e_t}$ はエッジ e_t を通過するパスを形成するような Source ノードと Destination ノードのペアの集合 $\{(s_0, d_0), (s_1, d_1), \ldots, (s_n, d_n)\}$ である．行 1 から行 6 で，入力の $\boldsymbol{R}$ を用いて，そのような集合 $\boldsymbol{P}_{e_t}$ に初期化している．すなわち，行 6 において次の式（3.1）が満たされている．

$$\boldsymbol{P}_{e_t} = \{(s, d) \mid e_t \in \boldsymbol{R}(s, d)\} \tag{3.1}$$

Algorithm 3.1 naïve なバンド幅測定の擬似コード

Require: Routing table $\boldsymbol{R}$.

1: $\boldsymbol{P}_{e_t} := \varnothing$;
2: **for** $(src,\ dst),\ path$ **in items of** $\boldsymbol{R}$ **do**
3: **if** $e_t \in path$ **then**
4: $\boldsymbol{P}_{e_t} \leftarrow \boldsymbol{P}_{e_t} \cup \{(src,\ dst)\}$;
5: **end if**
6: **end for**
7: **parallel for** $(src,\ dst)$ **in** $\boldsymbol{P}_{e_t}$ **do**
8: $w_k := EndtoEnd\_bandwidth\_measurement\_between(src, dst)$;
9: **join and end for**
10: $W(e_t) := \sum w_k$;

続く行 7 から行 9 で $\boldsymbol{P}_{e_t}$ に含まれる全てのペア間で End-to-End バンド幅測定が一斉に実行され，それぞれの観測値が記憶される．Algorithm 3.1 中に記述されている *EndtoEnd bandwidth measurement between* は，例えば Iperf をホスト *src* とホスト *dst* 間で実行し，その観測値を返すような操作でよい．そして行 10 で求めたいエッジ e_t のバンド幅，$W(e_t)$ を先ほどの観測値の和として，一つのエッジのバンド幅を測定するプログラムが終了する．以上の意味をまとめて，Algorithm 3.1 の抽象度を上げた書き方をすると Algorithm 3.2 のように書ける．当然これは，定義である図 3.1 と意味的にほぼ同じ形になる．Algorithm 3.2 の 1 行目が Algorithm 3.1 の 1 から 6 行目に相当し，2 行目が 7 から 10 行目に相当する．

naïve な手法ではこのように簡単に中間エッジバンド幅測定プログラムを記述することができる．しかし，この方法では一つのエッジを測定するためだけにシステム中のエンドホストを総動員しなくてはならず，エンドホストにもネットワークにもかかる負荷は大きい．

3.2.3 moderate な実装

3.2.2 節で挙げたバンド幅測定アルゴリズムは負荷が大きく現実的な手法とは言うことができない．そのため，3.2.1 節の定義の意味を崩さない範囲で，エンドホストとネットワークへの負荷を軽減したアルゴリズムを考える必要がある．そのような実装例をこの節で述べ，本稿ではこれを moderate なバンド幅測定手法と呼ぶ．

Algorithm 3.2 naïve なバンド幅測定

Require: Routing table $\boldsymbol{R}$.

1: $\boldsymbol{P}_{e_t} := \{(s, d) \mid e_t \in \boldsymbol{R}(s, d)\}$;
2: $W(e_t) := Simultaneous\_EndtoEnd\_bandwidth\_measurement(\ \forall (s, d) \in \boldsymbol{P}_{e_t}\ )$;

負荷を軽減させるためには，バンド幅測定の結果に全く関与しない冗長なネットワークストリームを流さないようにすればよい．naïve な実装では，求めたいエッジの最大可用帯域を飽和させるために，そこを通過するネットワークストリーム全てを一斉に発生させていた．しかしそのように全てを使わずとも求めたいエッジが飽和する場合がある．その場合，全てのネットワークストリームのうち飽和させる本数のみを発生させるようにすれば，システムにかかる負荷を軽減することができ，それのために動員するエンドホスト数も削減することができる．

これは，初めから全てのストリームを一斉にスタートさせるのではなく，一つずつスタートさせることで実現できる．まず，求めたいエッジを通過するパスを全て列挙する．そこから一つずつパスを選びストリームを発生させ，飽和したと判断できた時点で，それまでのストリームによるバンド幅測定の結果の和を答えとする．飽和したかどうかは，ある時点での和とその次のストリームを発生させた後での和に変化がなければそう判断することができる．それ以降のストリームは，発生させても観測バンド幅の値の和が変化することはないと考えられるので，流す必要はない．このようにして，列挙したパス全てにストリームを流すことなく，最大可用帯域を求めることができる．

このようにして得た結果が 3.2.1 節のバンド幅の定義の意味を失っていないことは，以下のように説明できる．求めたいエッジを通過する全てのパスの集合から一つずつ選びそこにストリームを流していき，ある時点で飽和，すなわち観測バンド幅の値の和が頭打ちになったとする．その時点での残りのパスを横切るストリーム，すなわちもとのパスの集合のうちまだストリームが流されていないパスは，既に観測値ゼロのストリームが仮想的に流されているものと見なすことができる．このように考えることで，もとの集合の全てのパスにストリームを流し終えていることと捉えることができ，3.2.1 節の定義よりその観測値の和をエッジのバンド幅として決定してよいということが言える．以上のようにして，バンド幅測定の定義を崩さない範囲でシステムにかける負荷を軽減する方法が考えられた．

ただし以上の手法は飽和したかどうかの判断を慎重に行わなければならない．上で述べたような，ある時点での和とその次のストリームを発生させた後での和との変化を見るだけの方法では結果を誤る場合がある．なぜならば，観測バンド幅の値の和が頭打ちになるということが，着目しているエッジに因るものとは限らないからである．それは，それまでに選択されたパスが，着目しているエッジ以外の箇所のエッジを共有している場合に起こりうる．求めたいエッジを通過するパスを全て列挙すると，それらのパス中には当然そのエッジが含まれるが，それ以外のエッジを共通に含むいくつかのパスが存在する場合がある．そのような場合では，ある時点で和に変化が起きなかったとしても，そこで着目しているエッジが飽和したと言い切ることはできない．この段階では，着目しているエッジが飽和したことが原因なのか，それとも他の箇所で共有されているエッジが飽和したためなのかは判断することができない．よって，飽和したかどうかの判断は例えば以下のようにする必要がある．あるパスにストリームを流した時点でのバンド幅観測値の和とそのストリームを流す前の和に変化がなく，かつ，それまでにストリームを流していたパス全てに共通するエッジがただ一つである場合，そのエッジは飽和しており，そのバンド幅は今求めた和である．

以上のことをふまえて moderate なバンド幅測定手法の擬似コードを書くと Algorithm 3.3 のようになる．

コードの意味を説明する．まず Algorithm 3.3 ではある一つのエッジ e_t のバンド幅を測定しようとしている．行 1 の $\boldsymbol{P}_{e_t}$ はエッジ e_t を通過するパスを形成するような Source ノードと Destination ノードのペアの集合 $\{(s_0, d_0), (s_1, d_1), \ldots, (s_n, d_n)\}$ である．行 1 から行 6 で，入力の $\boldsymbol{R}$ を用いて，そのような集合 $\boldsymbol{P}_{e_t}$ に初期化している．すなわち行 6 では Algorithm 3.1 と同様に，式（3.1）が満たされている．

行 7 の $\boldsymbol{M}$ は，現在ネットワークストリームを発生させている Source ノードと Destination ノードペアの集合である．moderate な手法はネットワークストリームを一つも発生させていない状態から始めるので，$\varnothing$ で初期化している．行 9 の while ループの条件 $\boldsymbol{P}_{e_t} \setminus \boldsymbol{M} \neq \varnothing$ は，行 6 までで生成した $\boldsymbol{P}_{e_t}$ に含まれる全てのペア間でネットワークストリームを発生し終わるまでループし，全て発生し終わっても飽和しなかった場合，それまでの和を答えとしてループを抜けだしプログラムを終了するということを意味している．なお，最初から $\boldsymbol{M} := \boldsymbol{P}_{e_t}$ としているものが naïve な手法に相当すると考えられる．

while ループに入り行 10 は，$\boldsymbol{P}_{e_t} \setminus \boldsymbol{M}$，すなわち行 6 までで生成したホストペアのうち，まだネットワークストリームが流されていないペアから一つのペアを選択し，そのペア間に持続的なネットワークストリームを流し始めることを意味している．Algorithm 3.3 中に記述されている *Start streaming between* は，例えばホスト s, d で socket を開き，s から d に向かってダミーデータを送信し続けるような操作でよい．行 10 でストリームを発生させ始めたペアを，次の行 11 で $\boldsymbol{M}$ に追加している．行 12 では，その時点での観測バンド幅の和を W_{new} に代入している．Algorithm 3.3 中に記述されている *Sum of observed bandwidth now* は，ダミーデータを受信しているホストがある間隔でその時々の瞬間バンド幅を定期的に更新しつづけ，このメソッドが呼ばれた時のバンド幅をプログラムのマスターに報告し，マスターがその和を返す，などという操作が考えられる．ここでのバンド幅の和が，前回集計したバンド幅の和と等しければ，e_t が飽和したかどうかを判別する処理である行 14 から行 26 の処理が始まる．条件分岐である行 13 で，もし $W_{prev} < W_{new}$ であれば，W_{prev} を W_{new} の値に更新して while ループの先頭に戻り，さらに追加でもう一つのペア間にストリームを発生させる処理に続く．その意味で W_{prev} は行 8 で 0.0 に初期化されている．

e_t が飽和したかどうかを判別する処理である行 14 から行 26 の前半部分を説明する．行 14 の $\boldsymbol{l}$ は今発生しているストリームが共通して使用しているエッジの集合を意味する．行 14 から行 17 で，$\boldsymbol{M}$ を用いて，そのような集合 $\boldsymbol{l}$ となるように初期化している．すなわち，行 17 では次の式（3.2）

$$\boldsymbol{l} = \left\{ e \;\middle|\; \bigcap_{(s,d)\in\boldsymbol{M}} \boldsymbol{R}(s,d) \right\} \tag{3.2}$$

が満たされている．続く行 18 から行 22 では，$\boldsymbol{P}_{e_t}$ のうちその時点で既に流す必要が無いと判断できる Source, Destination ノードペアを $\boldsymbol{M}$ に加えることで，バンド幅ゼロの仮想的なストリームを発

Algorithm 3.3 moderate なバンド幅測定の擬似コード

Require: Routing table $\boldsymbol{R}$.

$\boldsymbol{P}_{e_t} := \varnothing$;
for (src, dst), $path$ **in items of** $\boldsymbol{R}$ **do**
  if $e_t \in path$ **then**
    $\boldsymbol{P}_{e_t} \leftarrow \boldsymbol{P}_{e_t} \cup \{(src, dst)\}$;
  end if
end for
$\boldsymbol{M} := \varnothing$;
$W_{prev} := 0.0$;
while $\boldsymbol{P}_{e_t} \setminus \boldsymbol{M} \neq \varnothing$ **do**
  $Start\_streaming\_between(s, d)$; $(s, d) \in \boldsymbol{P}_{e_t} \setminus \boldsymbol{M}$
  $\boldsymbol{M} \leftarrow \boldsymbol{M} \cup \{(s, d)\}$;
  $W_{new} := Sum\_of\_observed\_bandwidth\_now()$;
  if $W_{prev} == W_{new}$ **then**
    $\boldsymbol{l} := \boldsymbol{E}$;
    for (s, d) **in** $\boldsymbol{M}$ **do**
      $\boldsymbol{l} \leftarrow \boldsymbol{l} \cap \boldsymbol{R}(s, d)$;
    end for
    for (s, d) **in** $\boldsymbol{P}_{e_t}$ **do**
      if $\boldsymbol{R}(s, d) \supseteq \boldsymbol{l}$ **then**
        $\boldsymbol{M} \leftarrow \boldsymbol{M} \cup \{(s, d)\}$;
      end if
    end for
    if $\boldsymbol{l} == \{e_t\}$ **then**
      $W(e_t) := W_{new}$;
      return true
    end if
  end if
  $W_{prev} \leftarrow W_{new}$;
end while
$W(e_t) := W_{new}$; {or more.}
return false

生させることを意味する処理をしている．流す必要があるか無いかの条件分岐の行 19 は，$\boldsymbol{l}$ が，今流されているネットワークストリームが共通に使用しているエッジの集合を表していることを利用している．今，行 13 の条件に合致しているので，いずれかのエッジが飽和しているはずである．エッジの飽和は，複数のネットワークストリームに重複して使用されているエッジにおいて起こり，またそのようなエッジでしか起こらない．そのようなエッジは $\boldsymbol{l}$ で列挙されているため，$\boldsymbol{l}$ のうち少なくとも一つのエッジは飽和しているということが言える．故にあるパス中に $\boldsymbol{l}$ の全てのエッジが含まれていれば，そのパスにネットワークストリームを発生させても全体の観測バンド幅の和に変化がないことは明らかである．この条件を擬似コード上で言えば，行 19 のように $\boldsymbol{l}$ が $\boldsymbol{R}(s,d)$ の部分集合となっているかどうかをテストするということになる．

最後に，行 23 から行 26 では飽和したエッジが e_t であるかどうかを判別するコードである．あるパスにストリームを流した時点でのバンド幅観測値の和とそのストリームを流す前の和に変化がなかった時，飽和したエッジが e_t であると断定できるのは，それまでに発生させているネットワークストリームが共通して通過するエッジが e_t ただ一つの場合に限るということを以前述べた．行 23 の条件分岐がその判別である．もしその条件が成り立てば，求めたかった $W(e_t)$ に今の観測値の和 W_{new} を代入し，プログラムを終了する．そうでなければ，W_{prev} を W_{new} の値に更新して while ループの先頭に戻り，さらに追加でもう一つのペア間にストリームを発生させる処理に続く．行 9 から行 29 の while ループを抜けた場合，すなわち $\boldsymbol{P}_{e_t}$ の全てのペア間にストリームを発生させても e_t を飽和させたと判断できなかった場合，残る行 30, 31 を実行してプログラムは終了する．行 30 では $W(e_t)$ に W_{new} を代入しているが，もしまだ流せるストリームがシステム中に存在すればさらに大きいバンド幅を持っていると分かる可能性がある，という意味で，“or more” の注釈がなされている．ただし，入力の $\boldsymbol{R}$ にはそのようなストリームはもう存在しないので，このネットワークにおいては e_t のバンド幅はこれ以上の値が観測されることはなく，また利用されることもない．よって最大可用帯域として $W(e_t)$ を W_{new} と報告しても実用上問題はない．行 25 の **return true** に対して行 31 では **false** を返すようにしている．これは求めた $W(e_t)$ が確定的なものか，それ以上なのかを上層のアプリケーションで知り分ける為に使われることを想定している．以上の意味をまとめて，Algorithm 3.3 の抽象度を上げた書き方をすると Algorithm 3.4 のように書ける．Algorithm 3.4 の行 3，Pick one pair from the remains of $\boldsymbol{P}_{e_t}$ and start streaming は，Algorithm 3.3 の行 10, 11 に相当する．続く行 4 は Algorithm 3.3 の行 12, 13 に相当し，行 5 の Virtual streams emerged, if any は，行 14 から行 21 に相当する．Algorithm 3.4 の行 6 の飽和判定は，行 23 から 26 に相当する．

Algorithm 3.4 moderate なバンド幅測定

Require: Routing table $\boldsymbol{R}$.

```
1: P_{e_t} := {(s,d) | e_t ∈ R(s,d)};
2: while Not all pairs in P_{e_t} start streaming do
3:   Pick one pair from the remains of P_{e_t} and start streaming;
4:   if Additional stream has no impact to sum of bandwidths then
5:     Virtual streams emerged, if any;
6:     if e_t is saturated then
7:       W(e_t) is W_new;
8:       return true
9:     end if
10:  end if
11: end while
12: W(e_t) is W_new or more;
13: return false
```

3.3 並列化

3.3.1 基礎

個々のエッジについてバンド幅測定を行う手法は 3.2 節で述べたが，バンド幅マップ構築手法のためには測定の並列化が必要である．バンド幅測定の並列化とは，一度になるべく多くのエッジのバンド幅測定を行い全体の並列度を上げつつも，それらの測定をするために発生するネットワークストリームが互いに衝突しないようにスケジューリングをすることも含む．並列化を行わない場合は，入力されたネットワークトポロジーの全てのエッジについて逐次にバンド幅測定手法を実行すればバンド幅マップを得ることができる．このアルゴリズムを Algorithm 3.5 に示す．コード中の *Measurement* は，例えば 3.2 節で挙げたような naïve な手法や moderate な手法などの，中間エッジのバンド幅測定手法を実行しその値を返す操作を用いればよい．このアルゴリズムは非常に単純で実装も簡単であるが，エッジの数に比例した時間がかかるため非効率であまりスマートな方法とは言えない．しかし，ネットワークグラフを入力として確かにそのバンド幅マップを出力する基本的なアルゴリズムである．

Algorithm 3.5 基本的なバンド幅マップ構築手法

Require: Network topology $G(\boldsymbol{E}, \boldsymbol{V})$, routing table $\boldsymbol{R}$.

```
1: for edge in E do
2:   W(edge) = Measurement(edge, &R);
3: end for
```

並列測定を行うためのスケジューリングについて説明する．あるホストペア間でのバンド幅測定と，そのとき使用されていないエッジのみで構成されるパスを使ったバンド幅測定は，互いに干渉しないため並列に実行することができる．ネットワーク中の任意の数の任意のホストペア間の測定が互いに干渉するかどうかは，入力 $\boldsymbol{R}$ から知ることができる．すなわち，式（3.3）を満たすような集合に属する全ての Source ノード s，Destination ノード d のペアによるバンド幅測定は並列に実行することができる．

$$\left\{ (s,d) \middle| \bigcap_{s \neq d \in \boldsymbol{V}} \boldsymbol{R}(s,d) = \varnothing \right\} \tag{3.3}$$

よって，入力されたネットワークのエンドホスト群の直積にあたるペア群を，式（3.3） を満たす複数の集合でグループ分けをし，その時それぞれのグループにできるだけ多くの元を持たせて生成されるグループの数が少なくなるような集合族を形成することができれば，それが最適な並列測定スケジューリングとなる．そしてその全体の実行時間はグループの個数に比例し，逐次に実行した場合のエッジの数に比例するものと比べてはるかに効率化される．

ところが，このグループ分けをする計算には予め明確な答えや解法が一見して見つからず，全ての組み合わせをテストするというある種の箱詰め問題に帰着してしまう．バンド幅マップ構築の効率化を実現するためのスケジューリングに時間がかかってしまっては意味がないので，ある程度ヒューリスティックな切り口で最適に近い並列測定スケジューリングのグループ分けをしてやる必要がある．

その切り口一つに，ネットワークトポロジーをあるエッジで分断して複数の部分トポロジーに分解し，それぞれに逐次処理である Algorithm 3.5 を並列に適用して，完了次第マージしたものを全体のバンド幅マップとして答えを返すという方法が考えられる．これは前述の，衝突しないパスで並列測定グループを分けるという考え方ではなく，衝突しないグラフで並列測定グループを分けるという考え方であり，グループ分けの対称となる概念の粒度を荒くした考え方と言える．

この考え方で新たに問題となってくることは，トポロジー中のどのエッジで部分トポロジーに分割すれば，より最適に近く並列度の高いグループ分けになるかということである．3.3.2 節で詳しく述べるが，この手法の実行時間は，全体のトポロジーをルートとして，トポロジーが分割されていく様子をツリーで表現したときのツリーの深さに比例する（図 3.3）．ただし，それぞれの部分トポロジーのバンド幅マップ構築には逐次アルゴリズムが使われるため，ツリーのある深さの並列測定ステージを完了させる操作は，その深さにある部分トポロジーのうちエッジが最も多く含まれる部分トポロジーのバンド幅マップ構築に律速される．よって分割された部分トポロジーで同じ深さにいるものはエッジ数がほぼ同じになる様に，バランスを考えた分割方法が望ましい．例えば入力のネットワークがツリー構造であった場合，そのバランス木を考えてそのルートノードから子ノードを均等に分割していけばこのような分割ができる．2 章で紹介した bhtree では入力するネットワークをツリー構造に制限して，このような分割を行い測定の並列化を実現している．本研究では入力のネットワークがツリー構造とは限らないので，一般的なグラフにおいて，木でいうバランス木の

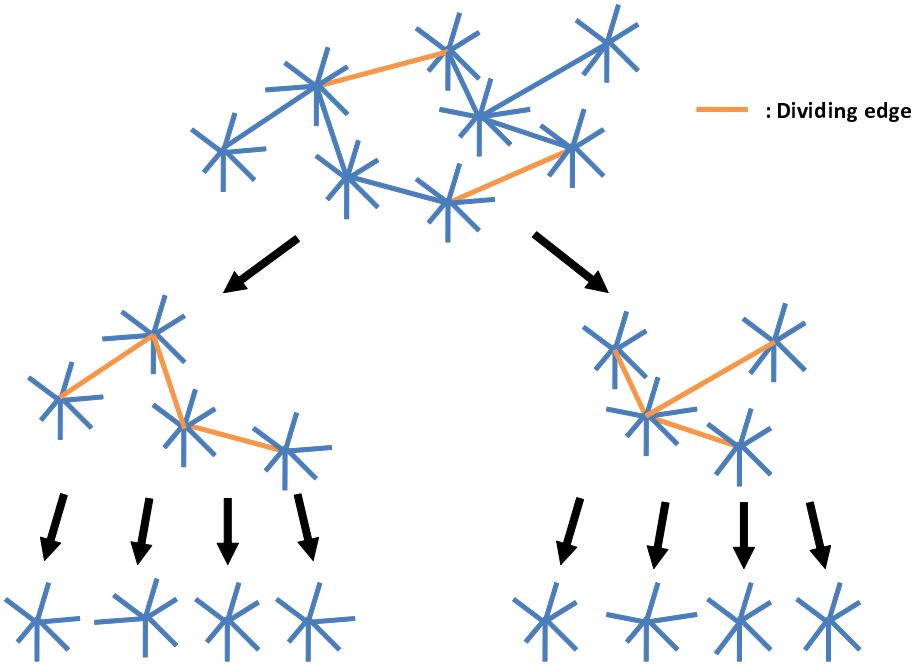


図 3.3: トポロジー分割のツリー表現

ルートという概念に相当する，グラフの中心に着目し，それに近いエッジから分割していくというアプローチをとる．

3.3.2 グラフ分割による並列化

本研究ではグラフをエッジで分割する際の目安としてエッジの *Betweenness Centrality* を用いた [9][5]．Betweenness Centrality とはグラフに存在するノードのうち，どのノードが最も中心らしいかを数値で表す指標である．Betweenness Centrality は全てのノードに対して割り当てられ，あるノードのその値とは，全ノード対全ノードの shortest path の数に対する，そのノードを横切った回数の比率で求まる．より多くの shortest path に現れるノードほど，グラフ中のノードとノードを橋渡しする役目になりやすく，故にグラフにおける中心性が高いという考え方からこの中心性の概念が考えられている．この定義を式で一般的に記述すると，グラフ $G := (\boldsymbol{V}, \boldsymbol{E})$ に対してノード v の betweenness $C_B(v)$ は式（3.4）で与えられる．σ_{st} とはノード s からノード t への shortest path の数であり，$\sigma_{st}(v)$ とはノード s からノード t への shortest path のうち，ノード v を通過するものの数である．

$$C_B(v) = \sum_{s \neq v \neq t \in \boldsymbol{V}} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{3.4}$$

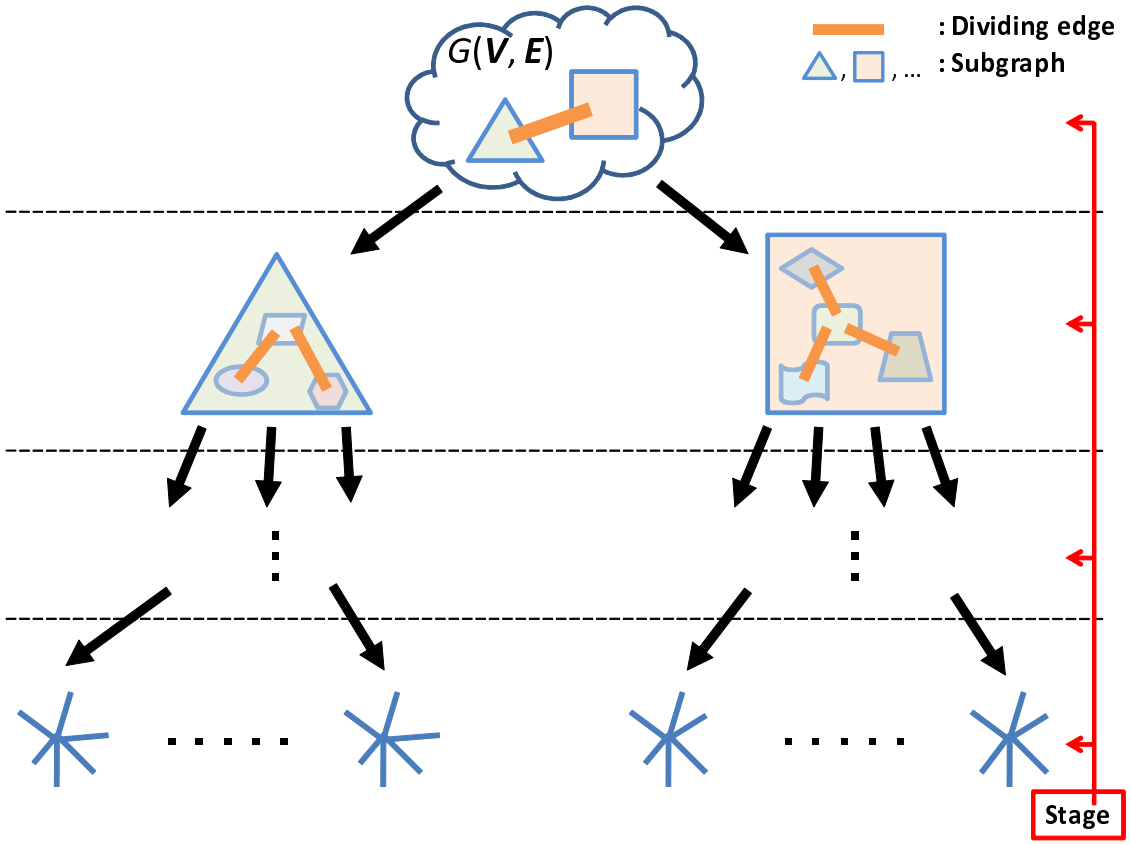


図 3.4: グラフ分割

ところで Betweenness Centrality は，その定義式のノードに当たる部分をエッジと読みかえることで，エッジに対しての Betweenness Centrality の値を計算することができる．本手法ではエッジに対する Betweenness Centrality を導入し，グラフを分割する際の目安とする．本稿の記号を使って式（3.4）をエッジに対する Betweenness Centrality の式に書き直すと，式（3.5）の様にかける．ただし $|\boldsymbol{A}|$ は集合 $\boldsymbol{A}$ の元の個数を表す．

$$C_B(e_t) = \frac{|\boldsymbol{P}_{e_t}|}{|\boldsymbol{P}|} \tag{3.5}$$

本手法ではまず入力のネットワークトポロジーの全てのエッジの Betweenness Centrality を式（3.5）によって算出し，その値が大きいエッジ，すなわちグラフの中央に近いエッジから順に切り離してグラフを分割していくということを行う．グラフが分割がされていく様子を図 3.4 のように描く．本研究にとってのエッジの Betweenness Centrality でグラフを分割していくということの意味とは，ネットワークグラフを中心からバランスよく分割ができるということ，そしてその結果並列

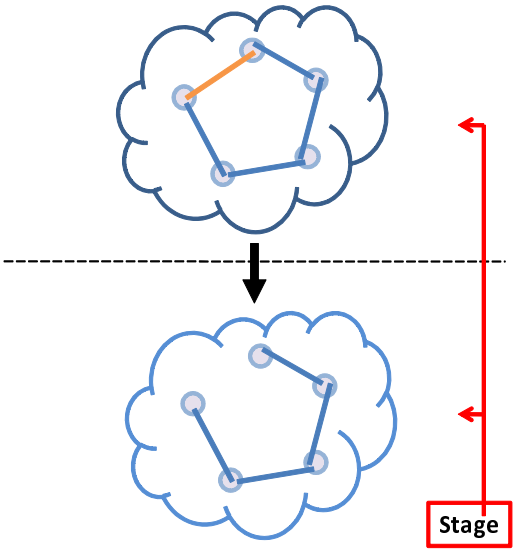


図 3.5: ループ解消を伴うグラフ分割

度高くバンド幅マップの構築が進むということ，そして全体的なアルゴリズムや実装の見通しを良くすることなどがある．2 章で我々の先行研究として紹介した bhtree では，入力のネットワークトポロジーはツリー構造であるという制限をかけ，まずツリーネットワークのバランス木を考えそのルートノードから子ノードを均等に分割していくというグラフの分割を行っていた．本手法の手順を適用すれば，ネットワークトポロジーがツリーであればそれと同等の分割がなされ，さらにツリーでなく循環構造をもつようなネットワークに対しても同一のアルゴリズムを適用するだけでバランスのよい分割処理をすることができる．つまり，バンド幅マップ構築の並列化を実現するための下準備として行う，ネットワークグラフの分割アルゴリズムの一般化がこれによってなされたと言える．分割処理は，ある大きさよりも部分グラフが小さくならないよう，ある閾値を設定し停止するようにしておく．通常，n 本のエッジを切断すると，その子として生成される部分ネットワークトポロジーは $(n+1)$ 個である．ただし，親のネットワークトポロジーが循環構造を持ち，エッジを分断したとしてもその子のネットワークトポロジーが連結である場合もある．そのような場合は，そのエッジで分断することによって生成される子のネットワークトポロジーは一つであるとして扱う（図 3.5）．より一般的に記述すると，ネットワークのある n 本のエッジを切断すことで k 個の循環構造が解消されたとき，その子として生成される部分ネットワークトポロジーは $(n+1-k)$ 個となる．

ネットワークグラフの分割処理がなされた次は，小さいグラフからボトムアップ式にバンド幅マップ構築とマージを行っていく．ネットワークグラフが分割されていく様子を先ほどの図 3.4 のように描いたとき，同じ深さにある部分ネットワークグラフ全てに対して，基本的なバンド幅マップ構築アルゴリズムである Algorithm 3.5 を並列に適用することができる．同じ深さにある部分ネットワーク

トポロジーは，もともと連結であった親のグラフのエッジを分断して生成されたものなので，それらは非連結でありネットワーク的にお互い独立していることが保証されている．よって，それらに並列に Algorithm 3.5 を適用してもバンド幅測定の衝突が起こることはなく，各部分ネットワークトポロジーに対して正しいバンド幅マップ構築がなされる．以上のように，ある一つの深さにある部分ネットワークグラフの全てのサブバンド幅マップ構築を並列に行うということを，ここでは一つのステージを終えると言うことにする．本手法ではまず図 3.4 の葉にあたる部分ネットワーク群のステージから開始し，ステージが終了次第，図 3.4 のツリーを遡るようにボトムアップ式にステージを完了させていく．最終的に図 3.4 のルートのステージ，すなわち入力トポロジーそのものに対するステージが完了して本プログラムは終了となり，それまでに求めてきたサブバンド幅マップをマージしたものを答えとして返す．ただ，各ステージにおいて素直に Algorithm 3.5 のバンド幅マップ構築を最初から始めていたのでは，分割と並列化の意味がない．あるステージを実行する際には，真っ新な状態から始めるのではなく，それ以前のステージで既にバンド幅測定がなされてバンド幅の値が確定しているエッジを除いて，残りの未確定エッジのみに対して Algorithm 3.5 の **for** 文を回すようにする．これによって，グラフの分割と並列測定による本手法の効率性の向上が実現される．理想的な実行フローをたどった一つの例を次に挙げる．あるステージが完了した時点でそこにある全てのバンド幅の値が確定したとすると，次のステージで行うべきバンド幅測定とは，ネットワークグラフ分割をした際に分断されたエッジに対してのみでよい，ということとなる．

以上のようにすることで効率的にバンド幅マップを構築していくことができるが，分割した後のサブグラフ内で発生させることのできるネットワークストリームの数は，分割する前に比べて少なくなっているということに注意しなければならない．我々の研究では，バンド幅は図 3.1 によってのみ決定できるということを述べた．すなわちあるエッジのバンド幅を測定するためには，図 3.4 のルートのステージにおいて，全てのエンドホストを動員してネットワークストリームを発生させてそのエッジを飽和させなければならなかった．この問題により，あるステージでは飽和させることができるがそれより下位のステージでは飽和させることのできないエッジが存在する可能性が出てくる．下位のステージで飽和していないにもかかわらず，そのエッジについてのバンド幅測定は完了したという誤った情報が上位のステージに伝搬していくと，並列化を適用したバンド幅マップ構築アルゴリズムは誤ったものになってしまう．

この問題を解決するために，Algorithm 3.5 の *Measurement* の操作に 3.2 節で述べた moderate なバンド幅測定を用いる．moderate なバンド幅測定は，仮想ネットワークストリームという考え方を用いることにより，$\boldsymbol{P}_{e_t}$ の全てのホストペア間にネットワークストリームを発生させなくとも測定対象のエッジが飽和したと判定することができるということを述べた．つまり，moderate なバンド幅測定手法を用いることで，図 3.4 のルートのステージにおける全てのエンドホストを動員せずとも，分割後の各サブグラフ内にあるエンドホストだけで，測定対象のエッジを飽和させることができたかどうかを判別することができる．そしてその判定は，プログラムから見ると Algorithm 3.3 の **return** 値が **true** であれば飽和すなわち確定，**false** であれば未確定であるとして扱うことができ

る．これを利用し，あるステージを実行している最中に各エッジのバンド幅測定とその値が確定的であるかどうかのフラグをたてる作業を同時に行う．そしてそのステージが完了し親のステージに取り掛かる際には，既にバンド幅の値が確定しているというフラグが立っているエッジを除いて，残りの未確定エッジのみに対して Algorithm 3.5 の **for** 文を回すようにすればよいということになる．以上によって，並列化を適用した正しいバンド幅マップ構築アルゴリズムとなる．Algorithm 3.3 で **false** が返された場合，すなわちそのサブグラフ内では動員できるエンドホスト数が足りず飽和させることができなかった場合は，そのエッジは未確定のままとし，上位のステージでもう一度測定対象とされることになる．上位のステージであるほどグラフに存在するエンドホスト数が多く発生させることのできるネットワークストリーム数も多いため，そのエッジを飽和させることができる可能性は高くなる．

最後に，並列化を適用した場合のアルゴリズムの実行時間について述べる．比較対象として逐次に行った場合と，並列に行ったときの最善と最悪の場合の計 3 つのパターンについて考え，実行時間を Algorithm 3.5 の *Measurement* が呼ばれる回数で比較する．まず，逐次に行った場合は明らかに $|\boldsymbol{E}|$ である．次にこの節で述べた並列化を適用したアルゴリズムを考える．一つのステージが完了するまでにかかる時間は，そのステージで行われるサブバンド幅マップ構築のうち最も時間のかかるものの時間である．サブバンド幅マップ構築に用いる Algorithm 3.5 はエッジの数に比例した時間がかかるので，一つのステージを完了させる操作は，そのステージにある部分ネットワークのうちエッジが最も多く含まれるもののサブバンド幅マップ構築に律速される．今，グラフ分割を表す図 3.4 のツリーが平均して m 分木で，深さが d であったとする．ある深さ x のステージに存在する部分ネットワークが含んでいるエッジの数は平均 $|\boldsymbol{E}|/m^x$ と表せる．すなわちステージ x は $|\boldsymbol{E}|/m^x$ だけの時間がかかるといえる．さて，この並列化アルゴリズムにおける最善の場合とは，全てのステージにおいて未確定エッジを残さずに測定が進む場合であり，そのときの実行時間は次の式（3.6）のように表すことができる．

$$\frac{|\boldsymbol{E}|}{m^d} + d\,(m-1) \tag{3.6}$$

第一項は初回のステージにかかる時間，第二項はグラフ分割の際に分断されたエッジを測定するためにかかる時間である．平均して m 分木，すなわち $(m-1)$ 本のエッジが各深さで分断されているので，第二項はこのような式になる．一方，この並列化アルゴリズムにおける最悪の場合とは，全てのステージにおいて確定エッジが一つもなかった場合であり，そのときの実行時間は次の式（3.7）のように表すことができる．

$$\begin{aligned} & \frac{|\boldsymbol{E}|}{m^d} + \frac{|\boldsymbol{E}|}{m^{d-1}} + \cdots + \frac{|\boldsymbol{E}|}{m} + |\boldsymbol{E}| \\ = \;& |\boldsymbol{E}| \left(1 + \frac{1}{m^d}\frac{m^d-1}{m-1}\right) \end{aligned} \tag{3.7}$$

上で求めた実行時間についてさらに詳しく解析する．グラフ分割をする際には，ある大きさよりもサブグラフが小さくならないように，ある閾値を設定して分割処理を停止するようにしておくこ

とが必要だと述べた．今，その閾値を t とする．t の意味は，あるグラフを分割しようと考えた時に，子として生成されるサブグラフのなかに t 以下の数のエッジしか持たないようなものが現れてしまう場合，その分割は行わないようにするということである．閾値を設定したとき，次の式が成り立つ．

$$t \simeq \frac{|\boldsymbol{E}|}{m^d} \tag{3.8}$$

この式は，m 分割が d 回行われたサブグラフ，すなわち図 3.4 の葉の部分にあるサブグラフは，そのエッジの数が t に近いであろうということを意味している．なぜならば，分割処理がそれ以上進んでいないということは，そのサブグラフのエッジの数が閾値 t の付近に達したためと考えられるからである．この関係を用いると，逐次版アルゴリズムの実行時間 $|\boldsymbol{E}|$ は

$$tm^d \tag{3.9}$$

と書け，式（3.6）で示した最善の場合の並列アルゴリズムの実行時間は

$$t + d\,(m-1) \tag{3.10}$$

と書ける．式（3.9）から式（3.10）を減算すると

$$\begin{aligned} & tm^d - (t + d\,(m-1)) \\ = \;& t\,(m^d - 1) - d\,(m-1) \\ = \;& t\,(m-1)(m^{d-1} + m^{d-2} + \cdots + m + 1) - d\,(m-1) \\ = \;& (m-1)\left(t\,(m^{d-1} + m^{d-2} + \cdots + m + 1) - d\right) \end{aligned} \tag{3.11}$$

となる．実際問題として t, m, d はそれぞれ $t \geq 3, m \geq 2, d \geq 1$ であるため，式（3.11）は常に正であると言ってもよい．すなわち，並列アルゴリズムが最善に進められた場合は必ず逐次版アルゴリズムより高速なものになるということが言える．またネットワークが大規模になると，一般的にネットワークグラフの直径が大きくなるので，d の値も必然的に大きくなる．式（3.9）は d の増加に対してスケーラブルな手法とは言い難いが，式（3.10）はそれより遥かに増加幅を小さく抑えられているということが分かる．次に並列アルゴリズムの最悪の場合の実行時間について述べる．最悪の場合の実行時間式（3.7）は，明らかに $|\boldsymbol{E}|$ より大きく，故に必ず逐次版アルゴリズムを行うよりも遅いということが分かる．式（3.7）と式（3.8）を用いて，どれほど実行速度が落ちるかということを

式で表すと，

$$
\begin{aligned}
(3.7) \quad &= |\boldsymbol{E}| \left(1 + \frac{1}{m^d} \frac{m^d - 1}{m - 1}\right) \\
&= |\boldsymbol{E}| \left(1 + \frac{t}{|\boldsymbol{E}|} \frac{\frac{|\boldsymbol{E}|}{t} - 1}{m - 1}\right) \\
&= |\boldsymbol{E}| \left(1 + \frac{1 - \frac{t}{|\boldsymbol{E}|}}{m - 1}\right) \\
&\approx |\boldsymbol{E}| \left(1 + \frac{1}{m - 1}\right) \qquad (3.12)
\end{aligned}
$$

と整理できる．ただし t は $|\boldsymbol{E}|$ に比べて非常に小さいとし，$1 - t/|\boldsymbol{E}| \approx 1$ の関係を用いた．実際問題として $m \geq 2$ であることが多いため，最悪の場合の実行時間は逐次版アルゴリズムに要する時間の高々2 倍であるということが式（3.12）から分かる．逆に考えて，m が大きければ大きいほど，すなわち一度により多くのエッジを分断し，一つのステージでより多くのサブグラフを並列に扱うことができれば，最悪の場合であってもそのロスを小さく抑えることができる．

この節ではグラフ分割によるアルゴリズムの並列化について述べ，並列アルゴリズムの最善，最悪の場合と逐次アルゴリズムの計 3 つの場合について定性的な性能評価を行った．並列アルゴリズムの実際の処理が最善のパターンを辿った場合，逐次アルゴリズムと比較して高速化され，またネットワークの拡大に対してスケーラブルな手法となることを示した．逆に最悪のパターンを辿った場合は，並列化を行わずに逐次アルゴリズムを実行した方が高速になるということも示した．この並列化手法が最善の場合に近くなるか最悪の場合に近くなるかの度合いは，各サブグラフのサブバンド幅マップ構築においてどれほど未確定エッジが残ってしまうかに依る．未確定エッジの本数というのは，$|\boldsymbol{E}|, t, m, d$ そして $C_B(e)$ などとは違い，アルゴリズム実行前には全く予想することができないパラメータである．よって並列アルゴリズムか逐次アルゴリズムかを適応的に選択することはできない．しかし実際の環境では並列アルゴリズムを使用するほうが適している場合が多い．なぜならば，一般にネットワークはグラフの外周にあるエッジほどバンド幅が小さく，中心に近いエッジほどバンド幅が大きい．これは，WAN のエッジのバンド幅は LAN 内のエッジのバンド幅より大きいという経験則に基づいている．これが意味していることは，外周にあるエッジほど数少ないネットワークストリームで飽和させることができる可能性が高いということである．本手法ではまず第一ステージとしてそのようなグラフの外周にあるエッジのみで構成されるサブグラフから開始するため，各ステージにおいて未確定エッジが数多く残ってしまうという確率は低い．我々の研究が対象としている環境の一つとして，全国各所に置かれているそれぞれ 10 から 100 ノードで構成される PC クラスタを，WAN で相互接続した大規模分散環境がある．このような環境の接続形態に対して Betweenness

表 3.1: 全体のアルゴリズムの形

	逐次	並列
naïve	YES	NO
moderate	YES	YES

Centrality によるグラフ分割を行えば，その性質より，第一ステージは必ず各所の PC クラスタ内のみのネットワークとなり，ステージが進むにつれて WAN のエッジも含むネットワークになる．クラスタを構成する LAN では 1Gbps の NIC が現在最も使用されており，このようなエッジは早々に飽和するため，第一ステージだけで各所のクラスタのサブバンド幅マップは全て完成してしまう場合がほとんどである．以上の様に，実際の環境では並列アルゴリズムが適している場合が多い．

3.4 全体像

この章では，まず本研究が扱うべき問題を中間エッジのバンド幅測定と並列化の二つに分け，3.2 節と 3.3 節でそれぞれ述べた．この節ではそれら二つを統合した最終的なアルゴリズムの形を述べる．

3.2 節では，中間エッジのバンド幅測定の naïve な手法と moderate な手法を紹介した．3.3 節では，プログラムの進み方について逐次に進める場合と並列に進める場合の両方を解説しながら紹介した．これらの仕組みを組み合わせることで本研究のアルゴリズムの全体像となる．全体のアルゴリズムの形として可能なものを，表 3.1 に示す．

並列化アルゴリズムにおいて naïve なバンド幅測定手法を用いることができないのは，少ないネットワークストリームで測定したバンド幅が確定的なものであるのか未確定であるのかを naïve な手法では判断できないためである．加えて，今まで述べてきた通り，スケーラブルな手法を実現するためにはアルゴリズムの並列化が不可欠である．よって本手法は中間エッジのバンド幅測定手法として moderate な手法を採用し，グラフを分割することでそれをステージに分けて並列に実行するという形をとる．以上を踏まえて最終的なアルゴリズムを擬似コードで記述すると Algorithm 3.6 のようになる．ここで行 1 の *divide_graph* は内部で 3.3.2 節で述べた手順でグラフの分割を行い図 3.4 の様なツリーのデータ構造を返す処理である．グラフの分割が終わると行 2 で *parallel_proc* 関数が呼ばれる．この関数は先ほどのグラフ分割で返されたツリー構造を再帰的に処理するための関数であり，行 4 から定義してある．*parallel_proc* ではツリー構造を再帰的に辿り，自分の子にあたるサブグラフのバンド幅マップ構築が全て終わるまで行 7 で待つ．自分の子のサブグラフに対する *parallel_proc* 関数が全て終わり次第再開し，自らのグラフのバンド幅マップ構築処理が行 8 の *building_bandwidth_map* によって行われる．*building_bandwidth_map* の中身は Algorithm 3.5 であり，Algorithm 3.5 の中身の *Measurement* は Algorithm 3.3 である．ツリー構造の葉の部分にあたるサブグラフは子を持って

Algorithm 3.6 バンド幅マップ構築アルゴリズムの最終形

Require: Network topology $G(\boldsymbol{E}, \boldsymbol{V})$, routing table $\boldsymbol{R}$.

```
1: g := divide_graph(G);
2: parallel_proc(g.rootGraph, &R);
3:
4: parallel_proc(graph, *routes) :
5:     parallel for subGraph in graph.children do
6:         parallel_proc(subGraph, routes);
7:     join and end for
8:     building_bandwidth_map(graph, routes);
9: exit
```

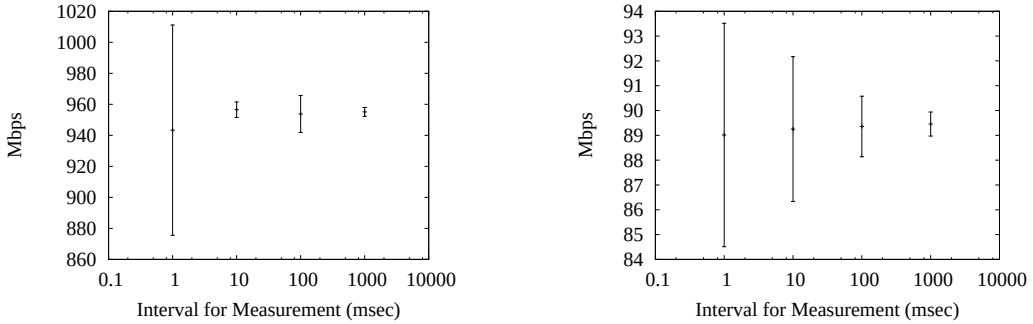
いないため行 5 から行 7 はスキップされ，このプログラム実行中で初めて *building_bandwidth_map* が呼ばれる．これが第一ステージの並列測定に相当し，ここよりボトムアップ式にステージが並列測定されていく．ルートのサブグラフ，すなわち実体としては入力のグラフ G に対する *parallel_proc* が完了すれば，プログラムは終了である．

3.5 実装

この節ではこれまで述べてきたアルゴリズムの具体的な実装方法について述べる．

本手法は分散アプリケーションにおける Master/Worker モデルで動作する．まず Master プロセスがいずれかの計算機で起動され，トポロジーデータ，ルーティング情報が入力される．Master はそのトポロジーを分割し，並列測定を進めるためのスケジューリングをする．処理が進み実際のバンド幅測定が必要になったら，該当するエンドホストのペアに，バンド幅測定を実行しその結果の値を返すようメッセージを送る．Worker プロセスはバンド幅マップ構築に参加する全てのエンドホスト上で起動される．Worker プロセスは Master からのメッセージを待ち続け，所定の相手とバンド幅測定を実行するよう Master からメッセージが届いたらすぐにバンド幅測定を実行し，得られた結果の値を Master に送り返す．以上のようなインターフェースを備える Master/ Worker フレームワークで本アルゴリズムを実装する．

上記のようなフレームワークを実装するのに GXP[21] を用いた．GXP は並列分散環境向けに作られたシェルであり，多数ホストの同時ログインや同時コマンド実行機能，プロセス同士のの ssh 通信網を用いた簡易メッセージパッシング機能などを提供する．GXP を用いて，まず Worker プロセスを起動させたいホストにログインし，次に本プログラムコードをそれらの上で同時に実行させる．Master と Worker の間のメッセージ交換は GXP の ssh 通信網上の簡易メッセージパッシング機能で実現できる．グラフのデータ構造の取り扱いについては，NetworkX[10][11] ライブラリを用いた．



(a) 同クラスタ内の計測時間と得られたバンド幅の分布

(b) 九州大/はこだて未来大間の計測時間と得られたバンド幅の分布

図 3.6: バンド幅測定にかけた時間と報告されたバンド幅

以上までの本実装の大枠は Python で記述されているが，ネットワークストリームを発生させる，受け取る，バンド幅を計算するなどといった高速な処理や高精度な計時が必要である部分に関しては C で記述されている．Python でこれらの処理を記述した場合，処理が追いつかずにバンド幅測定が本来より小さい値を報告してしまうなどの問題が生じるためである．ネットワークストリームは UDP によるパケットロスを無視した連続パケット送出で発生させる．TCP で最大可用帯域を測定しようとすると，パケットロスによる再送出処理や輻輳制御など影響により事実上の受信バイト数を計上することができない場合があり，これによって測定結果が誤る．

バンド幅測定を行う際にどれほどの間ネットワークストリームを流し続ければよいかについて簡単に言及する．本手法のバンド幅測定は，Iperf と同様の原理で，ネットワークストリームを流した時間で受信できたバイト数を割るという方法でバンド幅を計算している．この時間というのは全体のプログラムの実行時間にリニアに影響してくるものであるので，できるだけ短いほうがよい．しかし極端に短すぎると，ネットワークを経由する最中のノイズやエンドホストの負荷状況に影響されて，サンプルのパケットが適切に受信されずに本来のバンド幅の値を計算することが難しくなってしまうというトレードオフがある．本研究では，経験則からこの時間を 300 ミリ秒とすれば十分ばらつきの小さな結果が得られるとし，そのように設定している．図 3.6 は同クラスタ内の同じスイッチに接続されている二つのエンドホスト間，また九州大学とはこだて未来大学に配置されている二つのエンドホスト間のそれぞれについて，バンド幅測定にかけた時間と報告されたバンド幅の平均値と標準偏差を表した図である．横軸の各時間について 100 回実験を行っている．九州大学とはこだて未来大学の間の実験は，高遅延環境やパス中に多くのホップを含む長距離通信を想定している．

第4章　評価

4.1　実験環境

この章ではこれまで述べてきたアルゴリズムを実装し，InTrigger を実験環境としてその性能の評価を行う．InTrigger とは日本全国の大学に配置された PC クラスタを SINET3 [1]によって相互接続したもので，強力な計算資源を提供しているほか並列分散計算の研究のためのテストベッドとしての利用を想定して構築されている，大規模な並列分散システムである．現在は 15 のクラスタが稼働し，それらの接続形態は図 4.1 のようになっている．図中のアルファベットが振られているノードが一つのクラスタを表している．これらのクラスタは北海道から関東，関西，そして九州と幅広い地域に分散して配置されている．各計算機の性能やネットワーク性能はクラスタによって異なり，クラスタに含まれる計算機の個数も数個から数十個と様々である．全体としては WAN をまたいだ環状構造のあるネットワークを構成しており，大規模でヘテロな環境であると言える．

性能評価の詳細を述べる前に，本手法がこの環境に対して生成したバンド幅マップを可視化したものを図 4.2, 図 4.3 に示す．図中の数字はバンド幅を Mbps で表している．あるノードからエッジを通って別のノードへ向かおうとしているとき，今自分がいるノードから見てそのエッジの根本に振ってある数字がそのときのバンド幅である．

このようなバンド幅マップを構築するのにどれほどの時間を要するのか，またそれはどれほど正しいのかという二つの側面について，それぞれ 4.2 節と 4.3 節で評価する．

4.2　所要時間

本手法がどれほど高速でスケーラブルであるかの定量的な評価として，ノード数を増加させた時の所要時間の変化を，実環境の InTrigger を用いて調べた．本手法の他にも，比較対象として本手法の逐次実行版と，TopoMon が行うのと同様の手順を模倣したものの計 3 つの手法を取り上げて比較と検証を行った．それらのグラフを一つにまとめたものと，本手法のみを取り上げたグラフを図 4.4 に示す．ただし，TopoMon と同様の手順を実験で行うと，ノード数が増えるにつれて現実的な時間で終了しなくなってしまうため，一つのペアのバンド幅測定に 300 ミリ秒かかると想定し，図中のグラフは簡易的に $y = 0.3\,x\,(x-1)$ の曲線で表現している．この 300 ミリ秒という数字は 3.5 節で

[1] SINET3
http://www.sinet.ad.jp/

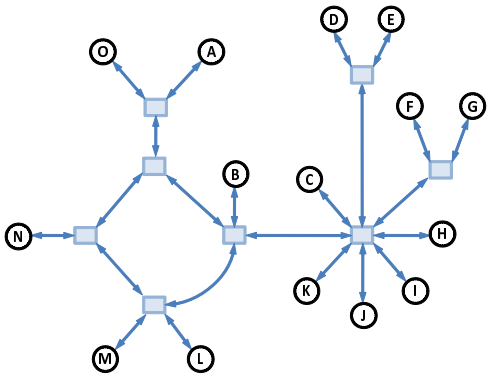


図 4.1: InTrigger の各クラスタの接続形態

述べた内容からきている．横軸はバンド幅マップ構築に参加したエンドホスト数である．この実験では横軸の値を増やすのに InTrigger 中のエンドホストをランダムに追加するのではなく，クラスタを一単位としてノード数を増やすようにしている．各計時はそれぞれのプログラムを 10 回走らせた時の平均値である．

まず図 4.4(a) より所要時間そのものを比較してみると，我々の手法は 311 ノードの環境で 50 秒程度でバンド幅マップ構築をすることができ，他の手法に比べてかなりの高速化がなされていることが分かる．逐次版のアルゴリズムでは 311 ノードで 1380 秒程度かかっており，本手法の並列化が効果をあげていることが分かる．TopoMon に基づいた手法は逐次版の本手法よりもさらに時間がかかり，311 ノードでは 28923 秒もかかっている．これは 2 章で述べたように，TopoMon は参加しているエンドホストの全ての組み合わせによる End-to-End バンド幅測定を逐次に行うため，所要時間はノード数の二乗のオーダーとなっている．この事実や図 4.4(a) から明らかに，TopoMon に基づいた手法は大規模な分散環境において適用することは不可能だと言ってよい．TopoMon-based の手法より我々の逐次手法の方が常に高速だという結果が出ているのは，この逐次手法の所要時間はノード数にほぼ比例したオーダーであるからである．我々の逐次手法はエッジの数に比例した時間がかかるということを以前述べた．InTrigger のような多数のクラスタを接続した分散環境においては，一つエンドホストを増やすということは一つのエッジが追加されるということを伴う場合がほとんどである．よってこれはノード数に比例したオーダーとなり，ノード数の二乗オーダーである TopoMon-based の手法より，我々の逐次手法の方が高速であると言える．さらに，グラフ分割による測定の並列化を行った本手法は，これらのいずれよりも常に高速だという結果になると予想され，実験からそのことが確かめられた．3.3.2 節で懸念した並列アルゴリズムの最悪の場合はこの実験では発生しなかった．

次に図 4.4(b) を考察する．ノード数がそれほど多くない範囲，およそ 100 ノードまででは，グラフの傾きはなだらかでありノード数の増加に対して実行時間の増加を小さく抑えることのできるス

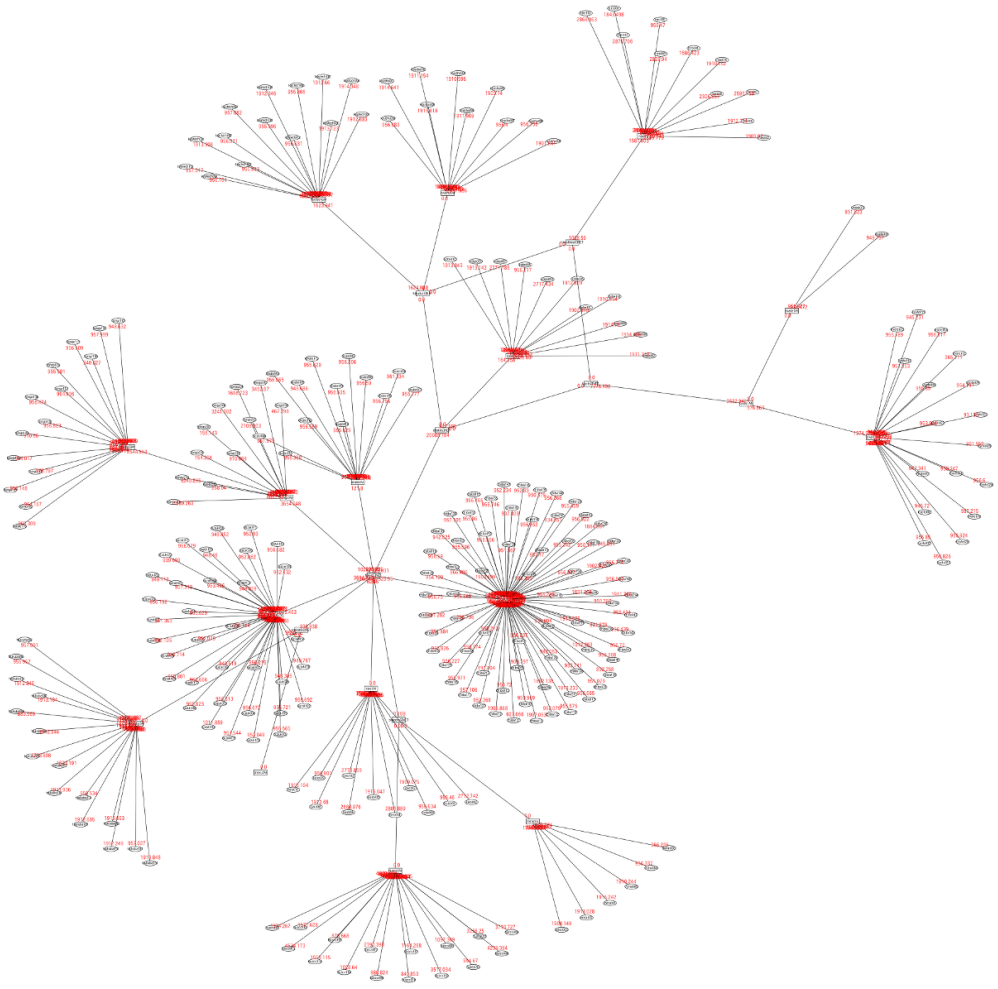


図 4.2: InTrigger のバンド幅マップ

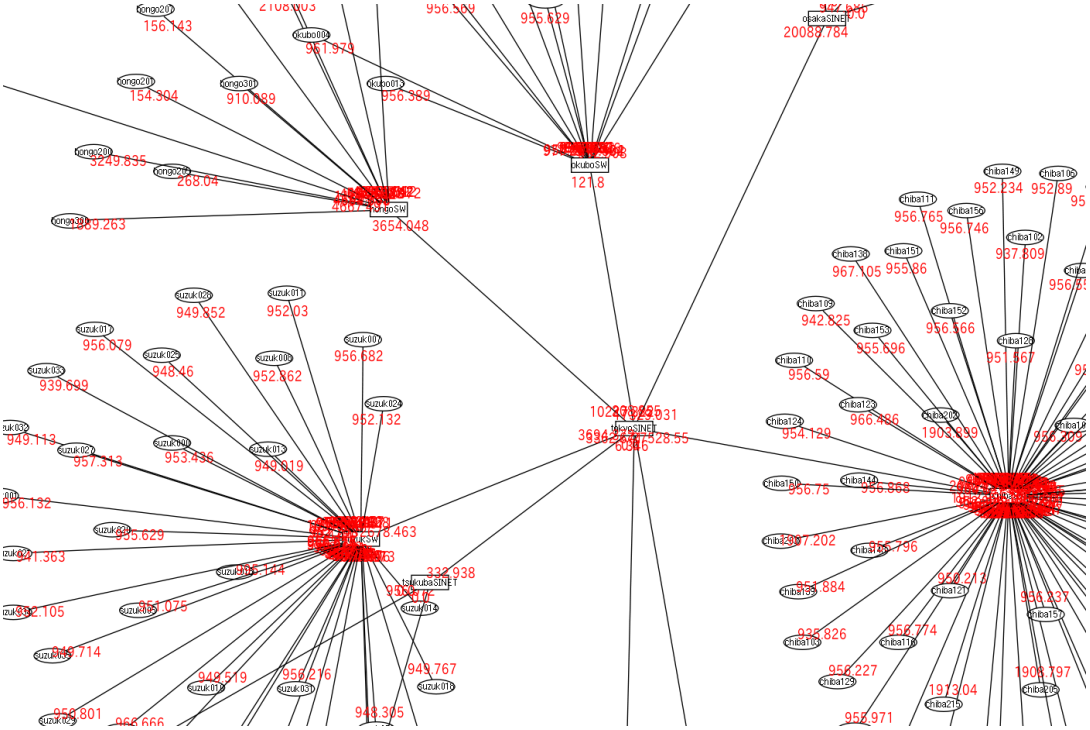
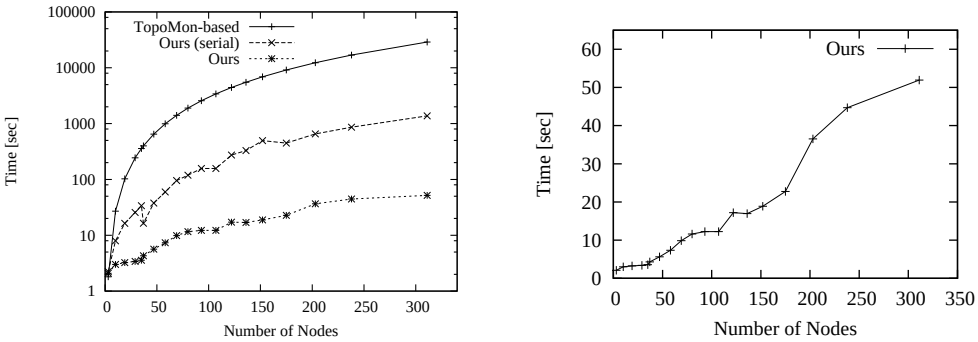


図 4.3: InTrigger のバンド幅マップ（拡大）



(a) 三つの手法の所要時間の比較　　(b) 本手法の所要時間

図 4.4: ノード数の増加に対する所要時間の変化

ケーラブルな手法であると見える部分もある．しかしグラフの右半分では急激に所要時間が増加している区間も見られる．これは，比較的規模の小さいネットワークに，多数エンドホストを有するクラスタが追加された場合に起こると考えられる．多数のエンドホストを有するクラスタが接続されると，そのクラスタ付近のエッジがネットワークの中心であると Betweenness Centrality によって計算されてしまう．このようになるとグラフ分割のバランスが崩れ，効率的な並列測定を行うことができなくなってしまう．この実験ではエンドホストの追加をシステム上からランダムに行うのではなく，クラスタを一単位としてノード数の増減を行っていた．さらにこのデータでは構成ホスト数の少ないクラスタから選択的に追加していくということを行ったので，この問題が起こってしまった可能性が高い．このようなことが起こらないために，単に Betweenness Centrality による分割だけでなく，他の指標やヒューリスティクスなどを用いてグラフ分割をバランスよく行うことが必要であると考えられる．

4.3 精度

本手法の精度の定量的な評価として，測定結果のバンド幅の値と正解の値がどれほど合致しているかを調べた．本手法のバンド幅マップの精度の比較対象として，TopoMon の手順を模倣して実装されたバンド幅マップ構築手法で得られたものを取り上げる．実環境では，我々の管理外のネットワークなどの，正解の値が不明なエッジが多いため，ランダムグラフを生成してそれに対して本手法を適用するシミュレーションを行った．ここでのランダムグラフとは，ノード数とその接続形態，そして各エッジのバンド幅の値をランダムに生成したものである．ランダムグラフを生成する際にグラフの直径をパラメータとして，直径が 2, 4, 6, 8, 10 の五つの仮想ネットワークに対して本手法と TopoMon-based な手法を実行し，それぞれが構築したバンド幅マップの各エッジについて正解の値と比較し，精度を求めた．正解とする仮想ネットワークのバンド幅の値は 10.0 から 1000.0 の範囲のランダムな実数が振られ，ノード数はそれぞれ 64, 87, 147, 228, 498 であった．検証結果を図 4.5 に示す．横軸は直径の異なる五つのシミュレーション環境であり，縦軸は正解率である．正解率は，ネットワークグラフに含まれるエッジの総数に対する，測定結果のバンド幅の値が正解のものと同値であったエッジの本数の比率で求めている．

我々の手法ではネットワークの規模が大きくなっても精度を落とさないで結果が得られていることが分かる．これは，中間エッジであっても複数のネットワークストリームによって飽和させてバンド幅測定ができているためである．ただし，3.2.1 節でも述べた通り，ネットワーク中にはシステム全体を動員してでも飽和しきらないエッジが存在する可能性があるので，それを本手法では最大可用帯域として報告している分，精度が落ちてきていることが見て取れる．しかしこの実験で最も大きな直径 10 のネットワークに対しても 98%の正解率を得ることができた．

一方 TopoMon-based の手法は，直径 4 のグラフにおいて既に一割程度の本数のエッジのバンド幅を誤って報告してしまうということが分かる．最終的に，TopoMon-based の手法の正解率は 7 割

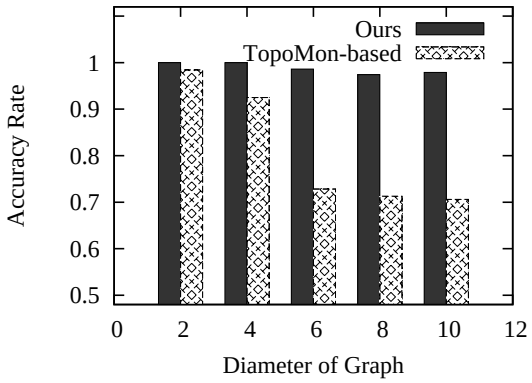


図 4.5: バンド幅マップの精度

程度に落ち，我々の手法と比較して測定精度の低い手法だということが分かる．TopoMon の測定結果が誤る原因は，TopoMon が一対一のバンド幅測定，すなわちネットワークストリームを一つしか用いない測定を行っているためである．一つのネットワークストリームでは飽和しない中間エッジがネットワーク中に存在した場合，確実に測定は誤る．ネットワークの直径が大きくなるにつれて中間エッジの数は増えるので，大きな環境に対して一対一のバンド幅測定のみを用いたのではバンド幅マップ構築は不可能である．ランダムグラフの直径が大きくなるにつれて TopoMon-based の正解率が低くなっていることも図から見て取れる．

第5章　結論

5.1　まとめ

本論文では効率的なバンド幅マップ構築アルゴリズムを提案した．実験によりそれが既存手法に比べて高速であること，正確であること，また大規模な分散環境にも適用することのできるスケーラビリティを持つことが確かめられた．実験の結果，WAN をまたぐ 15 クラスタ 311 ホスト，640 エッジの環境のバンド幅マップを 50 秒程度で構築することができた．

我々の手法の主な特徴は，中間エッジのバンド幅測定とグラフ分割による並列化である．既存のバンド幅測定は基本的に一対一のエンドホストペアで行われるため，ネットワークトポロジーの中央付近に位置する中間エッジのバンド幅を測定することはできなかった．我々の手法では多対多のエンドホストペアでネットワークストリームを一斉に発生させることで，中間エッジのバンド幅を最大可用帯域まで測定することを可能にした．また手法自体の高速化とスケーラビリティを持たせるために，測定の並列化を行った．並列化に際してエッジの Betweenness Centrality を分割の指標として用いることで，個々のバンド幅測定を誤ることなく効率的な並列化を実現することができた．

構築したバンド幅マップを利用するアプリケーションとして様々なものが考えられる．例えばバンド幅の値を考慮して行う MPI の適応的な集合通信といったものから，巨大ストリームデータ配信やファイルレプリカ配置の仕方などがある．それらのような並列分散処理における基本的な操作から応用的なアプリケーションといった幅広い分野において，実行速度を上げたり頑健性や利便性を高めたりするための指標となるものがバンド幅マップである．

5.2　課題

今後の課題を挙げる．まずバンド幅測定自体について，本研究では計測時間を経験的な数字で 300 ミリ秒と決め打っていた．これは同一クラスタ内などといった近いエンドホスト同士では十分すぎる値である．これを入力トポロジーの形状やプログラム実行中の測定値具合から動的に設定することによって全体の実行時間を短くすることができると考えている．次にグラフ分割による並列化について，本研究で分割の指標とした Betweenness Centrality は一つの実現例だということを忘れてはならない．さらに効率的な並列測定が可能になる分割方法が見いだされる可能性は十分にある．また，本手法では分割されたサブグラフに対してマスターノードは一つのままであった．今後さらに

分散環境が普及して規模が拡大していくと，メッセージと負荷が集中して一つのマスターでは処理が滞ってしまう可能性がある．これには，分割されたサブグラフ内から代表者を決め，そのサブグラフ内の一時的なマスターとして動かすなどの対応が必要である．

参考文献

[1] Intrigger, a distributed platform for information technology research. https://www.intrigger.jp/.

[2] Netperf. http://www.netperf.org/netperf/.

[3] perfsonar. http://www.perfsonar.net/.

[4] Jeff Boote and Aaron Brown. Bwctl, bandwidth test controller. http://www.internet2.edu/performance/bwctl/.

[5] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, Vol. 25, pp. 163–177, 2001.

[6] Mathijs den Burger, Thilo Kielmann, and Henri E. Bal. Topomon: A monitoring tool for grid network topology. *In ICCS '02: Proceedings of the International Conference on Computational Science-Part 2*, pp. 558–567, 2002.

[7] Constantinos Dovorolis, Parameswaran Ramanathan, and David Moore. Packet dispersion techniques and capacity estimation. http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/pathrate.html.

[8] Constantinos Dovorolis, Parameswaran Ramanathan, and David Moore. What do packets dispersion techniques measure? *IEEE INFOCOM*, pp. 905–914, 2001. http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/pathrate.html.

[9] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, Vol. 40, No. 1, pp. 35–41, 1977.

[10] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Networkx. http://networkx.lanl.gov/.

[11] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. pp. 11–15, August 2008.

[12] Van Jacobson. pathchar - a tool to infer characteristics of internet paths, April 1997.

[13] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. Magpie: Mpi's collective communication operations for clustered wide area systems. *Symposium on Principles and Practice of Parallel Programming*, pp. 131–140, May 1999.

[14] Kevin Lai and Mary Baker. Nettimer: A tool for measuring bottleneck link bandwidth. *Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems*, Vol. 3, pp. 1–5, January 2001.

[15] Bruce A. Mah. Pchar: A tool for measuring internet path characteristics, July 1999. http://www.kitchenlab.org/www/bmah/Software/pchar/.

[16] Sho Naganuma, Kei Takahashi, Hideo Saito, Tkeshi Shibata, Kenjiro Taura, and Takashi Chikayama. Improving efficiency of network bandwidth estimation using topology information. *Symposium on Advanced Computing Systems and Infrastructures (SACSIS)*, pp. 359–366, June 2008.

[17] Stanislav Shalunov. thrulay: Network capacity and delay tester. http://shlang.com/thrulay/.

[18] Tatsuya Shirai, Hideo Saito, and Kenjiro Taura. A fast topology inference — a building block for network-aware parallel computing. *In Proceedings of the 16th IEEE International Symposium HPDC 2007*, pp. 11–21, June 2007.

[19] Quinn O. Snell, Armin R. Mikler, and John L. Gustafson. Netpipe: A network protocol independent performance evaluator. *IASTED International Conference on Inteligent Management and Systems*, June 1996.

[20] Kei Takahashi, Hideo Saito, Takeshi Shibata, and Kenjiro Taura. A stable broadcast algorithm. *to appear the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid2008)*, March 2008.

[21] Kenjiro Taura. Gxp: An interactive shell for the grid environment. *International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems.* http://www.logos.ic.i.u-tokyo.ac.jp/gxp/.

[22] Ajay Tirumala, Feng Qin, Jon Dugan, and Jim Ferguson. Iperf. http://www.dast.nlanr.net/projects/Iperf/.

[23] Dave Turner and Xuehua Chen. Protocol-dependent message-passing performance on linux clusters. *Cluster 2002 conference in Chicago*, September 2002. http://www.scl.ameslab.gov/Projects/NetPIPE/.

[24] Dave Turner, Adam Oline, Xuehua Chen, and Troy Benjegerdes. Integrating new capabilities into netpipe. *Euro PVM/MPI conference in Venice Italy*, September 2003. http://www.scl.ameslab.gov/Projects/NetPIPE/.

[25] Rich Wolski. Dynamically forecasting network performance using the network weather service. *In Proceedings of the 6th High-Performance Distributed Computing Conference (Aug. 1997)*, pp. 316–325, August 1997. http://nws.cs.ucsb.edu/ewiki/.

[26] Rich Wolski, Neil T. Spring, and Jim Hayes. Implementing a performance forecasting system for metacomputing: the network weather service. *Proceedings of the 1997 ACM/IEEE conference on Supercomputing*, pp. 1–19, 1997.

[27] Takahiro Yagi, Shigeo Shioda, and Kenichi Mase. A measurement methodology for end-to-end bottleneck-link bandwidth. *IEICE Transactions*, Vol. J87-B, No. 10, pp. 1636–1647, October 2004.

[28] Shota Yoshitomi, Ken Hironaka, and Kenjiro Taura. An adaptive gather algorithm avoiding contention. *Symposium on Advanced Computing Systems and Infrastructures (SACSIS)*, pp. 71–78, May 2009.

発表文献

1. 長沼 翔, 田浦 健次朗. 大規模ネットワークにおける効率的なバンド幅マップ構築アルゴリズム. 情報処理学会第 72 回全国大会, 東京, 2010 年 3 月.

2. 長沼 翔, 田浦 健次朗. 大規模分散環境におけるバンド幅測定アルゴリズム. 情報処理学会研究報告 HPC-121 (SWoPP2009), 仙台, 2009 年 8 月.

3. 長沼 翔, 高橋 慧, 斎藤 秀雄, 柴田 剛志, 田浦 健次朗, 近山 隆. ネットワークトポロジーを考慮した効率的なバンド幅推定手法. 情報処理学会研究報告 HPC-116 (SWoPP2008), pp.175-180, 佐賀, 2008 年 8 月.

4. 長沼 翔, 高橋 慧, 斎藤 秀雄, 柴田 剛志, 田浦 健次朗, 近山 隆. ネットワークトポロジを考慮した効率的なバンド幅推定手法. 先進的計算基盤システムシンポジウム (SACSIS2008), pp.359-366, 筑波, 2008 年 6 月.

5. 長沼 翔, 高橋 慧, 柴田 剛志, 田浦 健次朗, 近山 隆. ネットワークトポロジを考慮したバンド幅推定の高速化手法. 情報処理学会第 70 回全国大会, 筑波, 2008 年 3 月.

謝辞

近山隆教授ならびに田浦健次朗准教授には，私が学部生の頃から長らくお世話をしていただきました．学部の卒業論文で私がやった研究をそれで終わりとせず，さらに上を目指すよう再び同様の研究テーマを任せていただき，一般的なことからより深い内容までご指導していただきました．本研究が進められたことだけでなく，そこでの数々の新しい課題や問題の見つけ方，そしてこの分野の深い知識など，非常に多くのことを得ることができました．

特任助教の横山大作さん，鴨志田良和さんをはじめ研究室の皆様にはテクニカルなアドバイスや本研究のアルゴリズムに関するご助言を沢山いただきました．柴田剛志さんには私の学部の卒業研究の内容から本研究の内容まで，長い間見て頂きました．博士課程卒業の斎藤秀雄さんならびに修士課程卒業の弘中健さんは主にプログラムや OS などの非常に深い技術的なお話をしていただきました．斎藤さんには電気系サーバの TA でも大変お世話になり，また生活面でも色々と面倒を見ていただきました．

同期の方々にも感謝しております．私が少し飽きた時に，特に何も用事は無いのに隣の部屋へ行くとすぐそこにいて，他愛も無い話で相手をしてくれた栗田光晴君，ご迷惑をおかけしました．三木理斗君の麻雀プレイヤの研究は個人的にとても成果が楽しみで，よく三木君のディスプレイの前に遊びに行きました．他にも研究以外のこと，何かの締め切りや就職活動のことなど，何か分からないことがあればすぐに近くにいる同期に相談できる環境がありました．皆さんのおかげで楽しい研究生活が送れました．感謝します．

ここには書ききれませんが，本当に多くの方のおかげで本研究を進められたと思っています．皆様本当にありがとうございました．