Master Thesis

2014 年度　修士論文

Multipath TCP による経路切り替え手法を用いたデータセンターネットワークにおけるショートフロー完結時間の改善

Improving the flow completion time of short flows in datacenter network with path switching by Multipath TCP

平成 27 年 2 月 5 日 提出

東京大学大学院
工学系研究科 電気系工学専攻
37-136482 藤居 翔吾

指導教員 関谷 勇司

The University of Tokyo
Graduate School of Engineering, Department of Electrical Engineering and Information Systems

Shogo Fujii

要旨

クラウド型のサービスの性質により, 今日のデータセンターではデータセンター内のトラフィックが増大しており, 複数の経路を持つネットワーク環境を応用して, 通信性能向上を目指す取り組みがされている. しかし, 様々なニーズを抱えるトラフィックが混在している中で, レイテンシ志向なサイズの小さいフローに対し, 既存の Multipath TCP(MPTCP) 実装では性能を劣化させる問題が報告されている. そこで本研究では, この問題に対し, 並列分散処理アプリケーションが稼働しているクラスター PC のトラフィックを観測する事で, 単一 NIC(Network Interface Card) への通信集中によるキュー負荷の影響があることを検討し, サイズの大きいスループット指向なロングフローとサイズの小さいレイテンシ指向なショートフローが一つのリンクに混在して通信を行っていることを明らかにし, それによる通信性能障害が生じる技術的背景を考察した. そして, フローの指向毎に異なる通信経路を利用するレーンモデルを提案した. また, OS レベルから一つの通信を見た際に, そのフローの指向をフロー持続時間を用いて分類し, 経路の状況に合わせた経路切替手法を提案した. 提案手法を Linux OS に実装し, シミュレーションを用いた評価を行い, ロングフローとショートフローが混在する環境下で, それぞれの指向毎に経路を切り替える制御を行うことで, ショートフロー通信の短縮化ならびに, ロングフローの通信性能を改善することを示した.

Abstract

As increasing the amount of traffic in a datacenter by cloud service, the effective network for utilization of massive computer clusters has been studied. Recently, Multipath TCP(MPTCP) has been tackled this problem. MPTCP can achieve the effective consumptions of the resources with multipath, but a researcher reported MPTCP causes the delay of flow completion for short flows in such a multipath network environment. In this paper, I presented measurements of the distribute processing cluster PCs, and revealed the impairment mechanisms that lead to that latencies, rooted in single NIC(Network Interface Card) with intensive load traffic. I also show there are both latency-oriented short flow and throughput-oriented long flow in a datacenter network. As the result, they lead to the performance impairment in flow completion time. In order to solve this problem, I proposed datacenter lane model to utilize different paths for each requirements of the flows, and developed the method of flow-classification for the orient and path-switching scheme to react the path condition with flow duration time, as approach just for end-node. I implemented these proposals to the end-node Linux OS, and evaluated by simulation experiment. In my experiment, my proposals improve the performance for not only short flows in shortening the FCT but also long flows in improving the throughput with the control to separate the mixture traffic into both latency-oriented flow and throughput-oriented one.

目次

図目次

表目次

第 1 章

序論

1.1 研究背景

今日の一般家庭のインターネット接続環境が, ギガビット級の速度に達しようとしている中, 多様な端末がインターネットに接続できるようになり, 大量かつ多種多様なデータの取得が可能となった. 特にトラフィックデータ量の増加傾向は顕著で, 18~24 ヶ月単位で総データ容量が 2 倍になるという予測がされている [1]. また Facebook では, 300 ペタバイト以上のデータ量を保有しており, 1 日あたりに 1 ペタバイトのデータを解析している [5]. 近年では, ビッグデータの活用が着目され, 例えばウェブ検索エンジン, SNS(Social Networking Service) などのデータセンターを用いたクラウド型サービスにおいて, リアルタイムに近いレスポンスを返すような場面で使われ始めている. そのようなクラウドサービスには近年, より高いユーザーエクスペリエンスが要求されており, Amazon では 100[ms] の遅延により売り上げが 1% 下がる, といった報告 [2] があるように, 例えば e コマースサイトでの商品購入や, インターネット広告のコンバージョンのようなユーザの意思決定へのレスポンス遅延の影響は深刻な問題である [3]. そのため, 大規模データをより高速に処理することが求められており, データセンターではサーバ運用台数が増加の一途を辿っている. そうした中で, 可用性, 計算性能, 低コストの三つの要件がデータセンターの抱える課題となっている [4]. 特に計算性能について, 大量の計算機資源から最大限の性能を引き出すためには, 従来の仕組みではデータセンター内トラフィックに対して一部の資源にトラフィックが集中する問題に対応できないため, 計算機資源を有効活用するための研究が盛んに行われている [6–11, 24]. そのようなスケーラビリティ拡大には, ネットワークトポロジー, アプリケーション, プロトコルに対する三つのアプローチがある.

ネットワークトポロジーを改良するアプローチでは, 近年データセンター内の通信帯域が 1Gbps から 10Gbps, 40Gbps, 100Gbps へと拡大していく中で, 従来の単純な二分木階層構造では, データセンター内で発生するトラフィックに対して通信帯域を最大限割り当てることができない [7]. そのため, 近年ではデータセンター内のトラフィックに対してスイッチを二段, 三段と少ない段数で構成し, エンドノードから接続される Edge スイッチとそれらを集約する

ための Aggregation スイッチをより多く用いて接続することで, 垂直方向には低く, 鉛直方向には大きく広がるトポロジーを提案している. これによって, データセンター内のトラフィックに対して通信帯域を有効利用することができる. また, データセンター内の通信においては, ノード間の通信経路が複数設けられており, 仮に一つの通信経路が切断されてしまったとしても, 残りの経路を用いて通信することができる, 可用性を持ち合わせたマルチパス環境を実現している [7]. さらに, 近年のデータセントートポロジーの動向として, 通信機器のコモディティ化があり, 特殊なデバイスや高価な機器を用いずに, 安価で汎用性のある機器を複数利用することで, 同等の性能を出そうと, 低コスト化を目指している [8].

大規模データの処理速度を改良するアプローチでは, データセンターの抱える巨大なデータの有効活用のために, 並列分散処理技術を用いることが一般的である. 並列分散処理技術によって, 一つの処理を複数のサーバで同時並列で処理をすることで, 高スループットなデータ処理を実現しており, 大量の計算機資源から最大限の性能を引き出している. 並列分散処理技術では partition-aggregate 計算モデルによって, 処理を細分化し, 各サーバに処理を割り当て, 結果を集約していき, 処理を完結させていく. このような並列分散処理システムのアーキテクチャ上, 細分化された処理をそれぞれの処理サーバに割り当てる際, また計算結果を集約していく際に, サイズの小さいデータの通信が大量に発生し, データセンターネットワークの性能改善のボトルネックとなっている. MapReduce [6] 等の並列分散処理フレームワークは, これに従っており, 今日の大規模クラウドサービスにおいて必要不可欠である.

プロトコルを改良するアプローチでは, 従来の TCP を拡張した Multipath TCP (MPTCP) [13] をデータセンターネットワークに用いる提案がされている [7]. MPTCP を用いることにより, OS の制御によって複数の NIC(Network Interface Card), 複数の経路を同時に利用し, スループットを向上させることが期待されている. また, 近年のマルチパス環境を持つトポロジーにおいて, 通信経路が切断された中でも, 一つのデータ通信の中で他の経路を使って途切れることなくデータ転送できる, 耐障害性を実現している. MPTCP では, 従来の TCP と同様に, 3 ウェイ・ハンドシェイクでコネクションを確立させ, その直後に互いの持っている他の IP アドレスを交換する. その後, サブフロー[*1]を形成し, 複数の経路での通信を行う. しかし MPTCP の実装上, 比較的サイズの小さい通信では, サブフローを形成する前に通信が完了し, 結果的に一つの経路でのみ通信が行われることとなる.

クラウド型サービスを実現するデータセンターネットワークにおいて, 並列分散処理アプリケーションによって大量に生成されたショートフロー通信が改善の課題となっている [9]. 実際, データセンター内でのトラフィックの内, 約 80% がショートフローであり [14], 並列分散処理の構造上, ショートフロー通信時間が大規模計算処理高速化のための極めて重要な要素となっている. しかし, 既存の TCP を用いたクラウド環境では, このショートフロー[*2]通信の一部が大きく遅延する問題が報告されており, 既存のデータセンターネットワークを改善する上

[*1] 複数の TCP コネクションの内, ある一つのコネクションにおけるフロー
[*2] サイズの小さいデータ通信. 本研究では基本的に 100KB 以下のデータ通信を指す

で, ショートフローの遅延の問題は深刻であると言える [9,20].

1.2　研究目的

データセンターにおけるマルチパス環境でのショートフロー遅延の問題は並列分散処理性能の面で深刻な問題であり, 本研究は, 低レイテンシなネットワークによるコンスタントに性能の出せるデータセンターの実現を目指す. そのために二つの手法を提案する.

一つ目は, 異なる通信目的によって経路を切り替えるための通信レーンによるデータセンターモデルを提案する. これによって, 通信時間の短いフローについては, 輻輳の少ない良好な通信環境が保たれているショートフローレーンで通信することができ,　遅延の影響が小さく通信を終えることができる. 比較的長い時間通信を行うフローであれば, ショートフローレーンからロングフローレーンへと通信経路を切り替え, 通信を行う. このモデルによって, ショートフローの遅延を抑え, ロングフロー[*3]. はショートフロー通信に干渉しない制御が実現できる.

二つ目は, 経路状況に合わせた通信経路切替手法を提案する. この手法によって, データサイズの小さいフローについては, ショートフローレーンを用いて遅延なく通信を完了させる. また各経路について, 各経路の通信状況を表すメトリックを計算することで, そのメトリックを基に通信経路を切り替える制御を実現する. この手法によって, フローの指向を区別してそのレーン毎に通信を行い, 経路状況に合わせた経路切替が可能となる.

これらの手法の実装には, MPTCP を応用して実現する. これにより, OS のみの変更で, 既存のアプリケーションに対して変更なく, 特殊な実装, 機器を必要としないため, 実環境へ適用が容易である.

1.3　本研究の貢献

本研究では, 提案手法を設計する指針となった取り組みについて, 二つの成果がある.

一つは, MPTCP と TCP が互いに混在するような通信環境において, MPTCP が TCP 通信に対して性能劣化を引き起こすという点である. 主に通信経路の輻輳により, 通信開始時にパケットロスが生じることで遅延が生じることが分かり, 輻輳している経路を避け, 適切な経路を選択するためには, コネクションを接続する時点で行う必要があることを示した.

二つ目は, 実際の並列分散処理アプリケーションがどのようなトラフィックを生成するのか解析を行った. この解析によって, 並列分散処理アプリケーション特有のトラフィックを明確にし, それによって汎用的なネットワーク機器を用いたデータセンターにおけるショートフロー遅延が生じる技術的背景について, ボトルネックとなりうるネットワーク環境を検討し, その原因を示した.

本研究の成果として, 提案したネットワークレーンモデルと通信経路切り替え手法により,

[*3] サイズの大きいデータ通信

ショートフロー通信における遅延を抑制を達成した．また既存のデータセンター内通信では，サイズの大きいフローの影響を受け，通信時間の短いショートフローに遅延が生じていた問題に対し，本提案手法によって，平均フロー完結時間 FCT(Flow Completion Time) と特に遅延していた下位 5% のフローに対して，通信時間の短縮化を実現できた．さらに，本研究の提案手法では，MPTCP を応用することによる OS ベースの手法であるため，OS を書き換えるだけで利用することができるという実現可能性が高い手法としての利点がある．

1.4　本研究の構成

本論文の構成は以下の通りである．第 2 章では，低レイテンシなデータセンターネットワークに関する研究について，アプローチする場所ごとに述べ，データセンターネットワークが抱えている要求案件と 既存研究手法の課題を議論する．第 3 章では，データセンターが構成する技術要素と抱えている技術的課題を述べ，提案手法の位置付けを示す．第 4 章では，既存技術の抱える課題やこれまで報告された問題点を，再現実験により改めて見直すことにより，提案手法の設計する上での指針を示す．第 5 章では，複数のレーンを用いたデータセンターモデルと MPTCP を応用した通信経路切り替えアルゴリズムを併用した本研究のショートフロー改善手法を提案し，既存研究との差異を明確にすることで，本研究の位置付けを明らかにする．第 6 章では，実際のデータセンターネットワークを想定したトラフィックに対して，提案手法と既存手法を比較，また提案手法の性能評価実験の方法と結果について述べる．第 7 章では，実験の考察と実験結果をもとにした提案手法の応用に関する考察を述べる．最後に，第 8 章で，まとめと今後の課題について述べる．

第 2 章

関連研究

本章では, これまでに報告されている複数経路利用によるフロー完結時間短縮化技術について簡潔に述べ, その優位性や問題点を示す. 大量のネットワーク機器により構成されており, 短時間で大量のデータの処理を行うデータセンターネットワーク環境において, リクエストに対するレスポンスの高速化によるユーザエクスペリエンスのさらなる向上が求められており, 低レイテンシなネットワークが必要とされている. 本章では, これまでに報告されているデータセンターにおける低レイテンシなネットワークに関する研究について述べる. 既存の研究では, キュー長の縮小, 再送処理の高速化, ショートフローの優先付け, 複数経路の利用の 4 つの技術を用いることで実現しており, 4 つの分類ごとにその特徴や課題について考察する. そして, 本研究の位置付けを示す.

2.1 キュー長の縮小

レイテンシの大きいデータセンターネットワークの主な原因はキューイング遅延である [8]. 報告された解析結果 [8] によると, ネットワーク全体のトラフィック量が比較的小さく, 大きな輻輳が発生していない環境においても, ショートフローの FCT の大部分がキューイング遅延に依存する. キューイング遅延を減らし, FCT を改善するための一つの手法として, キュー長の縮小がある.

スイッチのキュー長を減らし, バッファの占有を抑えることにより, 遅延を改善するというアプローチは最も直接的な手法である. 高速にパケット処理のできない汎用的なスイッチではバースト性のあるトラフィックに対して, バッファを多く用意することで対応している. データセンターネットワークでは基本的に帯域遅延積が小さく, バッファが大きいと通信性能に影響が出る. 既存の TCP の輻輳制御では, ロングフローがバッファを使い切るまで通信しようとし, それにより大きなキューイング遅延が生まれ, ショートフローの通信性能に影響が出る. そのため, キューイング遅延を抑えるために, スイッチのバッファを小さくし, 新しい帯域制御手法や輻輳検出のための仕組みがエンドノードとスイッチに対して必要となる.

DCTCP [8] と HULL [56] はスイッチバッファの占有率を持続的に低く保ち, 遅延を抑える

Table.2.1　An overview of low latency datacenter networking proposals

Categories		Proposals	Objectives		Modifications to		
			FCT	deadline	TCP	Switches	applications
Redcing queue length		DCTCP	mean	×	√	×	×
		HULL	mean	×	√	√	×
Prioritizing flows basd on	Flow deadlines	D^3	none	√	√	√	×
		PDQ	none	√	√	√	×
		D^2TCP	tail	√	√	×	×
		MCP	tail	√	√	×	×
	Application assignment	pFablic	mean & tail	√	√	√	√
		Detail	tail	√	√	√	√
Exploiting multipath		RepFlow	mean & tail	×	×	×	√
Accelerating retransmissions		DIBS	tail	×	√	√	×
		FastLane	tail	×	√	√	×
		CP	tail	×	√	√	×

為に提案された手法である.

2.1.1　DCTCP

DCTCP(Data Center TCP) [8] はデータセンターネットワーク内の遅延問題の解決を目指した初めての取り組みである. Alizadeh ら [8] は実際に稼働しているデータセンターのトラフィックを観測, 1 ヶ月分のトラフィックデータの解析を行い, データセンターネットワークが生成するトラフィックの特徴を考察した. その結果, データセンターのトラフィックの大部分がレイテンシ指向なショートフローであることを示し, ロングフローによってバッファが占められた中継されるスイッチの影響でショートフローが大きく遅延することを示した. これらの観測結果が動機となり, 著者たちはスループット性能を大きく損なうことなく, バッファの占有を抑える為の輻輳手法として DCTCP という新しいトランスポートプロトコルを提案した.

DCTCP は汎用的なスイッチにも広く実装されている Explicit Congestion Notification(ECN) による輻輳通知の機能を用いている. バッファ占有のしきい値をあらかじめ設定しておき, それを超えると, スイッチがパケットの TCP ヘッダーの ECN エコーフラグを用いてエンドノードに明示的に通知する. この情報を用いてサーバは TCP 遅延 ACK に ECN エコーフラグを加えクライアントホストに知らせる. そしてクライアントでは, ECN マークされたパケットの割合によって, スイッチバッファの状態を推定し, ウィンドサイズを抑える. その結果, キュー溢れが起こる前に輻輳を回避することができ, バッファ占有を抑えることができる. DCTCP は既存の TCP 実装に対して, 30 行の変更を加えるだけで利用することができ, 今のインフラに対して容易に展開することができる. DCTCP はテストベッド環境での評価において, FCT を大きく改善することができ, 特にテール部分について, 99 パーセンタイル FCT

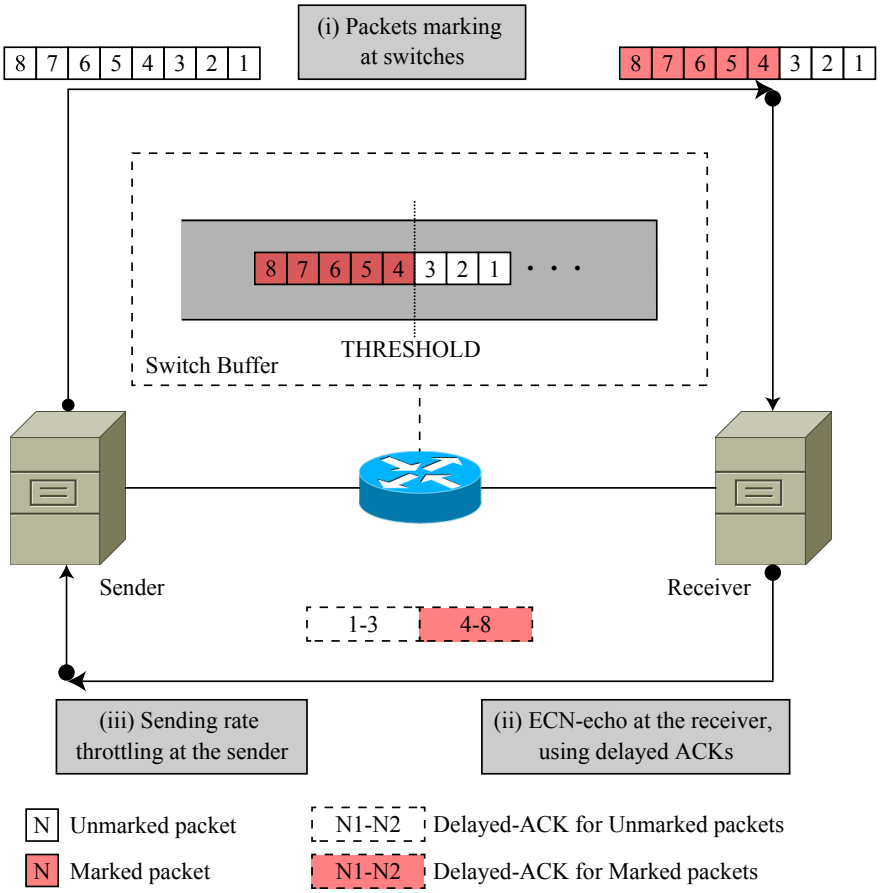

Fig.2.1　The control loop in DCTCP

を TCP の 40% 以上短縮化を達成した.

2.1.2　HULL

HULL(High-bandwidth Ultra-Low Latency) [56] は DCTCP 実装をベースに追加にした手法であり, さらなるキュー長の縮小のためのすべてのポートのキュー状態の推定を目的としている. HULL は直接的にキュー長の様子を通知する DCTCP とは異なり, 通信利用帯域によるリンク利用率を基にした輻輳通知手法である.

HULL ではリンク利用率を用いたファントムキューという仮想キューを利用している. これは, バッファ処理自身のオーバーヘッド影響を考慮せず, 実際の物理リンクよりも低い通信レートでの仮想リンクにおけるキューイングの様子をエミュレートすることによって算出される. ファントムキューがしきい値を上回った場合, DCTCP と同じ ECN を用いた通知機構を用いて, ウィンドウサイズの調整を行いキュー長を抑える. ファントムキューは物理リンクよりも低いレートにおいてシミュレートされているため, よりレートの高い実際の環境におけるバッファではより小さく保つことができ, キューイング遅延の影響がより小さくなる. 著者は

平均値, 99 パーセンタイル値の FCT 共に DCTCP の 10 倍以上, TCP の 40 倍以上改善することができたことを示した. しかし FCT の改善のトレードオフとして約 10% の利用帯域の減少も生じていることも示した.

2.2　再送の高速化

データセンターネットワークの遅延を抑える為の手法として, パケットロスによる遅延に対して改善を行う手法がある. 通常 TCP では, パケットロスが起こった場合, それを検知するためのタイムアウト時間 (RTO) 後に再送制御を行う. 一般に RTO はエンドノード間のラウンドトリップ時間 (RTT) よりも非常に大きな値を設定するため, RTO によって不必要な時間分だけ待たなければならない. これまでに提案された手法では, RTO を小さくする, あるいは RTO による再送制御を無視し, 明示的にエンドノードに対してパケットロスが起こったことを通知することで, 再送遅延の短縮化を実現している.

ショートフローとロングフローの両方が共存している場合, パケットロス, 再送制御が起こることは避けられれない. パケットロス自体は頻繁に起こるわけではなく, その割合も小さいため, 再送制御による遅延を改善することはテール部分の遅延の改善に対して大きく貢献する. TCP では, 基本的にはショートフローの FCT は RTO よりも短いため, パケットロスが起きた場合の影響は非常に大きく, 例えばアプリケーション性能に依存するショートフローにとって, その FCT がデッドラインを満たすためには RTO は大きすぎる. また TCP の輻輳制御では, パケットロスが起こった際に, ウィンドウサイズが減少され, フローを完結するためにより多くのラウンドトリップが必要となる.

先行研究では, この問題を解決するために, 二つの観点からアプローチしている. DIBS [57] ではパケットロスが起こった際, その混雑しているポートとは異なる他のポートをランダムに選択, パケットをリダイレクトし再送制御を行う. また, FastLane [58], CP(cutting Payload) [59] では高速に明示的な通知をエンドノードにおこうことで, 再送制御を高速化する.

2.2.1　DIBS

Detour-Induced Buffer Sharing [57] では, アイドル状態のネットワーク資源を利用し, ホットスポットでのバーストトラフィックを緩和させる. スイッチがパケットロスをしなければならない時, 本来転送しなければならないポートとは異なる他のポートをランダムに選択し, パケットを転送する. これにより 3 つの成果を生み出すことができる.

1. 他のパスに転送されることでより多くのホップ数を必要とする可能性があるが, 輻輳が起こっているホットスポットを避けることができる.
2. 転送したパケットがループバックしてきた際に, 本来転送するべきポートのバッファが占有されていないパケットロスが起こらない状態であればそちらに転送する.

3. そのパケットの TTL に達するまでに, アイドル状態のパスを見つけられなかった際には, パケットは破棄される.

複数の等価コストのパスをもつデータセンターネットワークの性質上, 迂回経路によって大部分のパケットロスを抑えることができる. また, アイドル状態のパスを見つけられない可能性はデータセンターでは低いが, その際にはタイムアウトや再送制御が行われる.

2.2.2 FastLane, CP

FastLane [58] と CP [59] は再送制御の高速化のための明示的なパケットロスの通知を生成する. それにより, 再送制御を起動させるまでのオーバヘッドを RTO 時間から 1 回分のラウンドトリップへと減少させる.

FastLane [58] は, パケットロスを引き起こしたスイッチで検出し, 直ちにエンドノードへ通知を行う. CPU のオーバーヘッドを抑えるために, ロスを引き起こしたパケットの IP アドレスをスワップし, エンドノードへ通知する. これにより, エンドノードはどのフローでパケットロスが発生したかを素早く知ることができる. また, 通知するためのトラフィックに対し, 通信帯域の制限 (bandwidth cap) をすることで, ネットワーク全体の帯域オーバーヘッドを抑えることができる. 一方, CP [59] は通知のために新しいパケットヘッダーを用いる. パケットロスが起こった時に, ペイロードの大きいパケットについてはペイロード部分を取り除く. 通知を受けたノードは, ヘッダーを見てペイロードが除去されていることを認識し, SACK のようなピンポイントでの ACK によって再送制御を呼び出す. FastLane も CP もどのトランスポートプロトコルと互換性を持つ手法である.

2.3 ショートフローへの優先付け

ショートフローの FCT を改善するための手法として, 優先制御がある. どのパケットも同じ扱いをする TCP とは異なり, ショートフローに対しては優先度を高く設定し, 他のキューイングされているパケットよりも前に処理されるような優先制御が行われ, それによりショートフローの FCT が大きく改善される. フローの優先付けには二つの方法がある. 一つは, アプリケーションによって明示的に優先度を定められ, 優先度を割り当てられる手法 [10, 11]. もう一方は, 今日のデータセンターの扱うアプリケーションが暗黙的に定めるデッドライン情報を用いて優先度を割り当てる手法である. いくつかのアプリケーションでは各フローに対しておよそ 200ms∼300ms のデッドラインを満たす要求があり, 優先制御によりそれを実現し, また, そうしたデッドラインが明示的に定められないものについては, レスポンス時間を保証するために用いられる [40, 41, 54, 60]. そのため, ネットワーク内のスケジューリングによるフローの優先付けは既存のトランスポート層, データリンク層のプロトコルを改良することで効果的な手法が実現される.

一般的に, インターネットにおける通信の公平性では, ロングフローであるかショートフロー

であるかに依らず, どのパケットも公平に扱おうとし, 多くの手法がそれに従っている. しかし, データセンターネットワークでは, ロングフローによるショートフロー通信性能の劣化を引き起こす中で, こうした考えは, 不公平であるという議論もある [40, 41, 54, 60]. そうした不公平性の問題を解決するために優先制御手法が提案されているが, 大きく 2 種類の情報を使う手法に分類することができる. 一つは, フローの FCT のデッドラインによって優先度を決める手法. 他方はアプリケーションによって明示的な優先度を設定される手法がある.

2.3.1　デッドラインベースの優先付け

D^3

D^3 (Deadline Driven Delivery) [40] はデッドラインベースの優先制御手法である. D^3 では, フローのデッドラインを満たすために必要な帯域を割り当てることによってネットワークリソースの最適な配分ができるという考えに基づいている.

Wilson ら [40] は, サーバとクライアントでインタラクティブな通信を必要とする多くのアプリケーションは, そのアプリケーションフローを生成する処理ノードによってデッドラインが定められていると述べており, このデッドライン情報を用いることで, D^3 はデッドラインまでに通信を完了できる最小のレートを算出し, パケットのヘッダー情報として追加する. それにより, フローは中継するすべてのスイッチに対して必要な帯域を要求することができる. スイッチは ACK パケットによってそのリクエストに対する予約帯域をエンドノードに通知するそしてクライアントノードはその通知に従って送信レートを調整する.

D^3 が適用されるスイッチでは, 計算, メモリー容量を抑えるため, スイッチによる帯域割り当ては, FCFS(first-come-first-serve) に基づいて実行される. そのため, すべてのフローに対して要求を満たすための帯域が不十分な状況であれば, 先着順で初めのいくつかのフローしか満たせない可能性もある.

PDQ

D^3 の FCFS スケジューリングは十分なパフォーマンスを引き出せない場合がある. 十分にパフォーマンスが引き出せない状況について, Fig.2.2 に示す. こうした公平性の考えが PDQ(Preemptive Distributed Quick) フロースケジューリングの設計指針となっている. PDQ は D^3 のスケジューリングに対して横取りスケジューリングをすることを許可し, デッドラインを素早く満たせるフローについては積極的に帯域を割り当てるような制御を行う.

PDQ では, デッドラインが短いフローから, サイズが短いフローから, の二つの規律に従ってスケジューリングを行っている. このスケジューリングを実現するために, 分散スケジューリングレイヤーによって, 他のスイッチと連携を取り, 安定的な帯域割り当てを実現する. PDQ によるスケージュリングは FIFO droptail キューを用いて近似的に実現している. シミュレーションによる実験では, 現実世界でのトラフィックでの検証を行い, D^3 に対して, 平均 FCT を 30% 以上改善することができたと報告している.

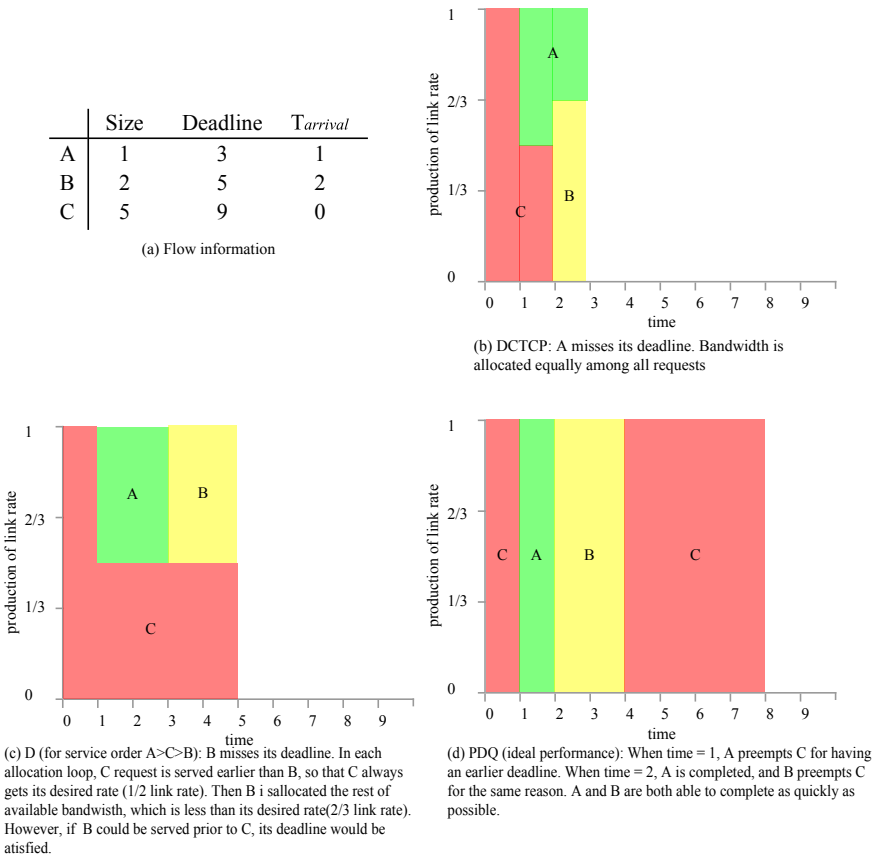

Fig.2.2　Fair-share(e.g. DCTCP)and D^3 are sub-optimal in meeting deadline

D^2TCP, MCP

D^2TCP [41], MCP [60] では, フローの TCP ウィンドウサイズを調整することで, デッドラインを満たす制御を行う. いずれの手法も, ECN によって輻輳状態を推定する DCTCP [8] を基にした手法で, DCTCP の手法にフローのデッドライン情報を考慮に入れた, 優先制御を実現している.

D^2TCP はガンマ補正関数を用いてデッドラインを満たす TCP ウィンドウサイズに調整する. 具体的には, デッドラインに近いフローほど, より多くのウィンドウサイズが割り当てられるような制御がされる. MCP [60] はさらにもう一歩先に進んだ手法となっており, 通信が開始した直後にデッドラインを満たすようなウィンドウサイズの調整が行われる. 具体的には, ECN ベースの輻輳制御が行われ, 長期的なパケットごとの平均遅延時間を最小にする最適化問題としてアルゴリズムが実装されている. Chen ら [60] は最適なウィンドウサイズの割り当てを凸計画問題として解き, 数値評価によって提案した近似アルゴリズムの有用性を示している.

2.3.2　アプリケーションベースの優先付け

DeTail

Detail [10] はテール FCT の改善を目的としたクロスレイヤーフレームワークである．Zats らはロングテールなレイテンシを持つ FCT となる二つの主な原因を示し, それらを低減する二つの手法を提案した．原因の一つは, それぞれのフローの中に優先度がないことである．一時的な輻輳状態の際, ロングフローの継続的な通信のためショートフローが十分なデータ量が通信できなくなり, 大きく遅延するフローが発生する．DeTail では, 近年標準化された PFC(Priority Flow Control) による QoS 制御によって, データリンク層においてこの問題を解決する．PFC は近年の L2 スイッチには実装されており, 輻輳時に優先度の低いフローについて一時的に通信を止めることによる優先制御をスイッチからエンドノードに対して通知することで実現可能である．

二つ目の原因は, ネットワークレイヤーにおける不均一なロードバランシングである．現在実装されている, ハッシュベースのロードバランス手法では, 輻輳が発生していない経路が存在しているにもかかわらず, ロングフローとショートフローが同一の経路を選択する可能性がある．DeTail では, 各経路の通信状況に適応させるマルチパスロードバランス手法 (§2.4) によって, この問題を解決している．

pFabric

多くの手法が様々な仕組みを組み合わせた複雑なシステムによって遅延の改善を目指してきたが, 提案した pFabric [11] では, 最小限のシステムによって問題の解決を目指している．pFabric の設計指針は, 優先制御による利得を最大限に引き出すことである．Alizadeh らは, フローによって通信レートを差別化する手法は広く提案されているが, それらは効果的でなく, 実装も難しく, そうした送り手側の通信レートの制御に代わり, スイッチにおけるショートフローの優先度付けのみで十分有効であると述べている．

pFabric では, デッドラインが短いフローや, ユーザエクスペリエンスに大きく関わるフローについてはヘッダーの優先値を設定しておき, 中継スイッチのキューにおいて, 優先値の昇順で処理を行っていくことで, 優先度の高いフローが先に処理されていくような制御を実現している．さらに, スイッチのバッファは非常に小さく設定しておき, 容易にバッファの占有, パケットロスを引き起こさせる．ns-2 でのシミュレーションでは, pFabric は平均, テール FCT 共に理論値に近い値を達成している．

2.4　複数経路の利用

4 つ目の手法は, データセンターに内在する複数の経路を利用することである．多くのネットワークトポロジーでは, サーバ間通信の等価な通信経路が複数存在している．既存のマルチパ

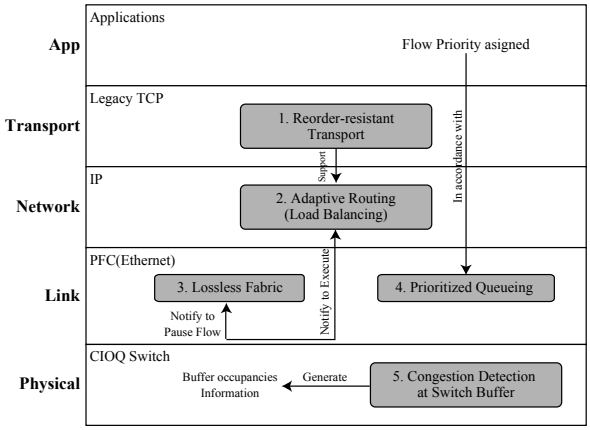

Fig.2.3　DeTail's cross-layer design

スを活用した手法として, ECMP(Equal Cost Multi-path) がある. これは, フローの 5 タプルを用いてハッシュベースに経路を選択する手法であり, 通信トラフィックを準最適化する. マルチパス環境において, 輻輳のしていない経路を選択することで, キューイング遅延の影響を避けることができ, 様々なマルチパスロードバランシング手法が提案されている.

DeTail

2.3.2 節で述べたように, DeTail [10] では, マルチパス環境で, ネットワークレイヤーの通信経路に適応させたロードバランシング手法によって, 複数の経路をより効率的に活用することができる. リンクレイヤーでの優先付けと併せて, Fig.2.3 にこれら二つの手法から構成されているクロスレイヤーによる DeTail のシステムを示す.

特に DeTail のロードバランシングでは, パケットごとに制御が行われ, パケットがスイッチに到着したときに, 最小経路の中で, 最も利用率の低い経路を選択して転送される. 通信している経路が輻輳状態になったら, PFC によるポーズ通知がすぐに送られ, 経路変更のトリガーとなる. アイドル状態の経路が存在しない場合, そのフローの通信が一時的に停止する. DeTail ではパケットごとに制御行われるため, それぞれのパケットを監視することは難しく, パケットの順序が入れ替わった場合の処理については, 既存の TCP に一任することとなる.

RepFlow

これまでの遅延改善手法は, スイッチ, エンドノード, ネットワークスタック, あるいはネットワークファブリックそのものに対して変更が必要だった. RepFlow [45] では, スイッチやエンドノードのカーネルに対する変更なしで, ショートフローを複製し各経路に転送することで FCT の改善を目指す, 単純で効果的な手法である.

RepFlow は, 例えば FatTree トポロジー [7] のような複数の経路が存在するネットワークでの経路の多様性に着目している. バースト性のあるトラフィックによる一時的な輻輳状態や

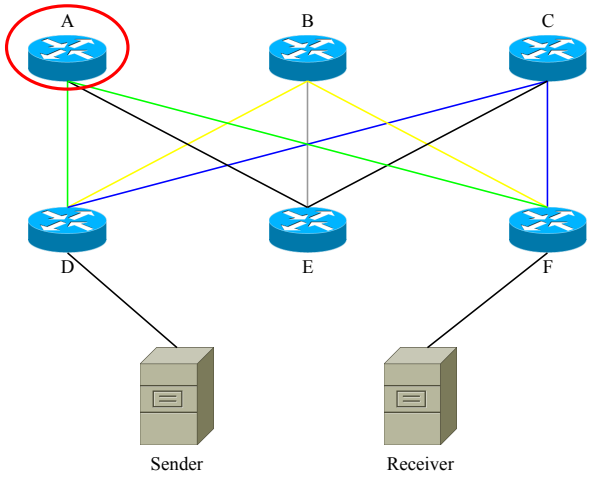

Fig.2.4　An example to understand RepFlow

ECMP によるハッシュの衝突は, 基本的には場所, 時間についてランダムに発生するものである. 結果的に, 異なる経路における輻輳の度合いは統計上独立であると考えることができる. RepFlow では, 複製されたフローと元のフローは高い確率で異なる経路を通り, その両方のフローがキューイング遅延が起こる可能性は小さい. Fig.2.4 にホットスポットを回避できるシナリオを示す.

データセントートラフィックの特徴として, 90% 以上がショートフロー (≤100KB) で全体のデータ量のおよそ 5% のみの割合である [8, 11] ことからショートフローの影響はとても小さいと考えることができる. さらに, RepFlow は他のトランスポートプロトコルとの互換性があり, 既存の TCP だけでなく DCTCP [8] に対しても適用することができる. Xu らはシミュレーション実験によって RepFlow の性能を評価し, FCT の平均値と 99 パーセンタイル値の 50%~70 改善することができたことを示した.

2.5　本研究の位置付け

これまでの関連研究を踏まえて, キューイング遅延が生じている原因であるスイッチに対する変更により, 直接情報を取得し, 改善を行う傾向がある. しかし, 実環境への適用の点では, すべてのスイッチ機器の変更を加えることは現実的ではないと考える. 近年のデータセンターネットワーク, またその上で稼働するアプリケーション性能向上の観点から, 以下のような要件が考えられる.

- 大規模計算機を有効活用するトポロジーの利用
- 分散処理の際に発生する大量のサイズの小さいフローの送信時間の短縮
- 特殊な実装やデバイスを用いず, 汎用的でシームレスな運用の実現

本研究ではこれらの要求案件を満たす, データセンターでのショートフローの改善手法を提案する.

第 3 章

データセンターネットワーク

本章では, データセンターネットワークを構成する技術に関して, その概要を述べる.

3.1 トポロジー

従来のデータセンターモデルでは, エンドノードが Edge スイッチにつながり, これらのスイッチが Aggregation スイッチに集約され, core スイッチに接続するといったように, 階層的にトポロジーを形成していた [7]. このような単純な階層構造を持つトポロジーは, トラフィックの大部分がデータセンター外の通信には有効であった. しかし, 今日のようなデータセンター内で生じるトラフィックが大半を占める場合, 帯域の割当が適切でなくなる. このような, データセンター内のトラフィックが主であれば, 階層型のトポロジーはボトルネックを引き起こす可能性がある. 近年の研究 [7,46,61] では, トラフィックがデータセンター内に集中した時の問題を, 物理的なアプローチとして, トポロジーを工夫する事で解消を試みている.

Fig.3.2 のように, FatTree [7] では, Core スイッチを複数用いる事で, 物理パスの最大帯域を供給する. また, 比較的狭い帯域の経路と汎用的な性能のスイッチを多数用いる.

このようなトポロジーを用いる事で, データセンター内のトラフィックに対し, 帯域を十分に使う事ができる. しかしこのような密な配置により, 複数の経路が形成され, ルーティングをどのように決定すべきかという問題も生じる事となる [9]. 例えば Fig.3.2 のような FatTree トポロジーでは, 4 通りの経路が考えられる. これら複数の経路をリンクエラー時の冗長性を持たせる目的だけでなく, 性能向上に活用することが求められている.

3.2 マルチパス, マルチホーミングを実現するプロトコル

本節では, 複数の経路を持つデータセンター環境においてネットワーク資源の有効活用を実現するプロトコルについて述べる.

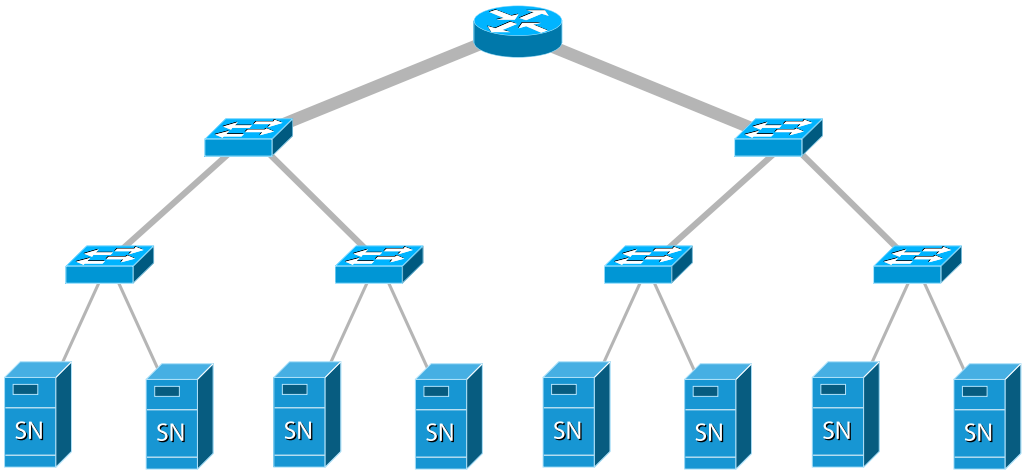

Fig.3.1　Hierarchical network topology

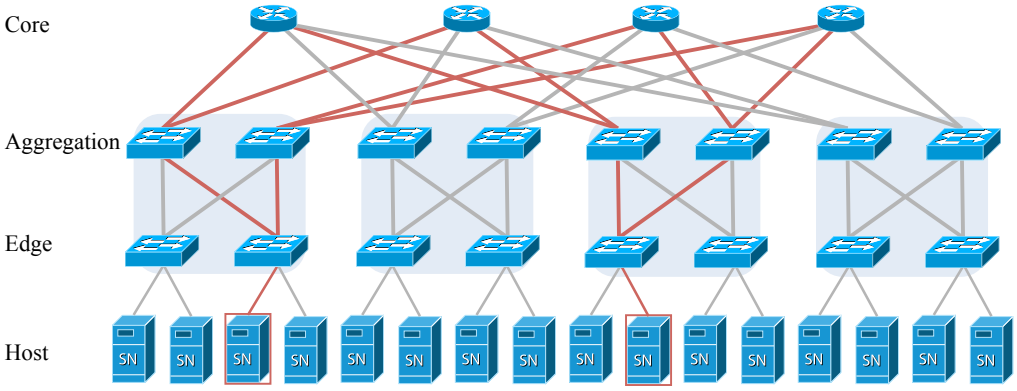

Fig.3.2　Fattree topology

3.2.1　データリンク層におけるアプローチ

LACP

データリンク層のプロトコルは, 通信媒体で直接接続された機器間で通信するための仕様を定めている. データリンク層における複数経路を利用するためのプロトコルとして, LACP(Link Aggregation Control Protocol) がある [32]. LACP により, 通信ケーブルが通信不良を起こした際にも, 束ねた回線のうち正常に動作しているケーブルによって冗長化することができる. また, 同一スイッチの異なる複数のインタフェースと接続し, 複数の物理リンクを束ねて, 一つの論理リンクとして扱うことで, リンク容量を集約することができる. LACP は, パケットヘッダーに対するハッシュ計算での物理リンクへのパケット多重化によって, 負荷分散を実現する. このように, LACP によって容易に帯域を拡張し, 通信性能を向上させることができる. データリンク層での負荷分散では, ハッシュ計算のアルゴリズムとして主に Mac ア

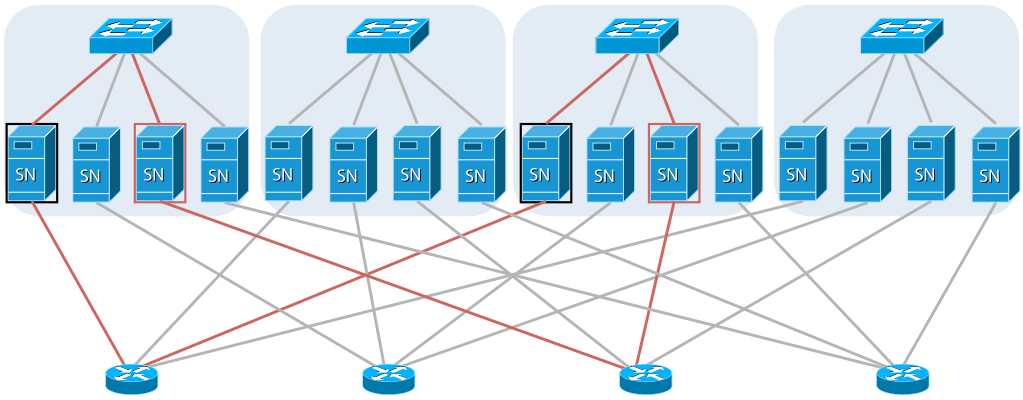

Fig.3.3　BCube topology

ドレス, IP アドレス, ポート番号などを用いるが, どの値をキーにするかは, 各ネットワーク機器の実装に依存する. 実際, MAC アドレスを用いて通信を行う際, 二つ目以降の MAC アドレスを取得するすることは ARP プロトコルの実装上できない [34]. そのためクライアントサーバ通信において, 異なる種類のスイッチによって異なるキー値を LACP の負荷分散アルゴリズムを用いる場合, 完全な負荷分散が実現できず, 通信性能の向上が見込めない. 完全な負荷分散を実現するためにはキー値を同一になるように調整する必要があるが, 現実的には困難である [33].

3.2.2　トランスポート層におけるアプローチ

SCTP

SCTP [29] とは TCP と同様にパケットの到達, またその到着順序を保証する信頼性のあるプロトコルであり, 標準でマルチホーム機能をサポートしている. 標準化されているプロトコルにおいてマルチホーム機能は冗長性のためのものであり, 冗長経路へ新しいデータを送信することは推奨されていなかった. そこで SCTP において, 冗長回線を用いた帯域集約を行うプロトコルである CMT (Concurrent Multipath Transfer) が提案された [30, 31]. CMT では, 期待されていた値よりもウィンドウサイズが増加しないという課題があり, 大きく 3 つの欠点に対して改善が行われた.

送信者による不要な再送処理

SACK パケットのパケット損失情報とそれぞれのパケットの宛先アドレスから, パケットロスなのか順序の入れ替えなのかを判断し, 最小限での再送制御を実現している.

送信者側におけるウィンドウサイズの更新頻度の少なさによるウィンドウサイズの増加の抑制

新しいパケットに対する ACK パケットのみに従いウィンドウサイズを更新していたため, ウィンドウサイズ増加が抑制された. CMT ではパケットをどの宛先に送信したかを把握しておくことで, 回線に応じたウィンドウサイズの増加を行うようにした.

ACK パケットの増加

SCTP や TCP においてパケットが順序通りに受信できた場合に, 遅延 ACK によって複数の ACK 応答を一つにまとめることで通信オーバーヘッドを減らしていたが, CMT では複数回線を集約しているためパケットが順序通りに届かないことが多く ACK パケットが増加していた. そこで, CMT では順序通りに届かない場合においてでも, ACK の送信を遅らせることにより, ACK のオーバーヘッドを削減した.

SCTP を拡張したマルチパス通信のメリットは SCTP 自体にマルチホーム機能をサポートしていることにある. しかし, SCTP は現在様々な通信において広く利用されている TCP や UDP とは異なるプロトコルであるために, 既存のアプリケーションに対して変更が生じるという課題がある.

Multipath TCP

MPTCP は, 一つの経路でデータ転送する TCP を拡張し, 複数のインタフェース, あるいは複数のポートを用いてデータ転送をするプロトコルである [13]. クライアントが複数の IP アドレスを持っていた場合, 新たにサブフローのコネクションが確立される. 追加されたサブフローは, クライアントの持つインターフェースが 1 つの場合, 同じ IP アドレスで異なる送受信ポートを用いる. インターフェースを複数持つ場合には, 異なる IP アドレスの組み合わせで通信を行う. ルーティングに関しては, 複数の宛先 IP アドレス, 送信元アドレスからそれぞれ経路決定される. このように, アプリケーション層より下のレイヤーのみで複数の経路を使ってデータ転送を行うため, アプリケーション側が MPTCP での通信を意識することなくデータ転送ができる.

MPTCP では, サブフローが, それぞれのシーケンス領域を持ち, 経路状態に合わせて輻輳制御をする [62]. 輻輳制御には, TCP と同様に AIMD(additive-increase and multiplicative-decrease) による輻輳制御がサブフロー単位で行われる. 以下に AIMD アルゴリズムを示す.

- サブフロー r において, 1ACK ごとにウィンドウサイズ ω_r を $\min(\frac{\alpha}{\omega_{total}}, \frac{1}{\omega_r})$ 増加させる.
- サブフロー r において, パケットロス時にウィンドウサイズ ω_r を $\frac{\omega_r}{2}$ へ減少させる.

ここで, ω_{total} は全てのサブフローのウィンドウサイズの総和, α は送信速度の増加量を示すパラメータで, 以下のように定義される [62].

$$\alpha = \omega_{total} \times \frac{\max_r \frac{w_r}{RTT_r^2}}{(\sum_r \frac{w_r}{RTT_r})^2} \tag{3.1}$$

ここで, RTT_r はサブフロー r での RTT を示している. MPTCP での輻輳制御には二つの性質ある. 一つは, サブフローのウィンドウサイズは, 全てのウィンドウサイズの大きさに依存

するということである．これにより, 混雑したサブリンクにおいては, ウィンドウサイズが抑えられ, ロードバランスができる．二つ目は,MPTCP のアルゴリズムによって, TCP での輻輳制御よりも悪化する事を回避している事である．しかし, もし複数のサブフローがそれぞれ混雑のないサブリンクを利用する場合, いずれかのコネクションが帯域を占有する可能性がある．

3.3　トラフィック

大量の計算機資源を有効活用するためには, 並列分散処理フレームワークを用いられ, 多数の処理ノードと分散処理の制御をする管理ノードから構成されている partition-aggregate 構造をとり, 管理ノードからクエリーが発行され, 処理ノードがそれを受け取り, レスポンスを返す．このとき, トラフィックパターンが (1)*Query traffic*, (2)*Short message traffic*, (3)*Backgroung traffic* の 3 つに分類される [8].

Query traffic. Query traffic とは, 大規模計算処理を分割して並列処理を開始する際に, aggregator ノードから処理ノードへ具体的な処理を割り当てるためのトラフィックである．Query traffic の特徴は, 非常に小さいフローサイズ (2KB∼20KB) で, フローの役割上, 処理全体の遅延に非常に強く影響を及ぼす事である．そのため, アプリケーション性能を考慮すると, 低レイテンシでの通信が求められている．また並列分散処理システムの構成上, Query traffic は ms∼ μs 単位で生成され, バースト性があるといえる [8].

Short message traffic.　Short message traffic とは, 処理ノードの動作を制御するためのトラフィックである．Short message traffic の特徴は, フローサイズは 50KB∼1MB で, Query traffic と同様に処理全体の遅延に影響を及ぼすという事である．しかし, Querry traffic ほどのフロー数は生成されず, 生成間隔も秒単位である．

Backgroung traffic. Backgroung traffic は, 各処理ノードへ更新データを送信するトラフィックである．Backgroung traffic の特徴は, フローサイズが 1MB∼50MB と大きいことにある．さらに, その生成間隔は大きい．また, Backgroung traffic での更新データは, 処理精度の向上に寄与するが, 処理に必須ではないので, 処理全体の遅延にはつながらない.

つまり, 分散処理開始時に生成される Query traffic が遅延すると, 処理全体に対し遅延を引き起こすので, Query traffic を含むショートフローのフロー完結時間は極めて重要なメトリックである.

また, Alizadeh らは, 実際のデータセンターのトラフィックでは, レイテンシ志向なショートフローとスループット志向なロングフロー, そしてバースト性のある Query traffic が混在していると報告している．さらに, Background traffic のフロー数自体は少ないが, 全体のトラフィック量の大部分が Backgroung traffic によって占められているという特徴がある [14].

フローがデッドライン時間までに完結するためには, フローのサイズも大きく影響がある．つまり, 事前に通信が発生するフローサイズを知っておくことが重要である．実際, 今日の大部分のクエリーレスポンスアプリケーションにおいては, 処理ノード同士のフローのサイズは通信開始時に決定する．例えばウェブ検索アプリケーションでは, top-k query アルゴリズム

などを用いて, 基本的には上位 k 件の結果のような固定的なレスポンスを返す. キーバリューストア [42,43] や並列分散処理 [6,44] などの技術でも同様である. 従って, 多くのウェブアプリケーションにおいて, 実際に処理が終わった後ではなく, 通信が開始した時点で, アプリケーションのコードからレスポンスフローのサイズを知ることができる.

3.4　アプリケーション

今日の主なデータセンターアプリケーションとして OLDI(OnLine Data Intensive) アプリケーションがあり, “オンライン性” と “集約性” の大きく 2 種類の特徴がある [35]. “オンライン性” とはクライアントとサーバーとのインタラクティブな通信を示している. 具体的には, クライアント側からブラウザ経由でクエリーを送り, サーバ側はクエリーを受けるとすぐにレスポンスを返す通信を行う. OLDI アプリケーションでは, 一つの通信に対して例えば 300ms 以内のようにデッドラインを超えないようにレスポンスを返すことが期待されている. 一方 “集約性” とは, 大量のデータからレスポンスを生成することを示している. 具体的には, Web サーチのような全文検索により結果を返すような演算のことである. 実際のデータセンターでは, 大規模なデータセットは数千台規模のサーバに分散して格納されており, クエリーに対して対象のデータが格納されているサーバーに対して検索を行う.

こうした二つの特徴を持った OLDI アプリケーションは Fig.3.4 に示すようなツリー型のアーキテクチャによって構成され, partition-aggregation 型アルゴリズムによって処理が行われる. このような構成を取るため, 上記のような性質が性能の面で問題になる. [39] Fig.3.4 ではアーキテクチャは二段によって構成されているが, アプリケーションの規模によってはより深いツリーとなることもある. partition-aggregation 型アルゴリズムでは, まず root ノードがクライアント側から送られてきたクエリーを受けとり, クエリーの対象データが格納されている Worker ノード (leaf) へとジョブが分割される. それぞれの Worker ノードがレスポンスを返し, Aggregator ノード (parent) はそれらを集約する. さらに, Root ノードがそれぞれの Aggregator ノードのレスポンスを集約し, 最終結果をクライアントへ返す. Root ノードから各 Worker ノードへのクエリーの下り通信と, 各 Worker ノードから Root ノードへのレスポンスの上り通信は, アプリケーション性能の観点から, デッドライン時間以内に完結されるべきである. それぞれのデッドライン時間は, アプリケーションが満たすべきデッドラインから, そのアプリケーションが構成しているアーキテクチャの段数によって割り当てられる. 例えば Fig.3.4 では, Worker ノードはクエリーを受け取ってから 30ms 以内に Aggregator ノードへレスポンスを返さなければならない. もし, デッドライン時間を超えてしまった場合には, Aggregator ノードは得られたレスポンスのみを集約し, Root ノードが最終的にレスポンスを返す. その結果, レスポンス結果の品質に影響が出ることとなる.

それぞれの Worker ノードの演算時間は, Worker ノード自身の演算時間と, Aggregator ノードとの通信時間のそれぞれの和となっている. それぞれのデッドラインの決定には, 処理結果の質とレスポンス時間のトレードオフとなる. 通信時間に対して余裕のあるデッドラインを設

定した場合, デッドラインを超えるノードの数は少なくなるが, 各処理ノードでの演算かける時間が少なくなり, 十分な質の結果が出せない可能性もある. 具体的には, 検索対象とするデータの範囲を小さくすることで処理時間を節約することとなる. そのため, より多くのノードがデッドラインを超えないようにするためのデッドライン時間を設定するだけでなく, より良い結果を返すために演算時間にも十分な割り当てが必要である. [41]

次に, Aggregatrtor ノードのが各 Worker ノードへクエリーを送った時の Aggregator ノードが構成するサブツリーでの挙動について考える. すべての処理ノードがほぼ同時刻にレスポンスを受け取り, 同じデッドライン時間を持っている. その結果, 多対 1 通信の大量のトラフィックが Aggregator ノードへ到達する [8, 36]. そのため一つのノードへレスポンスが同期しないように, ジッターを混ぜることで Aggregator ノードのメモリーを圧迫しないようなアプリケーションレベルの工夫もされているが, その結果テール状に遅延が発生する [37]. またデータセンターでは複数のアプリケーションが同時に起動しているため, こうしたバースト性のあるレスポンストラフィックは一つのスイッチにおいてでも同時多発的に発生すると考えられる.

さらに, こうした比較的サイズの小さい大量の多対 1 通信に加え, データセンターネットワークでは長時間通信を行うバックグラウンドトラフィックも同時に通信を行っている. バックグラウンドトラフィックは, 各処理ノードに対して, 最新のデータを更新するための通信であり, 長時間にわたり非常に大きなデータサイズを通信する. それにより, 既存の TCP の性質上, こうしたロングフローがスイッチのバッファを使い果たすこととなる.

上記のような大量の多対一トラフィックと長時間通信を行うバックグラウンドトラフィックによって, スイッチのバッファが圧迫され, 遅延が生じてしまう. こうした性能障害を解決するための手法の一つとして, バッファに溜まったパケットの処理高速化がある. 実際, データセンターの一部で用いられるネットワーク機器では, 特定の用途のために ASIC によるオンチップパケットバッファメモリを持つモジュールが用いられる. しかし, 近年のデータセンターでは汎用的なネットワーク機器を用い, コストを低くして運用する傾向があり, 特定の用途に特化した高価で複雑な構成のスイッチ機器を用いられることは多くはない. [38] また, バックグラウンドフローによるバッファの圧迫により大きなキューイング遅延を引き起こし, OLDI トラフィックのデッドラインを超えるほどの遅延を引き起こすため, バースト性のあるトラフィックによる遅延問題を解消するためには, バックグラウンドフローに対する改善が必要である [8].

この輻輳の問題は, OLDI アプリケーションが持つ基本的な性質であるが, 今日のデータセンターネットワークではデッドラインを超えないために大きく二つの改善を行っている.

一つは, ネットワーク機器によるスケールアウトである. TCP/IP ネットワークでは, 基本的にパケットロスが起こった際にタイムアウト時間からエンドノードへと経路の輻輳状態がフィードバックされる. そのため, パケットロスが生じたときには, RTO 時間の間, 新しいデータを送信することができず, デッドラインを超えてしまう. また, ウィンドウサイズも減少し, 通信レートも下がる. こうした課題に対して, (1) ネットワーク帯域の増強, (2) 通信時間の割

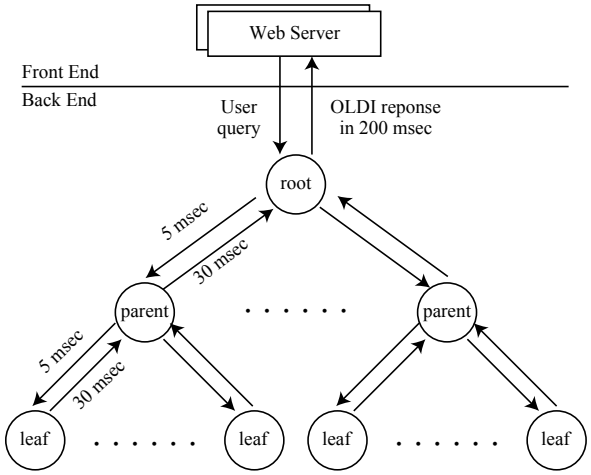

Fig.3.4　OLDI architecture

り当ての増加によって, 改善する手法がある. 前者のアプローチは比較的大きなコストがかかるが, 後者の場合, より多くの処理ノードを追加し, 一つのノードあたりの処理時間を減らすことによって, その分通信時間への割り当てを大きくすることができる.

もう一つは, フローに対する優先付けである. 理想的には, デッドラインが近づいているフローに対しては優先的に通信を行い, 比較的余裕のあるフローに対しては後回しにする処理をすればよい. しかし, 現状広く用いられている TCP は通信の公平性を持つプロトコルであり, デッドラインによる区別は行わない. OLDI アプリケーションはデッドラインに依存せず, 公平にネットワーク資源を分配するが, そのようなプロトコルがデータセンターネットワークに対して適しているとは言えない [40].

第 4 章

Motivated work

この章では, フローサイズの小さいトラフィックに対して, MPTCP では TCP よりも通信性能が劣化すると報告された問題 [9] に対して, 大規模データセンターネットワークへの MPTCP 適用時の問題点を把握するために, フローサイズの小さいトラフィックに対する性能を検証, ならびにクラウドサービスを想定したトラフィックの一例として並列分散処理アプリケーションを用いた二種類のトラフィックの測定結果を示す. 測定結果からトラフィックの特徴を示す事で, 従来の TCP で構成されたクラスターの抱えるボトルネックと複数のキュー, 複数の経路を持つマルチパス環境におけるデータセンターモデルの利点をそれぞれ示し, 提案手法の設計指針とする.

4.1 再現実験のための予備実験

この節では, MPTCP データセンターネットワークに対して実際のデータセンター環境を想定したシミュレーション実験として, 過去に行われた実験 [9] の再現実験を行うために, 個々のデータセントラフィックに対する MPTCP と TCP の挙動の違いを検証する.

4.1.1 想定環境

予備実験では, データセンターネットワーク上で partition-aggregate モデルに従う並列分散処理を想定する. ネットワークトポロジーについては, $k = 4$ FatTree トポロジーを用い, 全 4 ポッドの内, 1 ポッド (4 ノード) が管理ノード群として割り当てられ, 他ポッド (12 ノード) に対して並列分散処理を割り当てる. またベンチマークトラフィックについては, 3.3 節で述べた内, 2 つのトラフィックパターンについて評価・考察を行う. メトリックとして Table.4.1 を考える.

Table.4.1　Metric of each traffic patern

トラフィックパターン	メトリック
Query traffic	フロー完結時間 [sec]
Short message traffic	フロー完結時間 [sec]
Background traffic	スループット

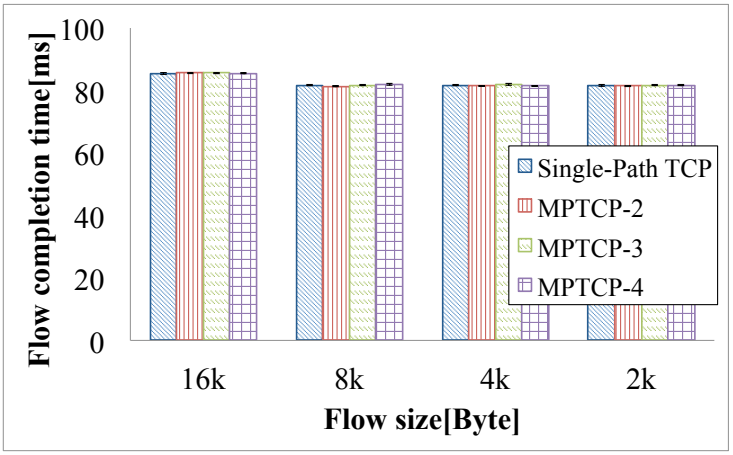

Fig.4.1　Flow completion time of Query traffic with Background traffic

4.1.2　シミュレーション結果

Query traffic

Query traffic に対する評価として, フローサイズを 1KB∼16KB とし, 12 の処理ノードへ平均 200ms のポアソン生起でトラフィックを発生させ, FCT を測定した. また, Query traffic のみ発生させた場合と, 50% の処理ノードに対し継続的にデータを送信するトラフィックである Background traffic を同時に発生させた場合の 2 パターンについて評価を行った. その結果を, Fig.4.1, 4.2 に示す. なお, エラーバーとして 99% 信頼区間を採用した.

この結果から, MPTCP は Query traffic に対し, 直接性能に影響を及ぼさず, Background Traffic による影響が性能差を生じさせたことが分かる. これは, 今の MPTCP の実装上, サイズの小さいデータ通信においては複数経路を利用せず, コネクション確立時に用いた経路のみで通信を完結するため, Singlepath-TCP と同じ挙動となる. 従って, Query traffic に対しては直接的に MPTCP が作用しない. 一方で, サイズの大きいデータ通信に対しては複数の経路を利用し, 利用する経路分のスループットを達成できる. そのため, より多くのデータを転送し, 帯域を大きく占有する MPTCP の影響受け, Query traffic の遅延が生じたと考えられる.

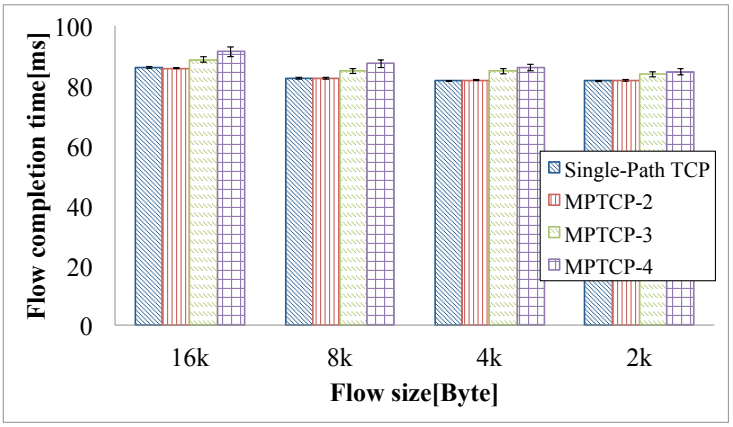

Fig.4.2　Flow completion time of Short message traffic with Background traffic

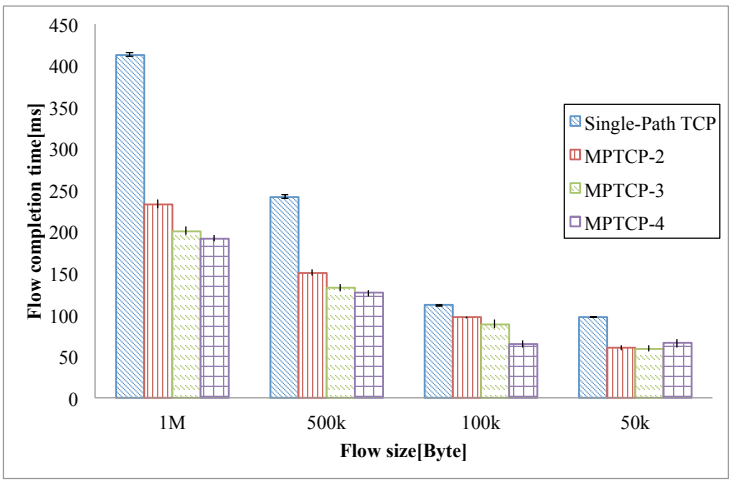

Fig.4.3　Flow completion time of Short message traffic with Background traffic

Short message traffic

Short message traffic に対する評価として, フローサイズは 50KB∼1MB とした. 50% の処理ノードに対し継続的にデータを送信するトラフィックを, Background traffic として同時に発生させた状態で, 同時に 12 の処理ノードへ平均 500ms のポアソン生起でトラフィックを発生させ, FCT を測定した. その結果を, Fig.4.3 に示す.

この結果から, MPTCP は Short message traffic に対して直接的に作用し, FCT を短縮させたことが分かる. これは, 先ほどの Query traffic よりも大きいサイズのフローであるため, MPTCP により複数経路を利用し, TCP よりもより効率的な通信が行われたことを示している. 実際フローサイズが小さいと, MPTCP による恩恵も小さくなり, MPTCP と TCP 間で FCT の差が小さくなっていることがわかる.

Table.4.2　Testbed on network simulation

環境パラメータ	値
ノード数	16
MPTCP	v0.86
帯域-core-aggr	400Mbps
帯域-aggr-edge	200Mbps
帯域-edge-host	100Mbps
RTT	0.5ms
バッファ	100KB

4.2　再現シミュレーション

この節では, Raiciu らによって示した FatTree を用いた MPTCP ネットワークモデルでのフローサイズの小さいトラフィックに対する性能評価シミュレーションを再現し, 解析を行った結果を示す.

4.2.1　再現シミュレーション実験環境

Raiciu らは [9] において, 各プロトコルがフローサイズの小さなトラフィックに対して及ぼす影響の評価を行い, フローサイズの小さいトラフィックに関しては, MPTCP によって FCT を遅延させることを示した. そのときのシミュレーション環境は, 以下の通りである. ネットワークトポロジーには, 4:1 にオーバーサブスクリプションされた FatTree を用いている. ベンチマークトラフィックについては, エンドノード間の 1 対 1 通信を用いている. 全てのノード間通信のうち, 33% を TCP または MPTCP により継続してデータ転送 (バックグラウンドトラフィック) を行う. 残りのノードを使って, TCP による 70KB のデータ転送をを毎 200ms のポアソン生起させ, 転送完了までにかかった時間を計測している.

今回の再現実験には ns-3 Direct Code Execution [51] を用い, MPTCP は, Linux カーネルソースを用いた [19]. Fig.4.4 に, シミュレーションで用いた FatTree(k=2) トポロジーを示す. このトポロジーでの物理パスでは, 一つのサブフローが 1 本の物理パスを占有するように, 設計している. すなわち, 4 つのサブフローを使う場合, ホストには 4 つのインターフェースに対しそれぞれ 4 つの IP アドレスが割り当てられることになる. また, Host-Edge 部分には, IP アドレスの数だけインターフェースを用意し, Aggregation-Edge 部分も, それに従いインターフェースを追加する. さらにルーティングに関しては, インタフェースごとに Core1∼Core4 へと分散するようにルーティングテーブルを設定した.

Table.4.2 に再現シミュレーション環境に対する各パラメータをまとめる.

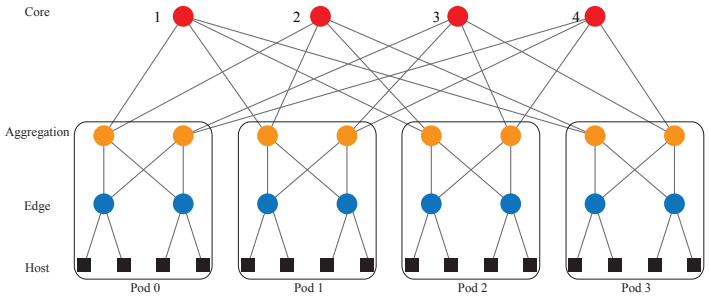

Fig.4.4　Network topology on reproducing simulation

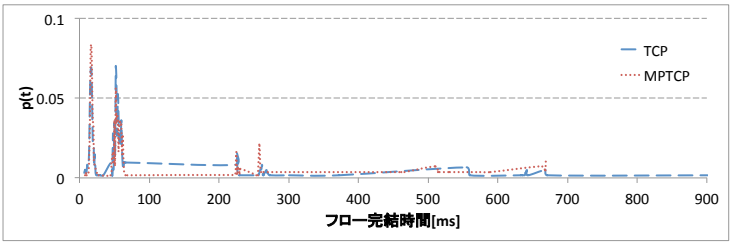

Fig.4.5　The result of the reproduction experiment

設定パラメータに対する有効性の検証

伝搬遅延については RTT(Round Trip Time) として, 0.5ms に設定した. これは, 一般的なデータセンター内の RTT が 1ms 以下であるためである [20].

ウィンドウサイズについては, 以下の帯域幅遅延積 (BDP) の式から, 400Mbps を最大限利用できるだけの値を設定した.

$$BDP[\text{byte}] = \text{帯域幅 [bps]} \times RTT \div 8 \tag{4.1}$$

各帯域については, 16 のノードを使って輻輳を引き起こす現象を再現するために, 実際のデータセンターのような広帯域のネットワークと比べ, 狭い帯域を設定した.

4.2.2　再現結果

Fig.4.5, Table.4.3 に, 上記の実験環境で再現した結果を示す. 再現結果から, フローの様子を完結時間別に 4 パターンに分類することができることが分かった. Table.4.4 にそのフローパターンの定義を示す.

4.2.3　考察

Table.4.4 に示した各フローパターンについて, それぞれの特性を分析する.

Table.4.3　Average flow completion time and stdev on reproduction experiment

プロトコル	平均フロー完結時間 [ms]	標準偏差 [ms]	95 パーセンタイル [ms]
TCP	78.4	122.5	266.7
MPTCP	91	140.6	510.5

Table.4.4　Flow pattern classified by completion time

フローパターン	完結時間 [ms]	パケットロスの有無
Full window	∼30	なし
Intensive flow	∼60	なし
Delay with loss	200∼300	あり
Extreme delay	300∼	あり

パケットロスが発生しないフローパターン

Fig.4.6 に Full window と Intensive flow のデータ転送の様子を示す.

Full window では TCP コネクション確立後, サーバがすぐに最大ウィンドウサイズ分だけパケットを送り, クライアントからの ACK が返ってくると, 随時次のパケットを送っていた. これは経路に輻輳がなく, 多くのウィンドウを利用できたということであり, 結果的に 30ms 以下でデータ転送を完了していた.

一方, Intensive flow では, サーバが最大ウィンドウサイズ分に満たない量のパケットを送り, クライアントからまとめて送られてくる ACK を受け取った後, 集約してパケットを送っていた. その結果, コネクションの切断時に, Full window と比較して遅延を引き起こし 60ms 程度転送時間がかかった.

パケットロスが生じたフローパターン

Fig.4.7 に Delay with loss と Extreme delay のデータ転送の様子を示す. いずれのフローパターンもデータ転送中にパケットロスが発生し, 再送処理, 重複 ACK 確認応答を行った. パケットロスが起きた原因は, 短時間にフローサイズの小さいトラフィックが中継ルータを集中したためである. 実際, 200ms のポアソン生起のうち, 数 10ms 単位の短い期間でトラフィックが発生したとき, 中継ルータにおいてパケットロスが生じた.

Delay with loss では, TCP コネクション確立後に数パケットのデータ転送を行い, パケットロスによるタイムアウトを生じた. その後, 再送処理を経て, Maximum Segment Size (MSS) である 1460byte でパケットを伝送した.

一方, Extreme delay では, TCP コネクション確立直後にパケットロスによるタイムアウトを生じた. その後も, パケットロスは生じないものの, 400ms 頃まで遅延が生じていた. ま

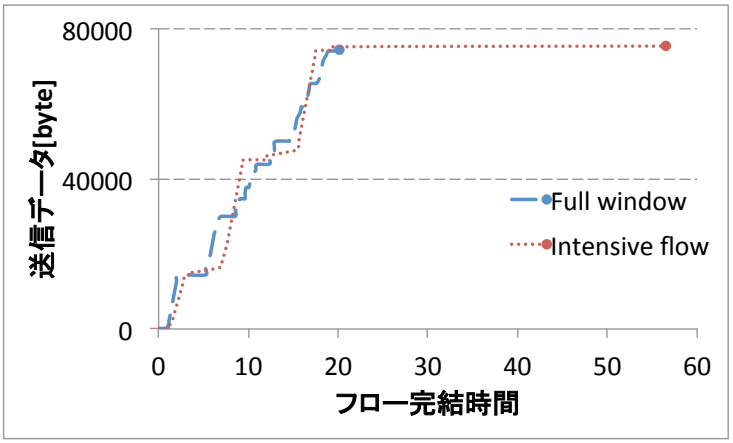

Fig.4.6　Comparison between Full window and Intensive flow

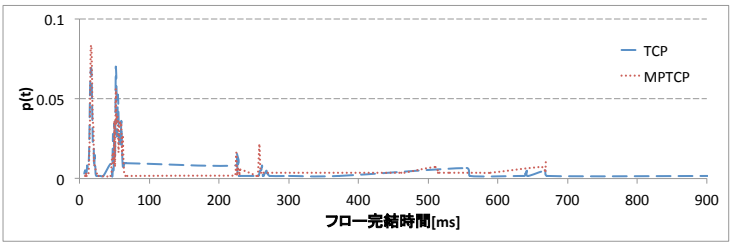

Fig.4.7　Comparison between Delay with loss and Extreme delay

た, Fig.4.7 における二つのグラフの傾きは, セグメントサイズ最小値の 586byte に設定され, 転送速度が上がらなかったことを表している. これは, 中継するルータにおいて QoE 制御による帯域制限が発生したことを示している. 実際, 同時刻に流れていたバックグラウンドトラフィックのスループットには変化がなく, QoE 制御の rate control によりバックグラウンドトラフィックのデータ転送が優先され, ベンチマークトラフィックにはウィンドウサイズが制限されたと考えられる.

TCP v.s. MPTCP

今回の再現実験において, TCP と MPTCP で FCT に差を生じた要因は, パケットロスが発生する割合にある. Fig.4.8 に再現実験での FCT ごとの累積確率分布を示す. Fig.4.8 から MPTCP では, 遅延が生じたフローの割合が大きいことがわかる. これは, 前述の理由のように MPTCP を用いることで, より多くの帯域が使われ, その結果一部のフローに対し, パケットロスを生じるなどの遅延の影響があると考えられる. パケットロスを生じないフローに関しては, 遅延がなく両者に性能差を感じなかったが, MPTCP を用いた方が, パケットロスを引き起こし遅延を生じさせる割合が大きいため, 全体的なサイズの小さい通信の性能としては劣化する.

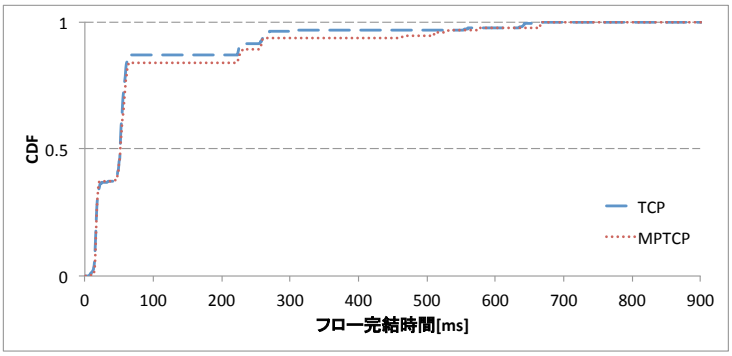

Fig.4.8　CDF of flow completion time on reproduction experiment

このように MPTCP が帯域を大きく占有することにより他のトラフィックを圧迫することは, MPTCP の輻輳制御によるものだと考えられる. 混雑のない経路でデータ転送する場合, MPTCP では積極的にウィンドウサイズを増やそうとするため, 他のフローに対し遅延を引き起こしたと推測される.

4.3　実トラフィック解析

この節では, クラウドサービスを想定したトラフィックの一例として並列分散処理アプリケーションを用いた二種類のトラフィックの測定結果を示す. 測定結果からトラフィックの特徴を示す事で, 4.2 において考察した状況での遅延が起こる可能性の検証, また従来の TCP で構成されたクラスターの抱えるボトルネックと複数のキュー, 複数の経路を持つマルチパス環境におけるデータセンターモデルの利点をそれぞれ示し, 提案手法の設計指針とする.

測定環境には, 管理ノード 1 台 (Master), 処理ノード 10 台の計 11 台のクラスター PC を用いた. 管理ノードは 10Gbps イーサネットリンクで Top of Rack(ToR) スイッチに接続されている.

このクラスター PC で Presto [5] によりインタラクティブなレスポンスを返す, 分散 SQL データベースを実現しており, §3.3 で示した三種類のトラフィックが混在している. トラフィックの測定には, 管理ノードのインターフェースを用いて, tcpdump [26] によるパケットレベルの測定を行った.

定常状態:　管理ノードに対し, ジョブ命令を一切与えていない中で約 10 時間程度トラフィックを測定した. Fig.4.9 に定常時のフローサイズの累積分布を示す. この分布から, 80% 以上のフローが 10KB 以下であるようにショートフローの数が全体のトラフィックの大部分を占めていることがわかる. 一方で通信量に着目すると, フロー数は比較的少ないがフローサイズの大きいトラフィックが大半を占めている.

次に, Fig.4.10 に管理ノードへのトラフィックの影響を示す. この分布が示すように, 各処理ノードから管理ノードへのトラフィックの割合が大きく, それぞれフローサイズも大きい. 一

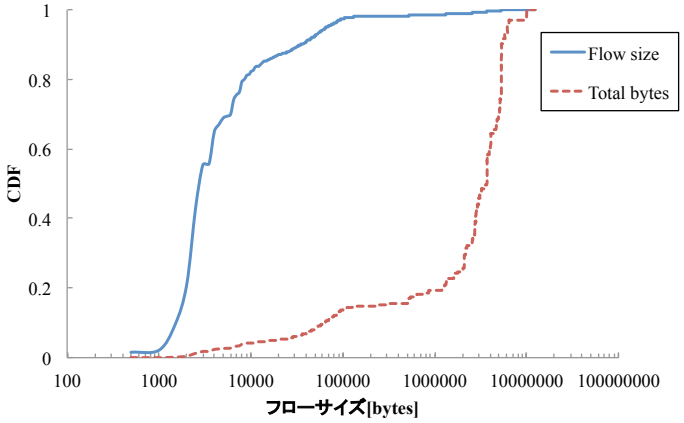

Fig.4.9　Distribution of Presto cluster in a steady state

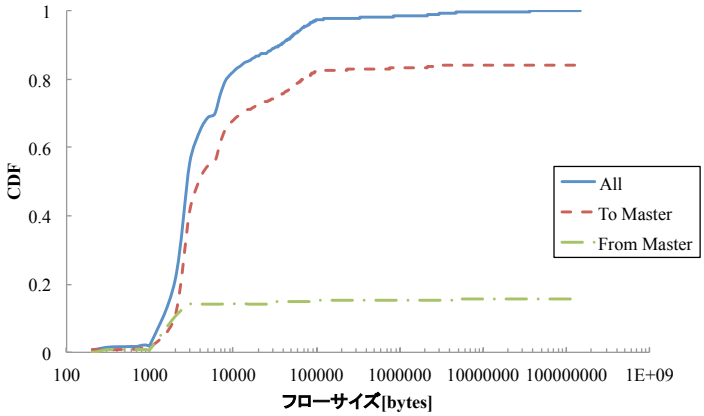

Fig.4.10　CDF of the traffic from master node in a steady state

方で, 管理ノードから各処理ノードへのトラフィックについては, 比較的フローサイズの小さいトラフィックの割合が大きい. これは, Presto の並列分散処理アーキテクチャ上, YARN フレームワークなどにより各処理ノードの監視を担っている管理ノードから複数の処理ノードへフローサイズの小さいクエリーを送るという 1 対多通信と, そのレスポンスとして, 複数の処理ノードから同時にノードの状態を送信するという, 多対 1 通信によるものである.

さらに, Fig.4.11 に時間毎の同時接続数の分布を示す. Fig.4.11 中の長時間通信は通信時間が全測定時間の 90% 以上であるフロー数を表している. この分布が示すように, 各処理ノードから管理ノードへのトラフィックの同時接続数が多く, 積極的に通信が行われている. また, 短い通信時間で通信が発生し終了する, スパイク性のあるトラフィックの中で長時間通信を行うフローが固定的に存在していることを示している.

並列分散処理実行時:　次に管理ノードに対し, 約 1 分間程度で完了する SQL ジョブを与えた中でジョブが完遂するまでの間トラフィック測定を行った. SQL ジョブには,

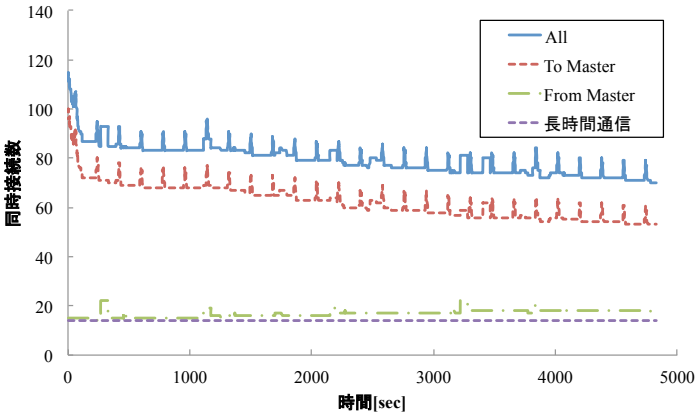

Fig.4.11　Distribution of the number of concurrent connections in a steady state

“$select * from$ \$テーブル $where$ \$条件”を実行し, 全ての処理ノードにジョブを与えられるようにした. Fig.4.12 にジョブ実行時のフローサイズの累積分布を示す. この分布が示すように, ショートフローの数が全体のトラフィックの大半を占めるが, 定常状態と比べると, 全体的にフローサイズ大きいトラフィックが増えている. 実際, 80% 以上のフローが 110KB 以下であるように, ショートフローの割合が小さくなった. 同様に通信量に着目すると, フロー数は比較的少ないがフローサイズの大きいトラフィックが大半を占めるという事が分かる.

次に, Fig.4.13 に管理ノードへのトラフィックの影響を示す. この分布が示すように, 各処理ノードから管理ノードへのトラフィックの割合が大きく, フローサイズは小さいものが多いことが分かる. しかし, Fig.4.10 の定常時のトラフィックと比べると, 管理ノードから各処理ノードへのトラフィックの割合が大きくなっている.

さらに, Fig.4.14 に時間毎の同時接続数の分布を示す. この分布が示すように, ジョブ実行中は全体的にフローの数は増え, とりわけ管理ノードから各処理ノードへのトラフィックの割合が大きくなっている. さらに, ジョブ終了後も同時接続数が大きく変化していないことから, 長時間通信を行うフローが固定的に存在している. また, 各処理ノードから管理ノードへのトラフィックに着目すると, ジョブ開始時に接続数が大きく増えている事から, バースト性があるトラフィックであるといえる.

これらの分布から, クラウド型サービスを想定したトラフィックの特徴として以下の事が述べられる.

- 定常時もジョブ実行時も同様に, 管理ノードへ送信されるトラフィック量は多い
- 長い時間通信を行うフローが固定的に存在している
- ジョブ実行時の処理ノードから管理ノードへのトラフィックには, フローサイズも小さく, バースト性がある

また, これらの特徴から, 管理ノードへのトラフィックの集中, ショートフローのバースト性,

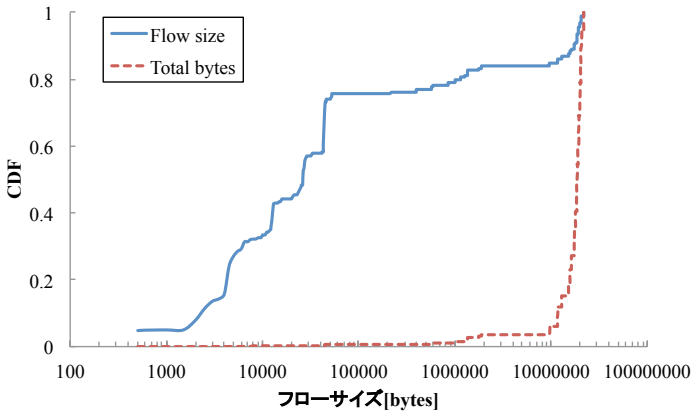

Fig.4.12　Distribution of Presto cluster in executing a job

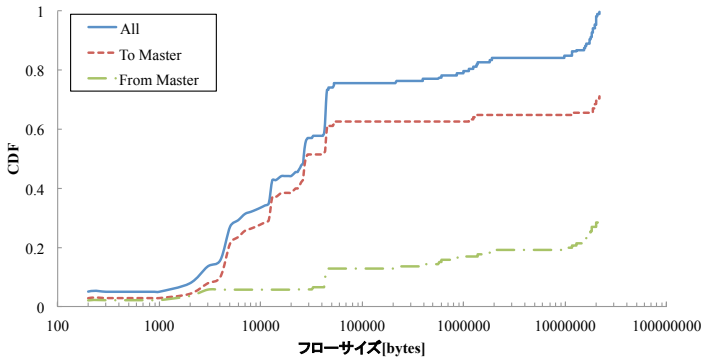

Fig.4.13　CDF of the traffic from master node in executing a job

そして長時間通信を行う Backgroung traffic の問題が生じていると考えられる. 従って, 管理ノードに対するトラフィックとして大きく二つのパターンを検討する必要がある.

1. ジョブ開始時のバースト性のあるショートフロートラフィック
2. アプリケーション性能に直接影響しない Backgroung traffic が通信している中で, 低レイテンシ通信が求められているショートフローの通信

こうしたトラフィックは主に, 複数の処理ノードから単一の管理ノードへの通信で発生する. そして, 中継スイッチ, エンドノードともに単一の NIC キューへとトラフィックが集中する事で, ボトルネックになり, ショートフローの遅延が問題となる [28].

4.3.1　性能障害

次に, これらのトラフィックパターンが引き起こす可能性のある二ヶ所のボトルネックについて検討する.

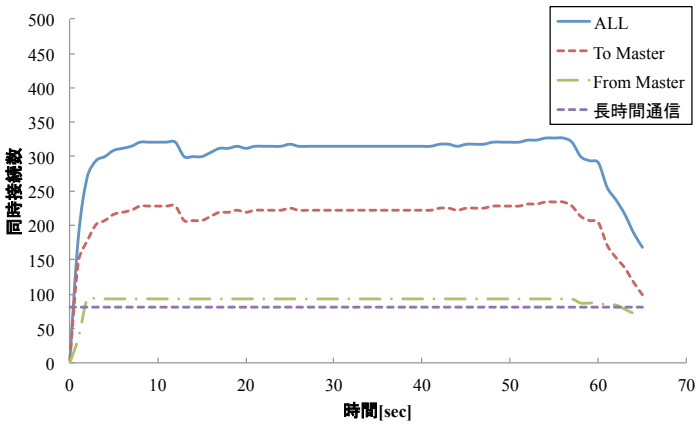

Fig.4.14　Distribution of the number of concurrent connections in executing a job

スイッチ - 性能障害

現在のスイッチ機器では複数のフローを多重に扱うための共有メモリを持ち, 共有メモリプールから MMU(Memory Management Unit) によって各インターフェースが利用できるメモリ量を動的に割り当てる事で, 複数の通信を公平に処理する事を目指す [23]. しかし, 比較的安価なスイッチでは制御できるメモリ量が制限されているため, 様々な性能障害を引き起こす [23].

Incast

Fig.4.15(a) に示すように, 短期間に一つのインターフェースへとフローが集中した場合, 用意されているキューが溢れ, 最悪の場合パケットロスを引き起こす. こうしたトラフィックは, §3.3 で示した partition-aggregate 構造によるもので, リクエストを受けた処理ノードが同期して一斉にレスポンスを返すことにより, そのレスポンスを集約して受け取るノードが接続しているスイッチでのインターフェースのキューサイズが大きくなり, 遅延, パケットロスを生じる.

Queue buildup

§3.3 で示したように, 並列分散処理のレスポンスには直接影響しない Backgroung traffic は, スイッチバッファにパケットロスを引き起こすほどの影響を及ぼし, そのインターフェースがボトルネックとなる可能性がある. Fig.4.15(b) に示すように, Backgroung traffic と Query traffic が同じインターフェースを利用する場合に, サイズの大きいフローによるショートフローのキューイング遅延が生じる. このとき, Query traffic には *Incast* とは異なり, バースト性は必要ではない.

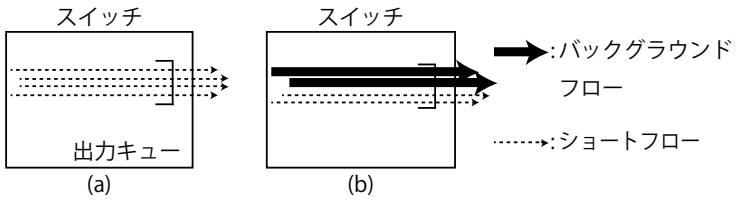

Fig.4.15　Bottoleneck in switch

エンドノード - 性能障害

今日の GbE(Gigabit Ethernet) 通信において, 割込み処理は大きなボトルネック要因の一つである. 例えば, 1GbE において 64 バイトフレームの最大受信可能数は, 毎秒約 150 万であり, 1 パケット受信する度に割り込み処理を行うと, CPU リソースが枯渇する. そのため, 割込み処理の回数を抑えることが必要であるが, その分レイテンシが上がる可能性があり, 互いのトレードオフを適切に対処し高い性能を得る必要がある. また, 今日の多くの CPU はマルチコアであり, CPU リソースを効率的に利用する事が求められている.

割込み処理

パケット受信の際の NIC によるハードウェア割込みは, 即座に受信処理を行う事ができ, キューイングの遅延を小さくする事ができる. しかし, 割込み処理が増えれば, その分オーバヘッドが大きくなり, OS の性能が劣化する. 割込み処理を扱う代表的な仕組みとして, ポーリング, interrupt coalescing がある.

ポーリングは NIC の割り込みを使わず, タイマーにより定期的に NIC の受信キューを監視することで, 割り込み負荷を軽減するソフトウェア技術である. しかし, NIC でパケットを受けてから即座に処理できない為, 遅延が発生する場合がある. 現在の Linux カーネルにおいては, NAPI により, 通信量が多く高負荷時にはポーリングが作用する [15]

interrupt coalescing は, 複数のパケット, あるいは一定期間待ってからをまとめて一度で割り込ませる事で, 割込み回数を減らすハードウェア技術である. しかし, ポーリングと同様, 即座に処理できない為, 遅延が発生する場合がある.

プロトコル処理

マルチコア環境においても基本的には一つの NIC の受信処理は 1 つの CPU でしか行えない. そのため, ハードウェアへのアプローチとして, 1 つの NIC に複数の受信キューを持たせて, 受信処理をそれぞれの CPU へ分散させている, Receive Side Scaling(RSS) がある [16]. しかし, 一般に複数受信キューを持ち, RSS 機能がある NIC は高価である [27]. そのため, 一つしか受信キューを持たない NIC であっても, 複数の CPU を分散させるソフトウェア技術として, RPS(Receive Packet Sterring) がある [18]. しかし RPS では, プロトコル処理とアプリケーション処理の CPU が異なる場合が生じ, その問題を最適化したのが RFS(Receive Flow Sterring) がある [17]. これらの技術により, CPU の複数のコアをより効率良く利用する事が

できる．また, プロトコル処理やアプリケーションでの処理については, RPS 等で複数の CPU へと分散させる事ができるが, その際の割込み処理についてはオーバヘッドが生じる可能性がある．

4.4　検証実験

この節では, 実機での実験を用いて, 低レイテンシでの通信が求められるショートフローに対して, バックグラウンドトラフィックが利用しているインタフェースを回避し, 適切に経路を選ぶ事で, 単一キューへの通信負荷の問題は解消され, ショートフローの FCT が改善できるという仮説の検証, また複数のキュー, 複数の経路を利用した経路切り替えによる改善手法に対する予備実験を行う．具体的には, 中継スイッチとエンドノードへのそれぞれの単一キューの負荷について, 複数の NIC を用いて分散させ, その効果を検証する．

4.4.1　実験環境

(1) 中継スイッチに対する負荷実験

ネットワークトポロジーには, 2 段で構成されたトポロジーを用いる．Fig.4.16 に, 用いたトポロジーを示す．ベンチマークトラフィックについては, 二つのペアに対してエンドノード同士の 1 対 1 通信を用いている．一方のペアに対しては, シミュレーションを実行している間, 継続してデータ転送 (バックグラウンドトラフィック) を行う．他方のペアに対しては, TCP による 70Kbyte のデータ転送 (ショートフロー) を毎 10ms 一様生起させ, 転送完了までにかかった時間 FCT(Flow Completion Time) を計測する．ショートフローのルーティングに関しては, Fig.4.16 に示す 3 つのパターンを用いて中継スイッチへのキューイング負荷の影響を検証する．

(2) エンドノードに対する負荷実験

ネットワークトポロジーには, 二つの NIC を持った二つエンドノード同士を L2 スイッチを介してそれぞれの NIC 毎に接続した．Fig.4.17 に, 用いたトポロジーを示す．ルーティングに関しては, それぞれの対をなす NIC 同士が通信を行う．ベンチマークトラフィックについては, エンドノード同士の 1 対 1 通信を用い, ショートフローとバックグラウンドフローを通信させる．バックグラウンドフローについては, ショートフローが通信している NIC ペアと同じものを使って共有して通信させるパターンとショートフローが通信している NIC ペアとは異なるペアの NIC を用いて通信を行うパターンの 2 パターンについて検証する．ショートフローは, TCP による 70Kbyte のデータ転送を毎 10ms 一様生起させ, 転送完了までにがかかった時間を計測している．バックグラウンドトラフィックは, シミュレーションを実行している間, 継続してデータ転送を行う．

Table.4.5 に用いた機器の詳細を示す．

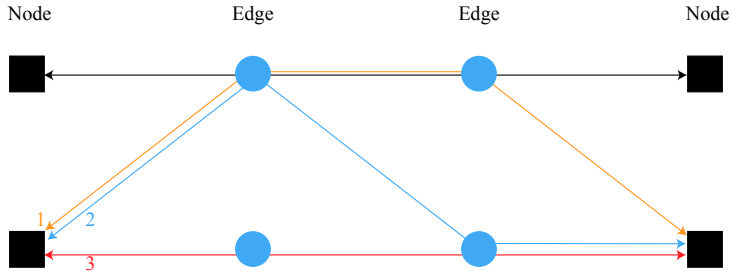

Fig.4.16　Topology in benchmark test for the NIC of the switch

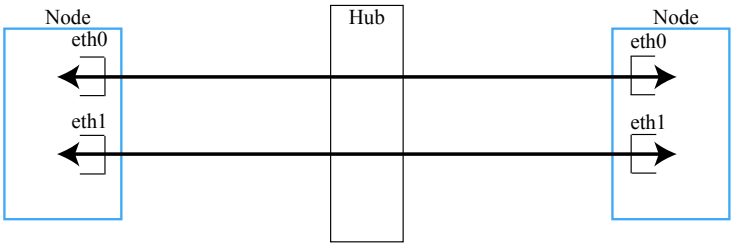

Fig.4.17　Topology in benchmark test for the NIC of the end-node

Table.4.5　Experimental parameter

項目	スペック
OS	Linux 3.13.0
CPU	Intel Xeon CPU L3426
メモリー	4GByte
NIC 対応ドライバ	e1000
スイッチ-実験 1	Catalyst 2940(100T)
スイッチ-実験 2	GS905L V2(1000T)
リンク	1GbE

4.4.2　実験結果

(1) 中継スイッチに対する負荷実験

Fig.4.18 に上記の実験環境での結果として, 70KB のショートフローの FCT とバックグラウンドフローの経路利用率を示す. FCT の箱ひげ図の上端には, 95 パーセンタイル値, 下端には最小値を用いている. 最大値でなく 95 パーセンタイル値を採用したのは, 特に遅延した下位 5 パーセントに着目することで, 遅延した割合の大きさを比較するためである. このメトリックにより, コンスタントにアプリケーション性能の出せるデータセンターネットワークの実現への指針となる. この結果から, ショートフローの通信が中継スイッチにおいて, バックグラウンドフローと経路およびインターフェースを共有した影響で, 一部のフローが大きく遅延し, その分散が大きくなっている事が分かる. 一方で, 同じスイッチでインタフェースは共有しな

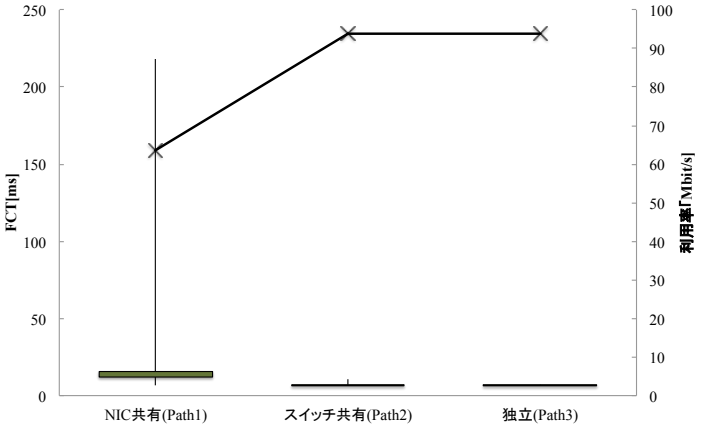

Fig.4.18　Link utilization and FCT of 70KB benchmark traffic for the switch

かったフローに対しては大きな影響はなかった．これは, 単一 NIC キューに対して二つのトラフィックが集中したことによる遅延の影響であると考えられる．その影響からバックグラウンドフローに対しても, スループットが下がっている事が分かる．

これらの事から, アプリケーション性能に直接影響しないバックグラウンドフローが通信している中で, 低レイテンシ通信が求められているショートフローの通信をする際, 中継スイッチでの利用するインタフェースが競合する場合, 単一のキューに対しトラフィックが集中し, 受信処理の割込みのオーバヘッドや, プロトコル処理の遅延の影響が生じたと考えられる．

(2) エンドノードに対する負荷実験

Fig.4.19 に上記の実験環境での結果として, 70KB のショートフローの FCT とそれぞれの経路の利用率を示す．FCT の箱ひげ図の上端には, 95 パーセンタイル値, 下端には最小値を用いている．この結果から, ショートフローの通信が, エンドノード間通信において, バックグラウンドフローと経路およびインターフェースを共有した影響で, FCT の分散が大きくなっている事が分かる．一方で, 経路, インタフェースは競合しなかったものの, バックグラウンドフローとショートフローが同時に通信を行ったことで若干の遅延の影響が生じ, 分散が大きくなっている．これは, 単一 NIC に対して二つのトラフィックが集中したことによる負荷分散の効果が得られたが, プロトコル処理以降の部分で, 複数のフローが同時に通信を行った事に対するオーバヘッドが生じたと考えられる．またバックグラウンドフローに着目すると, インターフェースを共有した場合においては, ショートフローだけでなくバックグラウンドフローにも遅延が生じ, スループットが低下している．

これらの事から, アプリケーション性能に直接影響しないバックグラウンドフローが通信している中で, バースト性のあるショートフロートラフィックの通信をする際, エンドノードに対して, 利用するインタフェースが競合する場合, 単一の NIC に対しトラフィックが集中する事で, 受信処理の割込みのオーバヘッドや, プロトコル処理の遅延の影響があると考えられる．

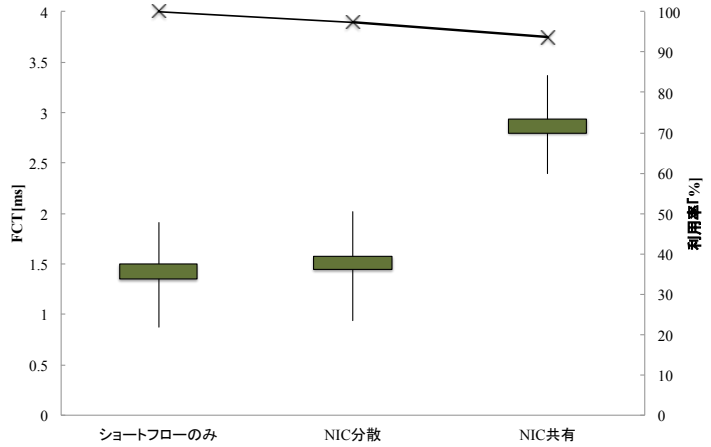

Fig.4.19　Link utilization and FCT of 70KB benchmark traffic for the end-node

4.4.3　考察

これらの解析結果から, エンドノード, スイッチに対する機能障害が引き起こる要因について述べ, 今後の改善手法の検討を行う. 大量の計算機資源をいかに効率的に利用するか, という課題を今日のデータセンターは抱えており, 並列分散処理アプリケーションを用いる事が一般的である. 今の並列分散処理システムが partition-aggrigation 構造である限り, 管理ノードや多段のクラスター構成であればアグリゲーターノードに対して, 処理ノードからのトラフィックが集中する問題は発生する. その結果, Queue buildup や Incast のような単一キューへの負荷集中の問題が中継スイッチやエンドノードに対して生じ, CPU 性能を効率的に引き出せず, 並列分散処理の性能が劣化する.

こうした遅延の影響を軽減する為には, 混雑時の通信量を抑える制御を行う, あるいは混雑時にも空いているリソースを効率良く利用する事が必要である. MPTCP による既存の計算機資源に対して複数 NIC を用いて性能向上を目指すように, 複数のフローを通信する際に異なる物理インターフェースを利用する事で, マルチコアを持つ CPU の効率的な利用につなげられる. すなわち, 複数のキューに対して通信を分散させるようなトラフィック制御により, 例えばレイテンシ志向なショートフローとスループット志向なバックグラウンドフローのような役割の異なるトラフィックを共存させ, 最適な通信の実現が可能であることが実験により明らかとなった. そのような物理的に複数の NIC によりマルチキューが介在する汎用的な機器で構成されているネットワークの中で, トラフィックをどのように制御するかという点については, スイッチやエンドノードの OS スタック等のどこで制御をするか, またどのようなアルゴリズムでそれを実現するかを検討する必要がある.

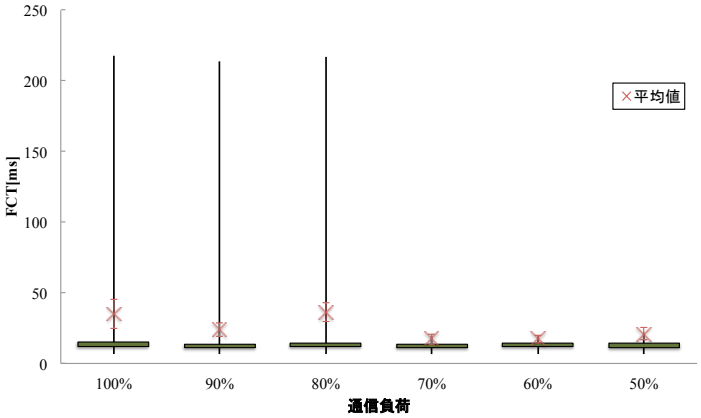

Fig.4.20　Impact of background flow for the switch

4.4.4　Directing result

経路混雑時のトラフィック制御について, 今後の指針となる一つの結果を示す. 提案手法では, 適切な経路を選んで通信を開始をする必要がある. 経路を決める際のメトリックとしては様々なものが考えられ, その中の一つとして経路の利用率がある. 経路利用率がショートフローの通信に対してどのような影響があるのか示した結果を Fig.4.20 に示す. 実験環境には, 中継スイッチに対する負荷実験と同様であり, バックグラウンドフローの負荷の度合いを変化させた. この結果から, 70 ～ 80 % の経路利用率の場合, 遅延するショートフロー発生する事が分かり, 適切な経路選択には利用率も一要因として考慮する必要がある.

第 5 章

提案手法

低レイテンシなデータセンターの研究動向と, 並列分散処理アプリケーションが生成する特有のトラフィックパターンが引き起こす機能障害をふまえて, 改善手法を提案する. 本提案手法では, 汎用的なネットワーク機器で構成されたマルチパスなデータセンターネットワークにおいて, エンドノードのみの改良によって, 長時間継続的にデータを転送し続けるトラフィックが通信している中でのサイズ小さいフロー通信に対する低遅延通信を達成することを目指す. この目的を達成するために, 提案手法では指向が異なるフローを区別する制御を行うことで, レイテンシ指向なフローについてキューイング遅延を小さくする通信を行う.

5.1 提案手法へのモチベーション

提案手法の動機となったのは, MPTCP を用いたデータセンターネットワークモデルである [9]. 今日のデータセンターネットワークは, FatTree トポロジーや Clos トポロジーに基づいた構成になっており [8, 46], 等コストな経路が複数存在している. 提案されたネットワークモデルでは, エンドノードが複数の NIC を持ち, MPTCP によって一つのフローの通信で複数の経路を同時に利用することでスループットを向上させる. このような複数経路の効率的利用には, IP ベースのルーティングで実現しており, それぞれのエンドノードが持つ IP アドレスのペアにより, 通信経路が決定する. 現在の MPTCP の実装では, TCP コネクション確立後に互いの IP アドレスを交換し, 新しいサブフローを形成する仕組みになっており, サイズの小さいフローの通信では, サブフローが形成されないまま通信が完結する問題がある. その結果, 一つの経路に通信が集中し, 中継スイッチのキューが圧迫され, フローの遅延が大きくなる可能性がある. 実際, §4.2 での解析でも, このコネクション確立の際に遅延が生じることが分かっており [21], どの経路を利用するかによって, 通信性能が大きく変わる.

複数経路の有効活用の手法として, ECMP によるフロー単位でのルーティングがある. ECMP ではパケットヘッダーの 5 タプルを用いたハッシュベースのルーティングやラウンドロビンによって, ランダムに経路を選択し, 負荷分散していく. この不完全なランダム性のために, ショートフローとロングフローが同一の経路に振り分けられることが起こり, ショート

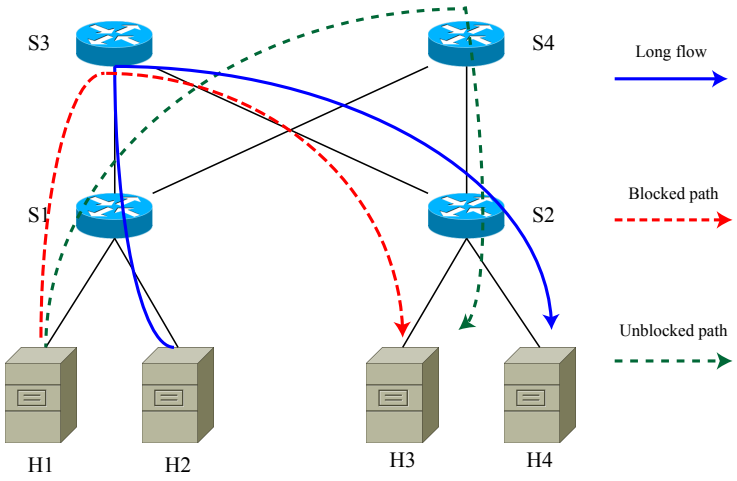

Fig.5.1　Scenario of queue buildup problem with multi-cost paths

フローは FCFS(First-Come-First-Serve) 方式のキューシステムによって, キューイング遅延が発生する. 今, Fig.5.1 のようなネットワークシナリオを考える. このトポロジーは $k = 4$ FatTree トポロジーでのある 1 ポッド内での通信を示している. スイッチ S1(S2) と接続されている二つのノードともう一方のノード間にはと二つの等しいコストの経路があり, H2 から H4 に対して S1-S3-S2 の経路を通り, ロングフローが通信している状況を考える. 今, H1 は H3 に対してショートフロー通信を行おうとしたとき, ECMP によるランダムルーティングにより, 0.5 の確率で同じ経路 S1-S3-S2 を選択してしまう. その結果, ロングフローによるキューイング遅延の影響を受ける. 実際, §4.4 に示すように実機を用いてこの同一経路を通る問題を検証したところ, ロングフローをうまく負荷分散した時と同一の経路で通信した時の 95 パーセンタイル値 FCT において, 約 10 倍程度性能が劣化することが分かった [22].

従って, **ロングフローは S1-S3-S2 の経路を通り, ショートフローは S1-S4-S2 の経路を通るように分散させて通信すること**が, マルチパス環境における複数経路の効率的な利用の理想的な状況である.

§2 に示した既存研究のように, データセンターネットワークでのショートフロー問題を解決するために多くの取り組みがされているが, 以下のような要求が考えられる.

1. スイッチに対して特殊な実装を施さないこと.
2. 既存のアプリケーションにも変更なく使えるようにすること.
3. ショートフローの FCT だけでなく, ロングフローのスループットについても改善すること.

1. については, §2 に示した多くの手法がスイッチに対して変更を加え, キューイング遅延が起こっている箇所から直接情報を得ることで, 改善を実現している. しかし, スイッチに対して変更を加えることで, 全てのスイッチに対してそれらを行う必要性があるため, 実環境への適

用に問題があると言える．実際, 様々な種類のネットワーク機器が混在している中ではなおさら実現困難である [40].

2. については, 既存研究の中で RepFlow は中継スイッチに変更を必要とせず, エンドノード間の変更のみで改善可能な手法であるが, 現状の実装だと, アプリケーションに対する変更が必要であるため, 既存のアプリケーションの変更が必要である [45]. そのため, 実現可能性の観点では有効であるとは言えない.

3. については, §4.4 に示すように, ロングフローとショートフローが同一の経路で通信を行った場合, ECMP による負荷分散手法では深刻な遅延の問題を引き起こすこととなる．§4.4 に示す検証実験では, ショートフローの性能劣化だけでなく, ロングフローのスループットについても約 30% 程度利用率が劣化する結果が得られた．RepFlow では, ショートフローに対して同一の通信を複製することによって, キューイング遅延のない経路を通る, ショートフローに対して最小の FCT を達成できるという手法であるが, 複製の結果, ロングフローのスループット性能も劣化することが考えられる.

以上を踏まえて本研究における位置付けを以下のように設定する.

1. エンドノード間のみのアプローチで改善を行う.
2. OS のみに変更を加え, 既存のアプリケーションに変更を加えず適用できる.
3. フローを指向ごとに区別し, それぞれに応じた改善を行うこと.

今, Fig.5.1 の環境において上記の前提の下, ロングフローとショートフローを区別し, 別々の経路を通るように分散させて通信することの実現を考える．しかし, 既存の技術では**フローの区別**と**それぞれの経路での通信**を達成することができない．フローの区別については, 基本的にはフローサイズの大きさで分類することができるが, OS から一つのフローを見た時に, 事前にどのくらいのデータが転送されるかを知る手段はない．たとえフローを区別できたとしても, それぞれのフローをどの経路に流すべきか判断するためのメトリックも既存のエンドノードでの技術にはない.

そこで本研究では, あらかじめ通信経路を決定するために, データセンターネットワーク内に指向毎にレーンを設けるデータセンターレーンモデルと, フローの指向を区別し, それぞれの経路を切り替えるためのアルゴリズムを提案する.

5.2　経路状況を考慮した経路切替による負荷分散手法の提案

本節では, 本研究における位置付けの下で, 指向毎にフローを区別し, それぞれの経路に切り替えて通信を行うことを実現するため, データセンターレーンモデルと経路状況を考慮した経路切替アルゴリズムを提案する.

5.2.1　データセンターレーンモデル

本小節では, 指向毎にフローを切り替えて制御するためのネットワークモデルとして, データセンターレーンモデルを示し, そのトポロジーと動作するアーキテクチャについて示す.

背景

今日のネットワーク機器はコモディティ化し, 安価な機器とある用途に特化した高価な機器のコストの差は明確となっており, データセンター事業者はその性能とコストのトレードオフについて考慮しなければならず, なるべくコストを抑えたい要求に対して頭を悩ませている [7]. 50 年以上前の電話網の形成の際にも, 多くの汎用的なスイッチを用いてより多くの端末に接続できるような設計が行われた [48]. そうした Clos トポロジーに基づき, Ethernet スイッチによって構成されるネットワークトポロジーの一つに FatTree トポロジーがある [7].

Full-bisectional なネットワークトポロジーに対する, 満たすべき要件として以下のようなものが考えられる [9].

- 局所性のないトラフィック
- すべてのホストが最大帯域活用できる
- どのアクセスリンクに対してもトラフィックの集中がない

今 FatTree トポロジーに対して, 並列分散処理システムを適用させることを考え, Hadoop Distributed File System(HDFS) によって同じラックの異なるホストにデータを複製し, MapReduce による map タスクがそれらに対して割り当てられるとする. つまり, 同一ポッド内でのトラフィックが発生する. MPTCP は基本的に複数の経路を持つ コア部分を経由するトラフィックに対して有効に働くが, このように経路選択肢が少ないローカルなトラフィックに対して, 十分に機能しない.

もしすべてのノードに対して MPTCP が実装されたとき, より MPTCP の特性を引き出すようなトポロジーを検討する必要がある. 具体的には, エンドノードと ToR(Top-of-Rack) スイッチ間のアクセスリンクでボトルネックが発生した時, 効率的な帯域の利用を考える. 手法として例えば, 1GEthernet リンクから 10G リンクへと変更することでより広帯域なネットワークを実現できるが, コスト面の問題がある. また, 従来の Sigle-path TCP では, 1 本のリンク容量以上の帯域を得ることはできないという制限がある. しかし MPTCP では, そのような制限はない. 実際, 近年のサーバ機器には基本的に複数のギガビットイーサネットインタフェースが搭載されており, エンドノードが二つの NIC を持つ際のトポロジーのプロトタイプとして, Fig.5.2 のような DHFT(Dual-Homing FatTree) トポロジーが提案されている [9].

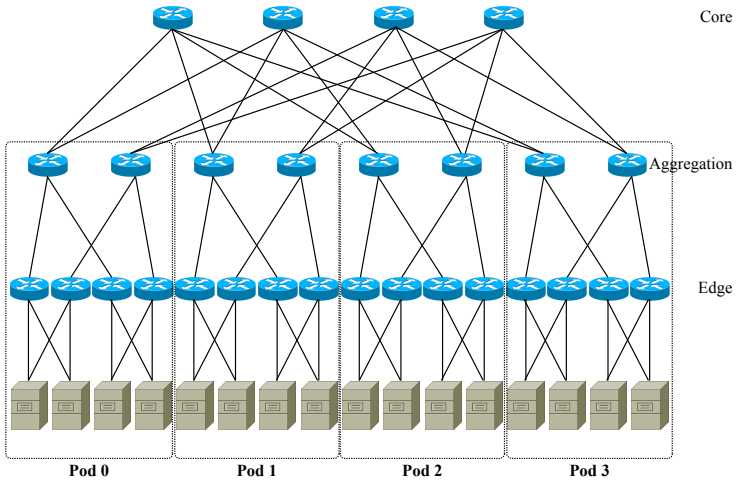

Fig.5.2　k=4 Dual-homing in the FatTree Topology

Motivation

§3 にて示したように, 現在の MPTCP の実装では, まず 3 ウェイ・ハンドシェイクによって TCP コネクションを確立し, その際にそれぞれの端末が MPTCP を利用可能かどうかのネゴシエーションを行う. MPTCP が利用可能な場合, 互いの持っている IP アドレスを交換し (add_address), その後サブフローを形成し, 複数の経路利用しながら通信を行う. その時 MPTCP の輻輳制御によって, それぞれの通信状況に合わせてウィンドウサイズが調整され, 通信される [25]. しかし, 今の MPTCP ではサイズの小さいフローについては, サブフローを形成する前に通信が完了する. そのため, たとえ TCP コネクションを形成するための経路が混雑した状況であっても, MPTCP によってその経路を回避する手段はなく, Fig.5.1 で示したような性能劣化を導いてしまい, マルチパス環境における複数経路の効率的な利用を実現できなくなる. ここで, 現状 MPTCP を用いて経路選択をする上での課題を以下に示す.

1. 基本的な IP ベースのルーティングでは, コネクションを確立する経路は毎回同じである.
2. あらかじめ経路を決定するには, 通信経路の輻輳状況を事前に把握しておく必要がある.

1. については IP ベースのルーティングでは基本的に宛先アドレスと送信元アドレスから決定されるので、コネクションを確立するアドレスペアが同じだと, それに従い通信経路も同じものが選択される. 異なる経路を選択するためには, 宛先アドレスとしてサーバ側の持つアドレスを事前に把握しておく必要があり, その上で ECMP やラウンドロビンのような分散手法の適用が考えられる. しかし先に示したように, これらの分散手法では完全な負荷分散は実現できず, あらかじめ通信状況を取得するなどのアプローチが必要である.

2. については, 本研究ではエンドノードに対する変更のみで改善を目指しているため, 遅延している中継スイッチから, 直接情報を得ることはできない. これまでの通信状況を把握しておくための取り組みとして, OpenFlow を用いた手法が提案されているが, 経路の統計情報を得るためのオーバーヘッドが発生するため, 現実的な解決策とは言えない [47].

これらを踏まえて, 用途の異なるフローに対して通信経路を切り替えるため, データセンターレーンモデルを提案する.

アーキテクチャ

複数のインタフェースを持つエンドノードに対して FatTree トポロジーを用いて, 指向毎のフローを切り替え制御の実現を目指す. 具体的には複数の等価コストな経路に対してレーンを定義し, 区別したフローに対して通信を行う経路を設置する.

このモデルの狙いは, スイッチから直接情報を得るような粒度の細かい制御を用いて遅延を回避するのではなく, あらかじめ通信すべき経路を区別しておくことで, それぞれの目的にあった通信を実現することにある. データセンターレーンモデルでは, 以下のような用途のレーンが設置される.

- *Query traffic* や *Short message traffic* のようになるべく短い FCT で通信したいショートフロートラフィックに対しては, レイテンシ指向なフローとみなし, 常に空いている状態に保たれたショートフローレーン SL(Short-flow Lane) を用いる.
- *Backgroung traffic* のようにより大きなスループットで通信したいロングフロートラフィックに対しては, スループット指向なトラフィックとみなし, 一般に複数の経路が設置してあるロングフローレーン LF(Long-flow lane) を用いる.

Fig.5.3 にネットワークレーンモデルを示す. 今, 互いのエンドノードはそれぞれのインタフェースごとに IP アドレスを持っているとし, それぞれ $LaneInfo$ が与えられているとする. この $LaneInfo$ の値は, 各ノードの OS に対して設定するパラメータであり, 各ノードが保有している送信元アドレスと 1 対 1 に紐付いている. これにより, 初めの TCP コネクションを形成する src : 10.1.0.1, dst : 10.2.0.1 の通信に対しては, Path1 でコネクションを形成し, 次に, 互いのアドレス (10.1.1.1, 10.2.1.1.) を交換し, サブフローを形成できる状態にする. サブフロー src : 10.1.1.1, dst : 10.2.1.1 の通信に対しては, Path2 を利用される. このように, それぞれのサブフローの通信においては, 異なる経路を形成し, 通信が行われる. 従って, 送信元アドレスが決定した時点で, 通信経路が決定するため, 送信元アドレスと紐付いている $LaneInfo$ が通信経路のレーンを示していることになる.

5.2.2　経路切り替えアルゴリズム

次に, §5.2.1 にて示した, データセンターレーンモデルに対して, フローを指向毎に区別し, 経路切り替えるアルゴリズムを示す.

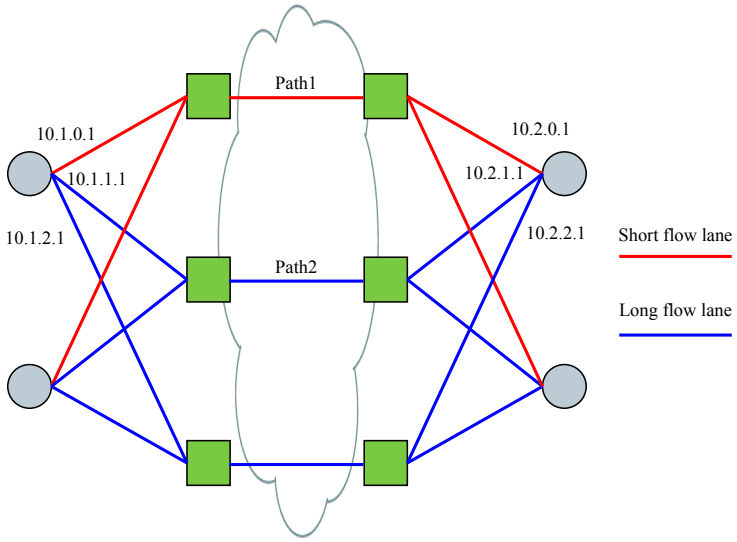

Fig.5.3　Datacenter Lane Network Model

モチベーション

今回のエンドノード OS のみのアプローチでは, フローサイズを用いた区別はアプリケーションに対する変更が必要となり, 現実的な解決方法ではない. そこで提案するアルゴリズムでは, 区別ためのメトリックとしてフロー持続時間を用いることで, SL を輻輳のない状態に保ち, フロー持続時間の長いものについては, LL に切り替えることで, それぞれの指向を最大限達成することを実現する. 特に SL については, ロングフローによるキューイング遅延が抑えられるため, FCT を短縮することができるように設計している.

先にも述べた通り, 通信経路は送信元アドレスと, 宛先アドレスによって決定され, 基本的には TCP コネクションを形成する通信経路は毎回同じものになり, また, フロー持続時間を用いて区別を行う手法であるために, 通信開始当初はフローの区別をすることができない. そのため, 提案するアルゴリズムでは, 以下のような動作により, 通信開始時にはすべてのフローがレイテンシ指向なフローと判断され, その後ロングフローの場合区別が行われる. 以下に通信経路の切り替えの流れを示す

1. SL に対して TCP コネクションを形成する
2. 互いの持つアドレスを交換し合い, サブフローを形成する
3. サブフローを形成した際, 送信元アドレスの $LaneInfo$ が LL であれば, cwnd=1 を代入する.
4. アルゴリズムがフローをロングフローであると判断すれば, SL のサブフローに対して cwnd=0 を代入し, LL のサブフローに対して設定されている輻輳制御のアルゴリズムを適用する

サブフローを形成した際に, cwnd=1 の小さい通信を行うことで経路の状況を取得する.

実装上の課題

上記のような経路切替アルゴリズムを実現するための実装上の課題がある.

· どのようにフローを区別するのか

先に示した通り, データセンターのアプリケーショントラフィックを考えたときに, それぞれの用途によって指向が異なる. 既存研究ではフローサイズ ($\geq 100KB$) をメトリックとして区別 [45] を行っていたが, 現状の実装だとカーネルにおいては通信開始時点でフローサイズを把握することはできない. そのため本提案手法では, 指向区別のための主なメトリックとして “フロー持続時間” を用いる.

· いつ経路を切り替えるのか

フロー持続時間のデッドライン時間を用いた最も単純な手法として, 例えば 300[ms] と設定することで, 300[ms] を越えた時点でフローの切り替えが行われるような制御が考えられる. しかし, SL が混雑しているにもかかわらず, SL で持続して通信を行うことや, LL が混雑しているにもかかわらず, デッドライン時間を超えると経路が切り替わる状況も想定されるため, このような単純な手法だと経路の状況によっては改善が見込めない. そのため, 経路状況に対応し経路を切り替えるためのメトリックとして “ リンクコスト” を定義し, リンクコストベースの経路切り替え手法を提案する.

リンクコストベースの経路切り替え手法

提案するリンクコストベースの経路切り替え手法によるトラフィック制御では, スイッチ, エンドノードに対してそれぞれのキューの混雑具合を考慮し, 経路を制御する. 本質的な狙いは, フローサイズに従って通信が終えるということであり, 具体的には, フローサイズの大きいロングフローが通信している中でショートフローが発生した場合, ロングフローが占有しているキューを避けた経路をショートフローが利用することでフロー完結時間の短縮化を実現するものである.

· RTT(Round-Trip Time)$\tau(t)$ モデル化

通信における遅延において, 様々な要因が考えられるが, 経路状況によって変動し. 最も遅延の影響が大きい要素として, キューイング遅延がある [63, 64]. 今クライアントノードでの ACK パケットを受けた時の RTT を $\tau(t)$ とする. この時, $d_{l,i}(t)$ はリンク i におけるリンク伝送遅延, $d_{p,i}(t)$ は伝搬遅延, $d_{q,i}(t)$ はキュー遅延を表している. この時, エンドノード間の経路として m 個のネットワーク機器を介する時, 以下のような式で RTT を表現できる.

$$\tau(t) = \sum_{i=1}^{m} \Big(d_{l,i}(t) + d_{p,i}(t) + d_{q,i}(t) \Big) \tag{5.1}$$

一般にセッションの間, ネットワーク内の経路は安定して通信を行うと想定することができ, 伝送遅延 $d_{l,i}(t)$ と伝搬遅延 $d_{p,i}(t)$ は定数としてみることができ, それらを併せて RTT に対する

バイアスとしてみることができる [63]. そのため, RTT が変化する最も大きな要因の一つは, 経路ごとのキューイング時間の変化であると考えられる.

キューイング時間の変化を最も単純なモデル式で表すと, ノード i におけるキュー長 $q_i(t)$ とリンク容量 c_i を用いて, ノード i での時間 t_i での $d_{q,i}(t)$ はキュー遅延を以下のように表現することができる.

$$d_{q,i}(t) = \frac{q_i(t_i)}{c_i} \tag{5.2}$$

これにより ACK パケットを受け取った時の時刻 t における RTT は以下のように簡単化される.

$$\tau(t) = \sum_{i=1}^{m} \frac{q_i(t_i)}{c_i} + bias \tag{5.3}$$

· SL, LL に対するリンクコスト

ここで τ_0 は最小 RTT, d_0 は最小キュー遅延を用いて, キューイング遅延の変化量である相対キュー遅延 $\nu(t)$ を以下のように表す.

$$\begin{aligned} \nu_i(t) &= \tau(t) - \tau_0 && (5.4)\\ &= \Big(\sum_{i=1}^{m} \frac{q_i(t_i)}{c_i} + bias\Big) - \Big(\sum_{i=1}^{n} \frac{q_i'(t_i)}{c_i} + bias\Big) && (5.5)\\ &= d_t - d_0 && (5.6) \end{aligned}$$

この $\nu(t)$ が通信経路の状況を表すと仮定し, これをパラメータとしたリンクパフォーマンス関数 [49] としてリンクコストを以下のように定義する. リンクコストについては, Davidson 関数を参考に導出した [49, 50].

$$\begin{cases} t_a^{SL} = t_0 \cdot \left\{1 + \alpha \cdot \nu(t)^{\beta}\right\} + \mathrm{sgn}(t - t_{deadline}) \cdot \gamma (t - t_{deadline})^{\delta} \\ t_a^{LL} = t_0 \cdot \left\{1 + \alpha \cdot \nu(t)^{\beta}\right\} \end{cases} \tag{5.7}$$

ここで, $t_a^{SL}(t), t_a^{LL}(t)$ はそれぞれ SL, LL のリンクコスト値であり, t_0 は最小リンクコスト, $sgn(t)$ は符号関数, $t_{deadline}$ はデッドライン時間, $\alpha \sim \delta$ はパラメータである. 一般に, このリンクコスト関数の変数, パラメータの性質は Table.5.1 のように表さるこのリンクパフォーマンス関数を用いた, 通信経路切り替えアルゴリズムを Algorithm 1 に示す. このアルゴリズムによって, サブフローが形成された際に, デッドライン時間 $t_{deadline}$ が設定され (Initialization), ACK パケットが届き, TCP スタックによる RTT の推定がされた時にコスト計算され (Calculating link-cost), それぞれのサブフローのリンクコストとの比較を行い (Judging Phase), SL のコストが LL よりも大きくなった場合, LL の方が有利に通信ができると判断され ($judge_flag \Leftarrow 1$), SL のウィンドウサイズを 0 に, LL のウィンドウサイズを設定

Table.5.1　The Natures of the parameter and variables in Link Cost Function

Variables, Parameter	Nature
α	リンクコスト関数の立ち上がりの速さを示す. α が小さい場合, キューイング遅延の影響が大きくなってもあまり RTT が増加しない
β	リンクコスト関数の傾きの度合いを示す. 経路の通信環境が悪化し ν が大きくなる領域は β の影響が支配的になる領域である. β が大きいと混雑に対する感度がよりアグレッシブな挙動を示す.
γ	デッドライン時間 $t_{deadline}$ へ近づく速さを示す. γ が小さい場合, デッドライン時間を設定することによる SL の優位性が小さくなる.
δ	リンクコスト関数のデッドライン時間 $t_{deadline}$ に近づく傾きの度合いを示す. δ が大きいとよりデッドライン時間に対する影響が強くなり, 通信状況よりもデッドライン時間に対する挙動の影響が大きくなる.
$sgn(t - t_{deadline})$	リンクコスト関数のデッドライン時間 $t_{deadline}$ を超えるまでの挙動の違いを示す. デッドライン時間を超えない時間では, 上に凸の関数をとなり, デッドライン時間を超えると下に凸の挙動を示し, デッドライン時間を超えた時の増加の速さが大きくなり, デッドライン時間に対する影響が大きくなる.

している輻輳制御アルゴリズムに従って算出された値を適用する. SL のコストが LL よりも小さいままの場合, SL のウィンドウサイズは輻輳制御に従い, LL のウィンドウサイズは 1 に設定される. これは, LL の用いる通信経路の状況を把握するため, 非常に小さなパケットを流し RTT を推定するためである.

5.3　提案手法の動作

5.3.1　利点

提案手法は, §4.3.1 に示す **Queue buildup** 性能障害を次のように解決する. 提案手法では, デッドライン時間, リンクコスト値を用いてロングフローであると判定された場合, 速やかに SL へとトラフィックが移行する. これにより, SL はレイテンシ指向なショートフローに対して, 通信経路を良好な状態に保つことができ, FCT を短縮化される. また, ロングフローに対しても, ショートフローによって輻輳状態となった LL を避けて通信することができるため, スループットの改善が期待できる.

Algorithm 1 Caluculating link-cost

▷ Initialzation
$t_{deadline} \Leftarrow now + THRESHOLD$
$\tau_0 \Leftarrow 0$
$t_0 \Leftarrow 1$
▷ Receiving ACK packet:Calculating link-cost
j = which subflow the ACK is
RTT_j = RTT estimation by TCP stack with smoothing
$measurement\_cost$ = Calculating link-cost
$cost_j \Leftarrow 3/4 * cost_j + 1/4 * measurement\_cost$ {Updating cost value with smoothing}
$base\_RTT$ = Updating if the RTT is the smallest one
$base\_cost$ = Updating if the cost is the smallest one
for $i = 0$ to $SUBFLOW\_NUMBER$ **do**
    $Min\_Cost'sLane \Leftarrow$ Detecting lane_info of minimum cost(LL or SL)
end for
▷ Judging phase
if $Min\_Cost'sLane$ is LL(Long-flow Lane) **then**
    $judge\_flag \Leftarrow 1$ {Change subflow's state}
end if
if $judge\_flag = 1$ **then**
    ▷ Switching phase
    if $lane\_info = 1$ **then**
        $cwnd \Leftarrow 0$ {This subflow in SL}
    else
        $cwnd \Leftarrow$ tcp_congestion_control {This subflow in LL}
    end if
else
    ▷ Keeping state phase
    if lane_info = 1 **then**
        $cwnd \Leftarrow$ tcp_congestion_control {This subflow in SL}
    else
        $cwnd \Leftarrow 1$ {This very small traffic in LL for probe }
    end if
end if

しかし実質的には, フローの種類が未知である通信発生時については, すべてのフローが SL で通信される. このため, ショートフローが短時間に大量に発生した場合や, ロングフローとショートフローが同時に通信を開始した場合には, 通信性能が劣化することが考えられる.

5.3.2　アルゴリズム実装の検討

提案手法では, 通信可能なすべてのサブフローに対して通信状況を表すリンクコストを計算し, 通信経路を切り替える際のメトリックとして用いる. その際, 最も重要な変数の一つは RTT である. 今回の提案手法では, RTT の推定には TCP スタックの tcp_rtt_estimator 用いる. この推定結果を用いてリンクコストの計算を行うが, RTT や前のパケットの到着時間の差分の揺らぎによっては, リンクコスト値も大きく変動する. そのため, リンクコストの平滑化を行う (Line 9).

また通信開始時の SL での通信の際にも, LL のリンクコストを計算する必要があるため, LL でのウィンドウサイズを 1 に設定し, 小さなトラフィックを流す (Line 31). これにより, すべての経路について経路状況を見ることができ, 経路状況に合わせた制御をすることができる.

第 6 章

評価

本章では, §5 にて提案した, データセンターレーンモデルにおける経路切り替え手法の性能評価を示す. 性能評価については, 3 つのパートに分かれている. 一つ目は, 提案手法の基本的な性能について, 提案手法が設定するパラメータによってどのような挙動を示すのかを評価する. 二つ目は, §5 にて示したデータセントラフィックがもたらす性能障害に対して, 提案手法を用いた結果どのように改善されたのかを評価する. そして最後に, データセンターネットワークが生成するトラフィックに対する評価として, トポロジー, トラフィック分布の点で実際のデータセンター環境に近いものを想定し, 性能評価を行う.

6.1 実験環境

この章での評価実験は全てシミュレーション上で行ったものである. シミュレーションには, パケットレベルでのシミュレーションが可能な ns-3 DCE(Direct Code Execution) [51] を用い, Linux カーネルに実装した提案手法を用いて評価実験を行った[*1]. それぞれの評価実験において, エンドノードにおける輻輳制御には TCP New Reno を用いた. 一般的に, データセンタースイッチは各ポート $128 - 256KB$ バッファが割り当てられている [8]. 今回の実験ではこの制約に従い, キュー長は $128KB$ の入出力バッファを用意している. また, 各スイッチのレイテンシとして, $25\mu s$ に設定した. この遅延時間も一般的なデータセンターの値となっており [8], 以下のような各遅延の影響に関する考察においても, 妥当であることがわかる [10]

- 伝送遅延 $12.24\mu s$: 1Gbps リンク 1530 バイトイーサネットフレームの遅延量.
- クロスバー遅延 $3.06\mu s$: クロスバー speedup-factor=4 における遅延量. HOL ブロッキングを抑えるために一般的にこの値が設定されている [52].
- 伝搬遅延 $0.476\mu s$: 銅線における遅延量 [53].
- 送受信処理遅延 $5\mu s$ [53]
- フォワーディング遅延 $4.224\mu s$ フォワーディングエンジンによる遅延量 ($25\mu s$ の残

[*1] 本シミュレーション実験でのソースコードは `https://github.com/ShogoFujii/mptcp` から入手できる.

り分)

6.1.1　評価メトリック

既存の取り組み [8,40,54] のように, 二つのメトリックを用いて評価を行う. 通信時間にデッドラインが設けられているような, レイテンシ指向のフローについては, FCT を用いる. FCT についても, 平均値だけでなく, テール部分に着目した 95 パーセンタイル FCT を導入する. テール部分で見られる大きな遅延については, OLDI アプリケーションのような, 各クエリーに対して最も遅いレスポンスを待ち, 最終的な処理時間となるようなタスクでの重要な指標となる [55]. より多くのデータを通信したいスループット指向のトラフィックについては, 各フロー毎のスループットを用いる.

6.1.2　比較対象

今回の評価実験では, Pure-MPTCP [13] と RepFlow [45] についても評価実験を行い, 提案手法との比較を行った. 比較する手法について以下にまとめる.

Pure-MPTCP [13]: 提案手法の評価におけるベースラインとして MPTCP-New Reno を用いる. $Initialwindow$ として 12 パケットに設定している. また, スイッチの入出力キューは DropTail に従うように設定している. その他の設定は, 既存の取り組みを参考にした [11,41]. 今回の評価では, MPTCP のソースコードとして MPTCP v0.88 を用いる. また, 今の MPTCP 実装では約 80KB 以下のフローに対しては複数の経路を用いず, Singlepath-TCP と同様の挙動となる.

RepFlow [45]: エンドーノードへのアプローチに焦点を当てた, 最先端のショートフロー短縮化技術として RepFlow を用いる. RepFlow はスイッチ, エンドノードのカーネルへの変更が不要な手法であり, 複数の経路を持つ今日のデータセンターネットワークに対して, 輻輳の起こっている経路を回避するため, 通信可能で等コストな経路にショートフロー通信を複製する. これにより, キューイング遅延の影響を確率的に減らすことができ, 結果的に最短時間で到着したフローによって短縮化することができる. 今回の評価では, 100KB 以下の全てのフローに対して複製を行う [45]. その他のパラメータは Pure-MPTCP と同じである.

デッドラインベースの経路切替手法:

§5 にて示した, デッドライン時間に基づいた単純な切り替え手法である. 切り替えのタイミングにはパラメータ $t_{deadline}$ を用いる. 評価実験では, $t_{deadline} = 300[ms]$ と設定する.

リンクコストベースの経路切替手法:

§5 にて示した各経路状況に対応した経路切り替え手法である. 切り替えのタイミングにはリンクコストを用いる. 基本的に評価実験では, $t_{deadline} = 300[ms]$, $\alpha = 1, \beta = 1, \gamma = 1, \delta = 1$ と設定する.

6.2　基本性能:スループット

初めに, 提案手法の基本的な性能として, 従来手法と提案手法の挙動の違いとして, 時系列でのスループットの変化について示す. 提案した二つの手法には, 予め設定するパラメータとして $t_{deadline}, \alpha \sim \delta$ があるが, $\alpha \sim \delta$ については固定し, $t_{deadline}$ を変化させることで経路切替による通信性能の影響について評価する. この評価ではエンドノード間の 1 対 1 の通信を想定しており, ベンチマークトラフィックには, iperf を用いて 2 秒間継続してデータ転送を行った. またトポロジーとしては, Fig.5.3 のようなエンドノード間の等価な通信経路が 3 つあり, そのうち 1 つを SL, 残りの 2 経路を LL に割り当てているものを用いる. すなわち, 提案手法の制御において通信開始時には, SL の最大 1 経路分と LL のウィンドウサイズ 1 分のスループットが可能であり, 経路切替が生じた後, 最大 2 経路分のスループットを出すことができる. なお, Pure-MPTCP では現状の実装上最大 2 経路分, RepFlow では基本的に TCP での適用を想定しているため, 最大 1 経路分のスループットを達成することが可能である.

Fig.6.1 に上記の評価結果として, 時系列ごとのスループットの変化を示す. この結果から, 提案手法の経路切替の際にデータ転送量の増加率に劣化が見られるものの, 約 100[ms] 以内には元の増加率に戻ることがわかる. これは, 提案手法では, 経路が切り替わる前の時点では SL はウィンドウサイズ 1 に設定しているため, 経路切替が発生した直後のウィンドウサイズ増加アルゴリズムとして, スロースタートの挙動を示す. そのため, スループット増加の傾きが緩やかになる. また, 200ms 付近での増加しきった後, 性能劣化する挙動については輻輳制御が働いていることと, サブフローでの通信においては従来の輻輳制御を無視し, 強制的に 1 に設定するため, 急激にスループットが下がり, 一経路分の通信性能へと落ち着く. また, デッドラインベースの切替手法では 300ms 時に切替が生じたが, リンクコストベースの切替手法では約 180ms 時に切替が生じた. これは, リンクコスト算出の際のキューイング遅延の項の影響を受け, より早く経路が切り替わるという判断がされたためである.

次に, デッドラインベースの切替手法について $t_{deadline}$ を $30, 300, 1000$ へ変化させた時の時系列でのスループットの変化の違いを Fig.6.2 に示す. この結果から, $t_{deadline}$ を変化させることで, パラメータに応じた切替が達成できていることがわかる. それぞれの挙動については先に述べたように, スロースタートによる輻輳制御と強制的にウィンドウサイズを 1 に設定した時の挙動を示している.

次に, リンクコストベースの切替手法について $t_{deadline}$ を $30, 300, 1000$ へ変化させた時の時系列でのスループットの変化の違いを Fig.6.3 に示す. この結果から, $t_{deadline}$ を変化させることで, パラメータに応じた切替が達成できていることがわかる. また, この手法では $t_{deadline}$ は主要なパラメータの一つではあるが, 式 5.7 が示すように, 他のリンクの状況に対応して経路切替時間はが変化する.

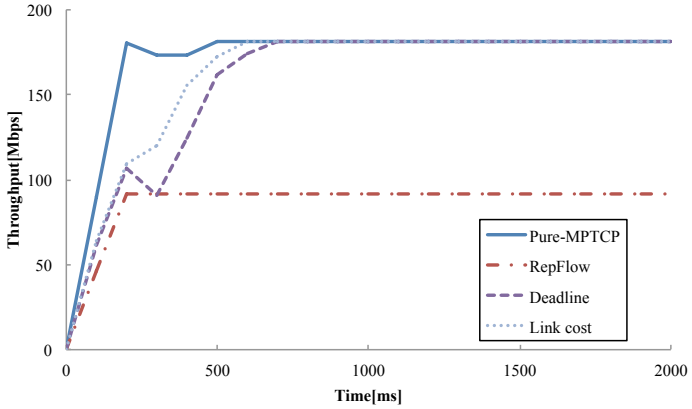

Fig.6.1　Basic performance comparisons

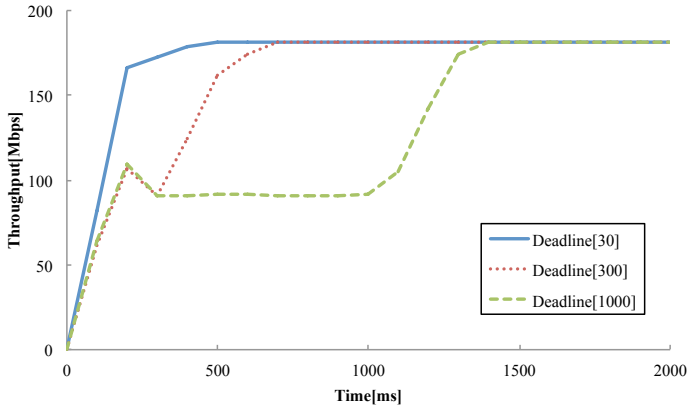

Fig.6.2　Basic performance path switching based on deadline

6.3　性能障害:Queue buildup に対する性能評価

次に §4.3.1 にて示した, Queue buildup に対する評価を行う. Queue buildup ではスイッチの一つのインタフェースに継続的にデータ転送を起こすロングフローとショートフローが混在し, ロングフローがキューを占有し, その結果, サイズの小さいショートフローが遅延する. この性能障害に対する提案手法の評価として Fig.6.4 に示す, k=4 DHFT ベースのトポロジーを用いる. このトポロジーでは, エンドノード間の 2 つの等価な通信経路にそれぞれ SL と LL に割り当てている. また 8 台のエンドノードに対して, 25% のノード (2) をバックエンドノード, 75% のノード (6) をフロントエンドノードと想定する. ベンチマークトラフィックには, バックエンドノードに対し, 実験時間中継続してデータ転送を行うトラフィックを用いる. また, 全てのバックエンドノードへの通信負荷を与えることで, Queue buildup の影響の大きさを変化

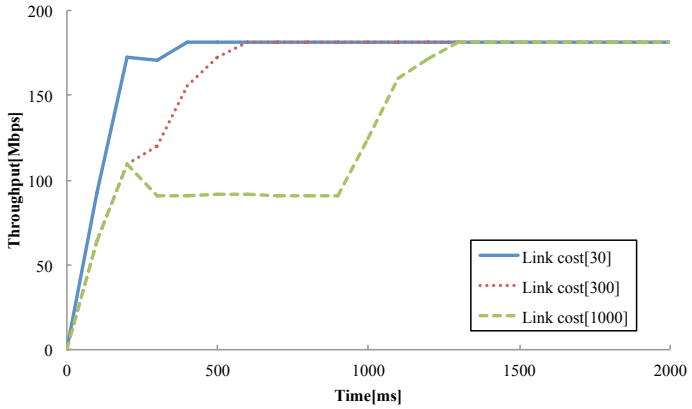

Fig.6.3　Basic performance path switching based on link-cost

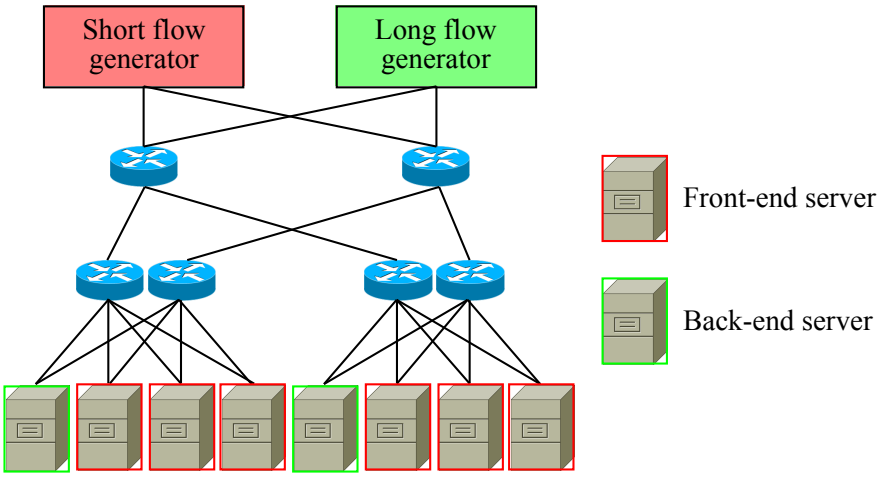

Fig.6.4　Topolgy for impairment of Queue buildup evaluation

させる．通信負荷については, Sender ノードの数を変化させ, 負荷の度合いに対するショートフロー性能の評価を行う．さらに, 全てのフロントエンドノードに対し, ロングフロー通信開始よりも前の時間から $64KB$ のショートフローを 200ms に 1 回ポアソン生起させる．これにより, 二種類のトラフィックが同時に開始することとなる．この時のショートフローの FCT を評価することで, 提案手法の Queue buildup に対する改善を評価する．評価の際のメトリックとしてはショートフローの平均 FCT と, 95 パーセンタイル値を用いる．特に遅延した下位 5 パーセントに着目することで, 遅延した割合の大きさを比較することができ, 本研究の目的である, コンスタントにアプリケーション性能の出せるデータセンターネットワークの実現への指針となる．このシミュレーションでは, ポアソン生起を用いたショートフローの発生間隔についてランダム性が発生し, 100 回シミュレーションを繰り返し行った．また, 提案手法のパラメータには, $t_{deadline} = 300[ms]$, $\alpha = 1, \beta = 1, \gamma = 1, \delta = 1$ と設定する．

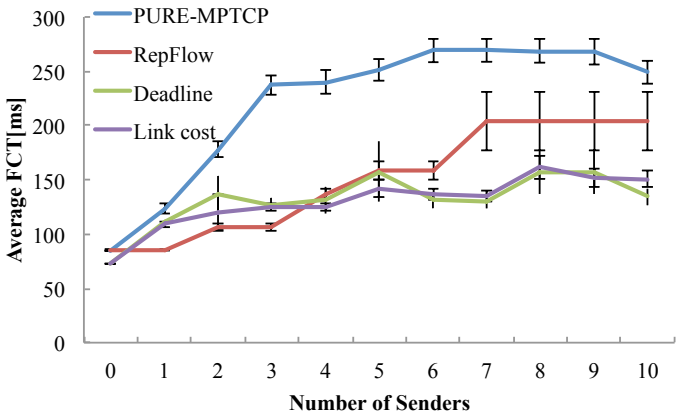

Fig.6.5　Short flows average FCT in Queue buidup

Fig.6.5 に Queue buildup での 64KB ショートフローの平均 FCT を示す．横軸は，バックエンドノードに対する通信負荷の度合いを示す Sender ノードの数，縦軸には平均 FCT を示す．この結果から，提案手法ではショートフローに対するロングフロー通信負荷の影響を軽減させていることがわかる．これはレーンモデルと経路切替手法によって，バックエンドに対するロングフローは LL へと切り替えられ，フロントエンドに対するショートフローは良好な経路状況である SL で通信することができるため，FCT が短縮化することができていることが理由である．一方，Pure-MPTCP では Sender ノードが 1 台の時でも，2 経路を同時に利用しながら通信するため，通信負荷が小さい段階から大きく遅延していることがわかる．また，RepFlow ではロングフローは 1 経路分しか利用しないため，通信負荷が小さい段階では，輻輳の影響が小さく，平均 FCT も小さいが，通信負荷を大きくしていくと，2 経路に対する利用率が大きくなり，FCT が遅延する．

次に，Fig.6.6 に Queue buildup での 64KB ショートフローの 95 パーセンタイル FCT を示す．横軸は，バックエンドノードに対する通信負荷の度合いを示す Sender ノードの数，縦軸には 95 パーセンタイル FCT を示している．この結果から，二つの提案手法では，大きな遅延が生じたフローの割合についても改善できていることがわかり，特にリンクコストベースの切替手法では，デッドラインベースのものと比べ，より大きく改善できている．提案手法において，特に遅延が生じるのはロングフローが発生した直後の SL を用いて通信を行う時間に，ショートフローが発生した場合である．デッドラインベースの経路切替手法で経路の状況によらず，$t_{deadline}$ 時間中は必ず SL にて通信が行われ，その際にショートフローが遅延する．これに対し，リンクコストベースの経路切替手法では，経路状況に合わせて切替時間が変化するため，SL で通信を行っているレイテンシ指向なフローに対する影響を抑制することで，FCT 改善の改善を達成した．一方で，RepFlow については，通信負荷が小さい時には遅延しているフローの割合は小さい．これは，提案された RepFlow では TCP 通信を想定しているため，バックエンドノードに対するトラフィックも 1 経路分の負荷しかかけることができない．そのため，他

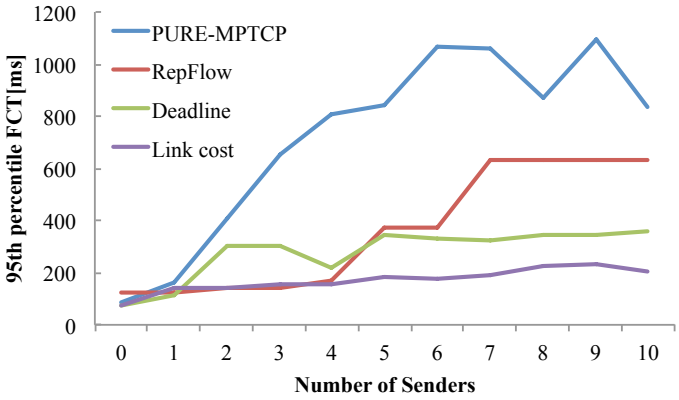

Fig.6.6　Short flows 95^{th} FCT in Queue buidup

の MPTCP による手法と比較して約半分の通信負荷となっており, 通信負荷が小さい時には RepFlow によるフローの複製が有効に機能する. しかし, 通信負荷が大きくなるとキューイング遅延が大きい経路を選択せざるを得なくなるため, 遅延が生じる. そのような状況においても, RepFlow による手法では Pure-MPTCP よりは改善が見られる.

今回の評価実験では, ショートフローが通信している中でロングフローが通信開始するため, SL の経路状況は LL よりも悪化していると考えられる. そのためリンクコストベースの経路切替手法の挙動として, 切替時間がデッドライン時間よりも短くなり, LL へ切り替わる時間が早くなる. その結果, SL にてショートフローとロングフローが混在する時間が短くなり, 遅延したフローの割合が小さくなる. 一方で, 二種類のフローを分類して通信する提案手法, 特に経路状況に対応して通信することができるリンクコストベースの経路切替手法では大きく改善できたのに対し, Pure-MPTCP と RepFlow については, それぞれ二種類のトラフィックを混在させ通信を行うため, 遅延する割合が大きくなり, ショートフロー通信の性能が劣化する.

6.4　実環境でのトラフィックを想定した性能評価

最後に, 実際のデータセンター環境を想定した評価実験を行った. この評価ではトポロジーとして, k=4 DHFT を用いる. このトポロジーでは, エンドノード間の 2 つの等価な通信経路にそれぞれ SL と LL に割り当てている. また 16 台のエンドノードに対して, 先ほどの評価実験と同様に 25% のノード (4) をバックエンドノード, 75% のノード (12) をフロントエンドノードから構成されている環境を考える.

ベンチマークトラフィックとしては, 二つのデータセンターアプリケーションを想定する. 一つは, ウェブ検索を想定したトラフィック [8] である. このトラフィックでは, 全体の 95% 以上のトラフィック量が, 全体割合の内 30% 以下である 1MB 以上のデータによってもたらされているという特徴がある. もう一方は, データマイニングトラフィックである [46]. このトラ

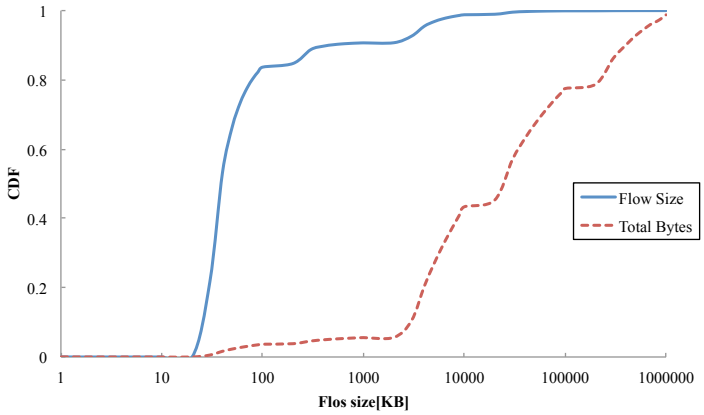

Fig.6.7　Web search workload

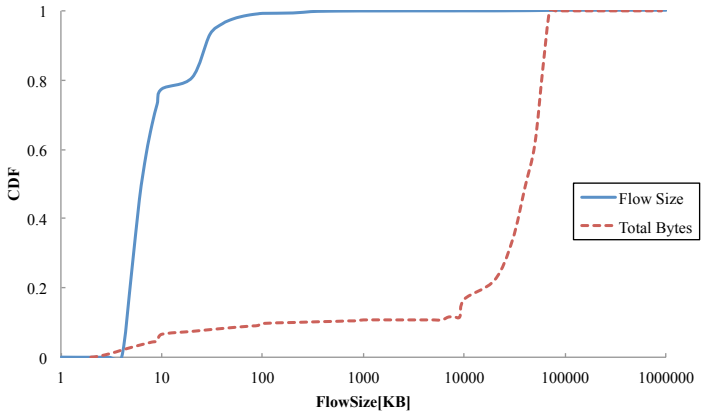

Fig.6.8　Data mining workload

フィックでは, 全体の 95% 以上のトラフィック量が, 3.6% 以下の割合であ 35MB 以上のデータによってもたらされており, 一方で, 80% 以上の割合が, 10KB 以下のフローである. これら二つのトラフィックは大きなテール状の分布をしており, ショートフローとロングフローが混在している. 今回の評価実験では, これらのトラフィックに対しテール状の確率分布を表現できるパレート分布を用いて生成した. Fig.6.7, 6.8 にそれぞれのトラフィックの分布を示す. また, Table.6.1 にそれぞれのトラフィックを生成するための統計情報を示す. さらに, バックエンドノードに対するトラフィックによる通信負荷を変化させた時の評価を行う. 具体的には, バックエンド, フロントエンドそれぞれに対し, 通信対象となるノードの割合を $Load$ とし, これを $Load = 0.1 \sim 0.8$ まで変化させる.

Fig.6.9 にウェブ検索トラフィックでのショートフロー (0, 100KB] の平均 FCT を示す. 横軸は, バックエンドノードに対する通信負荷の割合 $Load$, 縦軸には平均 FCT を示す. ウェブ検索トラフィックの特徴として, バックエンドへのトラフィックが平均フローサイズ 10MB の

Table.6.1　The detail of websearch and data mining workload with statistic approarch

Workload for	Websearch	Data-mining
Front-end	Pareto-distribution $a = 1.5, \mu = 50[KB]$	Pareto-distribution $a = 1.5, \mu = 10[KB]$
Back-end	Pareto-distribution $a = 1.2, \mu = 10[MB]$	800[MB]/n bytes from n differrent nodes

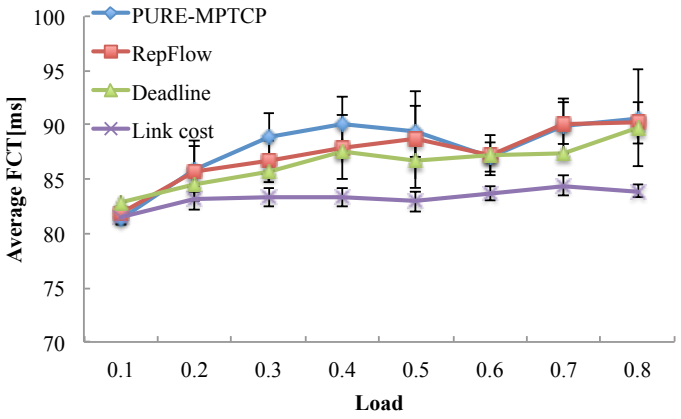

Fig.6.9　Short flows(0,100KB] average FCT in Websearch workload

データ転送が定期的に生じるということがある．これは, 非常に大きなロングフローが継続的に通信するのではなく, 細かく通信が発生することを意味している．デッドラインベースの経路切替手法では, たとえ SL が混雑していても, デッドライン時間に達するまでは LL に切り替わることはできない．そのため今回のように, SL でショートフローが頻繁に発生している中で, ロングフローが発生する状況には遅延が生じる．一方, リンクコストベースの経路切替手法ではそういった経路の変化に対し, 経路切替のタイミングを変化させて, ショートフローの遅延を最小限に抑えることができる．そのため, この結果からわかるように, リンクコストベースの経路切替手法では FCT が改善できている．しかし, Fig.6.9 の縦軸の範囲からもわかるように, 基本的にフロントエンドへのフローのサイズが小さいので, 遅延の影響を受ける確率は小さくなり, 大部分のフローが遅延なく通信を完了する．そのため提案手法による優位性はあるものの, 改善の度合いは小さい.

次に, Fig.6.10 にウェブ検索トラフィックでのショートフロー (0, 100KB] の 99 パーセンタイル FCT を示す．横軸はバックエンドノードに対する通信負荷の割合 *Load*, 縦軸には 99 パーセンタイル FCT を示している．この結果から提案手法では特に遅延したフローの割合についても, 従来手法より改善することができていることがわかる．特に通信負荷を大きくしてった際に, その傾向が顕著に現れる．これは Pure-MPTCP と RepFlow には, ショートフローとロングフローを公平に扱って通信を行うため, 同一の経路を利用し, Queue buildup の

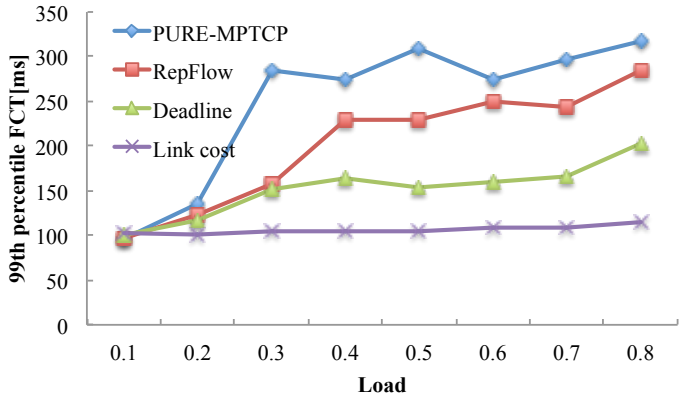

Fig.6.10　Short flows(0,100KB] 99^{th} FCT in Websearch workload

ような性能障害が生じるためである．RepFlow は通信負荷が小さい時には, Queue buildup の影響が小さい経路を避けて通信することができるため, Pure-MPTCP よりも改善することができているが, 通信負荷を大きくしていくとその改善の効果も小さくなっていく．一方, 提案手法ではデータセンターレーンモデルによりロングフローとショートフローを区別して通信を制御できるため, 従来手法よりもロングフローの影響を緩和できている．しかし, 経路切替の実装上, 通信開始直後には SL で通信するショートフローに対し, 遅延を生じさせてしまう．これにより, デッドラインベースの切替手法では, 通信負荷が大きくなるにつれ, ショートフローへの Queue buildup による遅延の影響が大きくなる．それに対し, リンクコストベースの切替手法では, SL に対する影響を抑えることができ, 通信負荷を大きくしても, 遅延が生じるフローの割合を抑えることができる．

次に Fig.6.11 にデータマイニングトラフィックでのショートフロー (0, 100KB] の平均 FCT を示す．横軸はバックエンドノードに対する通信負荷の割合 *Load*, 縦軸には平均 FCT を示す．データマイニングトラフィックトラフィックの特徴として, バックエンドへのトラフィックが非常に大きなデータ転送が継続的に行われる点がある．これは, バックエンドに対して通信されるフローの数そのものが少ないことを示しており, 提案手法での通信開始直後の SL に対する影響も小さいということである．また, フロントエンドへのトラフィックでは, フローサイズが比較的小さい．そのため, ラウンドトリップ回数も少なくなり, 遅延が生じる可能性も小さくなるため, どの手法においても大部分のフローについては遅延の影響なく通信を完結している．

最後に, Fig.6.12 にデータマイニングトラフィックでのショートフローの 99 パーセンタイル FCT を示す．横軸はバックエンドノードに対する通信負荷の割合 *Load*, 縦軸には 99 パーセンタイル FCT を示している．この結果から, 二つの提案手法では従来手法と比較し, 大きな遅延が生じたフローの割合について改善できていることがわかる．データマイニングトラフィックのバックエンドへのフローに対し, フロー自体は極めて大きなものであるものの, その数は小さいために通信開始時の SL への影響が深刻ではなくなっている．したがって, データマイニン

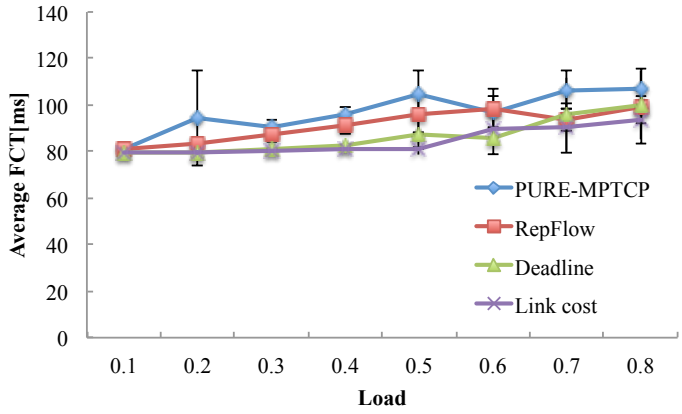

Fig.6.11　Short flows(0,100KB] average FCT in Datamining workload

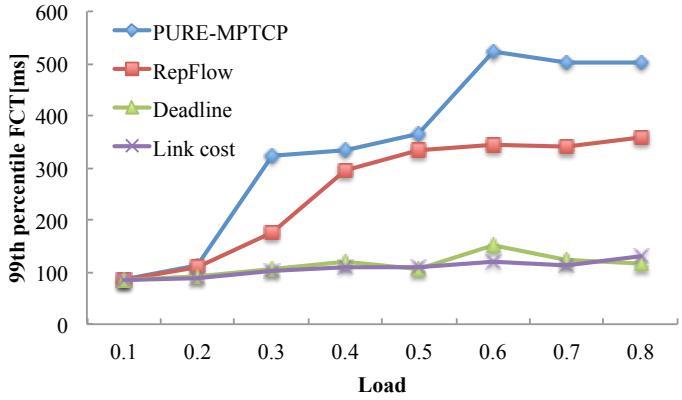

Fig.6.12　Short flows(0,100KB] 99^{th} FCT in Datamining workload

グトラフィックのような粗い粒度でロングフローが発生するようなトラフィックに対しては, データセンターレーンモデルによるフローの区別によって大きな改善は見られるものの, 経路状況に合わせたリンクコストベースの経路切替手法による改善を期待できる部分が小さいので, 両者における改善の差は小さい.

6.5　ロングフローに対する影響

今回の提案手法では, ロングフローとショートフローを区別して通信を行うことで, ショートフローに対するキューイング遅延の影響を抑え FCT を改善するものであった. 従来ではすべてのフローが公平に扱われるがため, 同一の経路を共有し, ショートフローに対して遅延が生じていた. この時, 競合したロングフローに対してもやはり, 通信性能の劣化があり, その結果スループットを落としていた [22]. しかし, 提案したデータセンターレーンモデルによって,

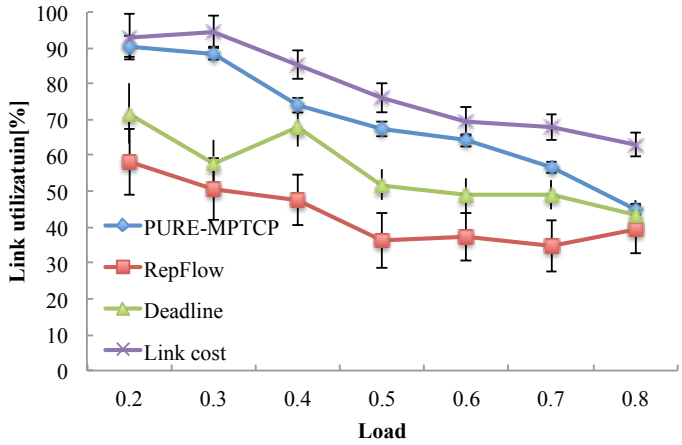

Fig.6.13　Link utilizatiin of long flow for back-end nodes in Websearch workload

フロー指向毎に切り分けて通信することができるようになったため, ショートフローだけでなく, ロングフローに対しても改善が期待できる. そこで, §6.4 のウェブ検索トラフィックにおけるバックエンドノードへのトラフィックに対し, 各フローのスループットを測定した. Fig.6.13 にその結果を示す. 縦軸は各フローのスループットを 1 経路の帯域で正規化したリンク利用率, 横軸はバックエンドノードに対する通信負荷の割合 *Load* を示している. RepFlow はその手法の特性上, 複製されたショートフローがロングフローに対してリンクを共有することで通信性能を劣化させてしまう. 一方で, 提案手法ではデータセンターレーンモデルによって, ショートフローと経路を共有することはないため, 通信負荷の増加に伴い他のロングフローによる影響はあるものの, RepFlow と比べるとその影響は小さい. また提案手法では SL と LL それぞれ 1 経路分の通信性能しか期待できないため, 通信開始当初にショートフローによる影響を受けた Deadline ベースの切替手法と比べ, 2 経路 riyoudekiru Pure-MPTCP ではより大きなスループットを達成している. リンクコストベースの切替手法では, 通信状況に応じ, そのフローが通信するのに有利な選択を行うことができる. そのためロングフローにおいても通信性能の改善が見られた.

6.6　まとめ

データセンターレーンモデルでは, サイズが小さく素早く通信を完了したいショートフローとサイズが大きく可能な限りたくさんのでデータを転送したいロングフローとを区別し, 個別に通信を行うよう制御することにより, Queue buildup のようにロングフローがショートフロー通信を劣化させる問題を解消している. また, エンドノード OS へのアプローチを取る以上, 通信開始当初のフローが区別できない状態において, ロングフローが SL に対し, キューイング遅延を引き起こす可能性がある. この問題に対し, リンクコストベースの経路切替手法で

は, 単純なデッドラーンベースの切替手法とは異なり, 経路の環境に合わせた制御を実現しており, 通信開始当初の SL への影響を抑制することができる.

以上の提案手法は, キューイング遅延が直接生じているスイッチに対する変更が不要で, エンドノードの OS に対する変更のみで実現ができる. 同様の指針で設計された手法として RepFlow があり, キューイング遅延が大きい経路を避けて通信することができ, 既存の MPTCP よりも通信性能改善することが可能である. しかし, それぞれの要求の異なるショートフローとロングフローが混在し, それらを公平に扱い通信するために, 通信負荷が大きくなった際に通信性能が劣化する. これに比べ, 提案手法では指向を区別し,　それぞれのフローが干渉しないような制御を行うことで, 通信性能劣化の問題を解決することができた. また, エンドノード OS に対する変更のみでの改善, また実際のデータセンターでのトラフィックを想定した評価によって, 提案手法によるショートフローの改善の実用性についても評価することができた.

第 7 章

考察

本章では, 実験環境に関する考察, 提案手法に関する考察および実環境への適用を考えた時の課題を考察する.

7.1 提案手法についての考察

本節では, 提案手法のパラメータに関してどのように挙動が変化するのかを示し, どのようにパラメータを背絵亭するべきなのかについて議論する.

7.1.1 リンクコストパラメータに関する考察

前述の評価実験において, リンクコストベースの経路切替手法が経路状況に対応し, 経路が切り替わるタイミングが変化することによって, デッドラインベースの単純な経路切替手法における問題を解消することを示した. 式 (5.7) に示すリンクコストは, 大きく二つの項から構成されている. 一つは $\alpha \cdot \nu(t)^{\beta}$ であり, 経路状況によって変化する項である. もう一方は, $\gamma(t - t_{deadline})^{\delta}$ であり, デッドライン時間に関する項である. それぞれの項にはパラメータ $t_{deadline}, \alpha \sim \delta$ があり, 今回の評価実験では, それぞれ $t_{deadline} = 300[ms]$, $\alpha = 1, \beta = 1, \gamma = 1, \delta = 1$ と設定した. しかし, 二つの項の変数を考えると, それぞれのパラメータによって経路が切り替わるタイミングが変動する.

今, リンクコストベースの切替手法における各パラメータ毎の挙動の違いを評価する. この評価ではエンドノード間の 1 対 1 の通信を想定しており, ベンチマークトラフィックには, iperf を用いて継続してデータ転送を行った. また, トポロジーとして k=4 DHFT を用いる. このトポロジーでは, エンドノード間の等価な通信経路が 2 つあり, それぞれ SL と LL に割り当てている. そしてそれぞれのレーンに対し通信負荷を与えることで経路の状況を変化させる. SL に対する通信負荷には, 64KB のフローを 200ms に 1 回ポアソン生起するトラフィックを用いる. LL に対する通信負荷には, 実験時間中継続してデータ転送を行うトラフィックを用いる. それぞれの通信負荷については, Sender ノードの数を変化させ, 負荷の度合いに対する評価を行う. なお, パラメータ $t_{deadline}$ は $300[ms]$ を設定している.

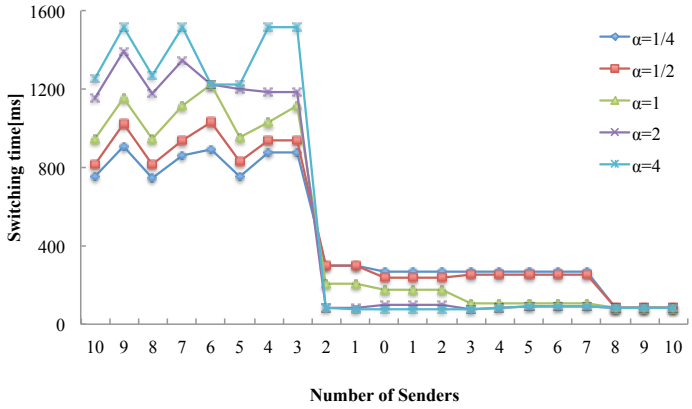

Fig.7.1　Basic performance path switching based on link-cost according each parameter α

経路状況に対する挙動に関わるパラメータ α, β を変化させたときの経路切替時間の移り変わりを Fig.7.1 に示し, 横軸は SL, LL への通信負荷の強度を表す Sender ノードの数, 縦軸には通信が発生したときの時間を 0 としたときの経路切替時間を示している. この結果から, パラメータ α が大きくなればなるほど, 経路状況の変化にアグレッシブに反応し, また値が小さいとパラメータ $t_{deadline}$ の影響を大きく受けることとなり, 設計目的の期待の狙い通りの結果が得られた. 実際, SL での Sender ノードが増えれば増えるほど, SL でのキューイング遅延が大きくなり, SL のリンクコストの値が大きくなる. これは, SL で通信をするよりも LL で通信を行ったほうが有利であると判断された結果であり, このため切替時間が短くなる. また, LL での Sender ノードが増えれば増えるほど, LL でのキューイング遅延が大きくなり, LL のリンクコストが大きくなる. これは, LL へ切り替えて通信するよりも SL に留まって通信を継続した方が有利であると判断された結果であり, このため切替時間が長くなる.

次にデッドラインに対する挙動に関わるパラメータ γ, δ を変化させたときの経路切替時間の移り変わりを Fig.7.2 に示す. この結果から, パラメータ γ が大きくなればなるほど, 経路状況よりも $t_{deadline}$ に大きく依存することとなり, 設計の狙い通りの結果が得られた. 実際, LL での Sender ノードが増えれば増えるほど, LL でのキューイング遅延が大きくなり, SL に留まるような判断がされるが, $t_{deadline}$ を超えれば経路切替が発生しやすくなるような挙動をする. そのため, 切替時間が $t_{deadline}$ を大きく超えるような挙動は発生しない.

今回の評価の際にはべき乗部分にあたる β, δ については固定して評価を行った. これはリンクコスト計算においてべき乗の影響は大きく, 一方を大きくするとその項が支配的に作用するためである. リンクコストの定義の意味を考慮して, $\beta = \delta$ と設定し, それぞれの項の次元をを揃えることが望ましいと考える.

$t_{deadline}$ については, 対象となるアプリケーションが最低限満たすべき通信時間の値を設定するべきである. 一般に, 近年のデータセンターにおける並列分散処理アプリケーションでは, 300ms 以内に通信を終えるべき [43] であるとしており, 評価実験においてもその値に従った.

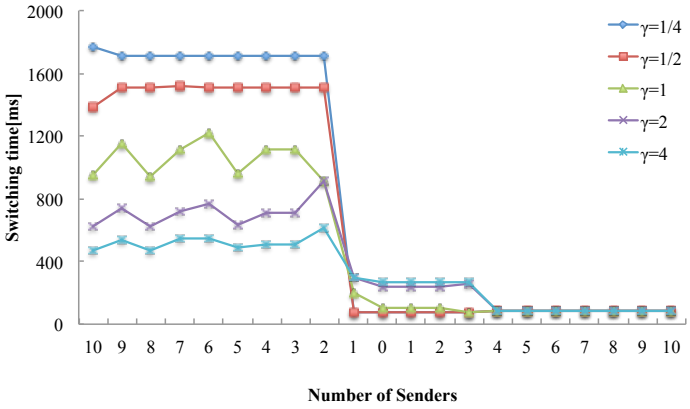

Fig.7.2　Basic performance path switching based on link-cost according each parameter γ

α, γ については, 通信状況に対し機敏に反応させたければ, α を大きく, γ を小さく設定するべきである. しかし, それらを極端な値に設定すると, 一方のレーンに通信が偏る可能性がある. また, しきい値を満たすショートフローの割合を大きくするには, $t_{deadline}$ を本来のしきい値より小さくし, $gamma$ を大きく, α を小さくすれば良い. これにより, SL への負荷を減らすことができ, SL を良好な状態に保つことができ, アプリケーション要求を満たすフローの割合を大きくすることができると考えられる. ネットワーク環境に最適なパラメータについては, そのネットワークが持つレーン数や, アプリケーションによって決まると考えられる.

7.2　実験環境についての考察

本節では, データセンタレーンモデルを利用した実験環境に関して, レーン構成の違いとそれを実現するためのトポロジーについて考察する.

7.2.1　データセンタレーンモデルに対する考察

レーンの割り当て方には様々な組み合わせが考えられる. 今回の評価実験では, 基本的にエンドノード間の物理的に異なる経路が二つあることを想定し, それぞれに対し SL, LL を割り当てた. 今後, MPTCP が普及していくと, 今以上にエンドノードにおいて複数のインタフェースを持つことが考えられる. 実際, エンドノードが複数 NIC を持ち, マルチホーミングな通信を実現することは, エンドノードのネットワーク処理改善の解決方法の一つと考えられる. その時, 異なる物理パスがインタフェースの数だけ用意することが可能となる. §6.2 の基本性能に関する評価実験では, 3 つの経路がある環境において, 1 つを SL に割り当て, 残りを LL に割り当て, 経路切替を行った. この構成では, 経路が切り替わる前は最大一経路分, 切り替わった後には最大二経路分のスループットを達成できる.

3 つの経路がある環境では, 2 つを SL に割り当て, 残りを LL に割り当てる構成が考えられ

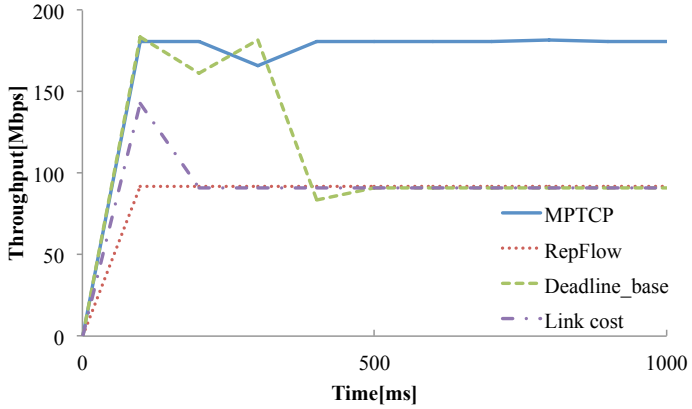

Fig.7.3　Basic performance comparisons with 2 SLs and a LL

る．この時の時系列ごとのスループットの変化を Fig.7.3 示す．評価環境は, §6.2 と同様である．SL が 2 経路あるため, 通信開始当初は, 2 経路分の通信性能を出すためにウィンドウサイズを増加させていく．リンクコストベースの切替手法では, 増加の過程で切替が生じ, LL1 経路分の通信性能へ収束する．デッドラインベースの切替手法では, デッドライン時間を迎えるまでに最大帯域利用可能となり, 経路切替まで 2 経路分の通信性能を実現できている．これは SL に 1 経路しか割り当てられていない場合に発生していた通信開始直後の遅延問題を解消できる可能性がある．実際, MPTCP で用いられている輻輳制御には, より有利な経路に対しウィンドウサイズを拡大していくというアルゴリズムがあるので, 混雑した経路の影響を抑えることができると考える．しかし, 切替が生じた場合, 1 経路分の性能しか出せないので MPTCP による効果が現れない.

また, 4 つの経路がある環境において, 2 経路ずつそれぞれ SL, LL 二割り当て流構成も考えられる．この時の時系列ごとのスループットの変化を Fig.7.4 示す．評価環境は, §6.2 と同様である．SL が 2 経路あるため, 通信開始当初は, 2 経路分の通信性能を出すためにウィンドウサイズを増加させていく．そして経路切替により, LL2 経路分の通信性能を実現できている.

このように, 異なる経路 (エンドノードにおけるそれぞれのインターフェース) によって, SL と LL の組み合わせを変更することができる．しかし, LL の性質から考えると, 長い時間, 大きなサイズの通信を行うフローが最終的に切り替わって来るため, より大きなスループットを出すためにも, LL になるべく多く経路を割り当てるように設定する必要がある.

7.2.2　MPTCP を有効活用するトポロジーに関する考察

ネットワーク帯域の効率的な利用を目指す取り組みとして, 様々なデータセンターネットワークトポロジーが提案されている．提案されているトポロジーは, 複数の等コストな経路がある冗長性を持たせた構成になっており, そうした経路を有効活用する手段として MPTCP が

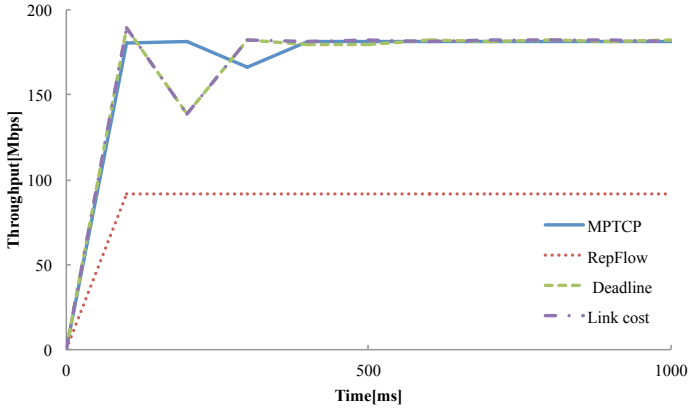

Fig.7.4　Basic performance comparisons with 4paths

ある．§3.3 に示したように, MPTCP を用いることで, 将来の広帯域化により発生すると予想されているノード側のボトルネックについても, 解決することができると期待されており, デュアルホーミングを活用したトポロジーも提案されている [9]．具体的には, 今日のサーバでは一般的である複数の NIC に対して MPTCP を利用するということである．そこでエンドノードの持つ複数の NIC を有効活用するためのトポロジーの一つの提案として, マルチホーミングな FatTree(Multi-Homing FatTree) が考えられる．

Fig.7.5 に k-MHFT を示す．k-ary マルチホーミングトポロジーは FatTree と同様に, k 個のポッドから構成されおり, 一つのポッドで $k/2, k^2/4$ それぞれ aggregator スイッチと edge スイッチを持つ．aggregator スイッチと edge スイッチではそれぞれ, $k/2$ のノードと上位レイヤーのスイッチ 1 つに接続し, 計 $k/2+1$ ポートが必要である．また, core スイッチは $k/2$ 台の $k/2$ ポートスイッチが必要である．さらに, MHFT は $k^3/4$ までのノードを持つことができ, $k/2$ のインタフェースが必要であるとする．この時, $k/2$ 本の経路が, 物理的に等価なコストの経路数となる．このトポロジーによって, 多様なデータセンター内でのトラフィックに対し, 複数のレーンから構成されるデータセンターレーンモデルにおいて, 通信負荷分散することが可能となる．

しかし, 今日の巨大なクラスターを持つデータセンターの設計には, 性能の優位さだけではなく構築に掛かるコストについても考慮しなければならない．そこで, MHFT 構築に掛かるコストについて, 保有できるエンドノードの台数とともに検討を行う．検討には, エッジ部分でのスイッチとして 48 ポート 1 ギガビットスイッチ ($7000) とアグリゲーション, コア部分でのスイッチとして 128 ポート 10 ギガビットスイッチを用いて構成する際のコストを検討する [7]．なお, 配線ケーブルのコストは考慮しないこととする．

Fig.7.6 に, 保有できるホストの数とトポロジー形成に掛かる費用の関係を示す．例えば, 20000 ホストに対するスイッチ機器を考えたとき, Fattree トポロジーでは, 1152 台の edge スイッチ, 1152 台の Aggregation スイッチ, 576 台の core スイッチによって構成することが

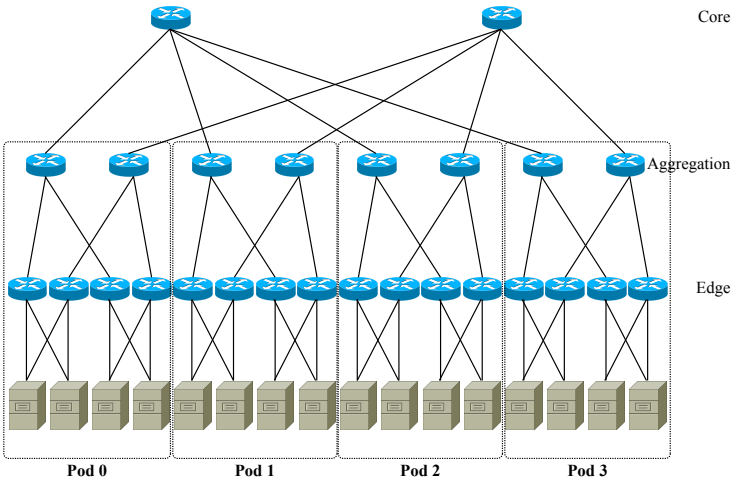

Fig.7.5　k=4 Multi-homing in the FatTree Topology

Table.7.1　Multi-homing FatTree constitution

k-ary	Number	Port/Interface
core	$k/2$	$k/2$
aggregator	$k^2/2$	$k/2+1$
edge	$k^3/4$	$k/2+1$
node	$k^3/4$	$k/2$

でき, かかる費用は約$1217M である．一方 MHFT において 20000 ホストを保有するトポロジーを構築しようとすると, 27648 台の edge スイッチ, 1152 台の Aggregation スイッチ, 24 台の core スイッチによって構成することができ, かかる費用は約$1016M である．

このように, MHFT ではコスト面では有効であると言える．しかし, FatTree に比べ, 等コストな経路の数は少なくなっており, 通信負荷分散の可能性としては FatTree の方が潜在している．

7.3　実環境への適用に関する考察

本節では, 実環境への提案手法の実装についての課題を考察する．

7.3.1　MPTCP アドレスペアの問題に関する考察

現状の MPTCP の実装上, サブフローを形成する際での信負荷の分散が有効に働かない問題がある．今, Fig.7.7 のような複数インタフェースを持つエンドノード間の通信を考える．この時, 現状の MPTCP 実装では, 互いのアドレスを交換 (ADD_ADDRESS) しあった後, フル

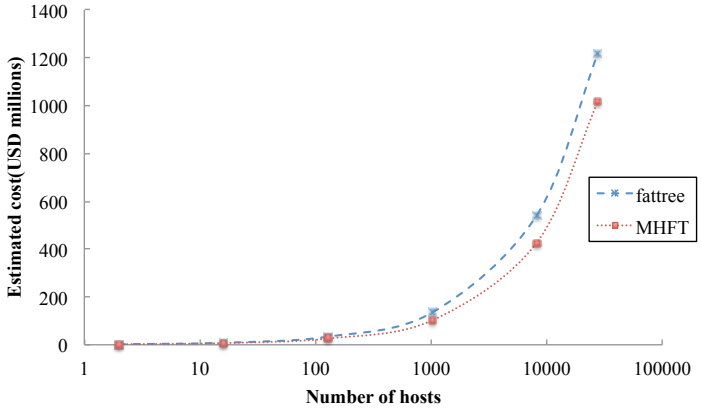

Fig.7.6　Current cost estimation vs. maximum possible number of hosts

メッシュな組み合わせの IP アドレスでサブフローが形成される．具体的には, 1 つの TCP コネクション (F1src:10.1.0.1, dst:10.2.0.1) と 3 つのサブフロー (F2src:10.1.0.1, dst:10.2.1.1, F3src:10.1.1.1, dst:10.2.0.1, F4src:10.1.1.1, dst:10.2.1.1) が形成される．静的な IP ベースのルーティングを想定した環境において, フロー F1 はサーバ側へのデータパケット通信に対して Data:S1-S2-S4, クライアント側への ACK パケット通信に対しては ACK:S1-S2-S4 の経路をそれぞれ用いる．同様に, フロー F2 は Data:S1-S2-S4, ACK:S1-S3-S4, フロー F3 は, Data:S1-S3-S4, ACK:S1-S2-S4, フロー F4 は, Data:S1-S3-S4, ACK:S1-S3-S4 の経路をそれぞれ通る．基本的にデータパケット通信の方がデータサイズもパケットの数も多いため, キューイング遅延がより起こりやすい通信であり, 例えばフロー F1 と F2 では異なるサブフローにも関わらず, Data パケットは同一の経路を通る．また, MPTCP はそれぞれのフローを同一のものとして輻輳制御を行うため, 経路によってはトラフィックが偏る可能性があり, 有効的な経路利用を実現できない.

今回のような複数の経路を持つネットワーク環境において, 通信不可の分散の点から理想的なサブフローの形成は, 1 つの TCP コネクション (src:10.1.0.1, dst:10.2.0.1) と 1 つのサブフロー (src:10.1.1.1, dst:10.2.1.1) が形成されることである．すなわち, 1 インタフェース 1 サブフローの原則による, 重複のない IP アドレスのペアを形成することで, 最適なトラフィック負荷の分散を実現できる．そのため, 本提案手法の実装にあたり, 既存の MPTCP 実装に対して変更が必要であった.

7.4　今後の課題について

本節では, データセンターが抱える性能障害の問題に対し, 提案手法が今後どのような解決策を提示できるのかについての考察と, 実環境における適用の際の課題について考察する.

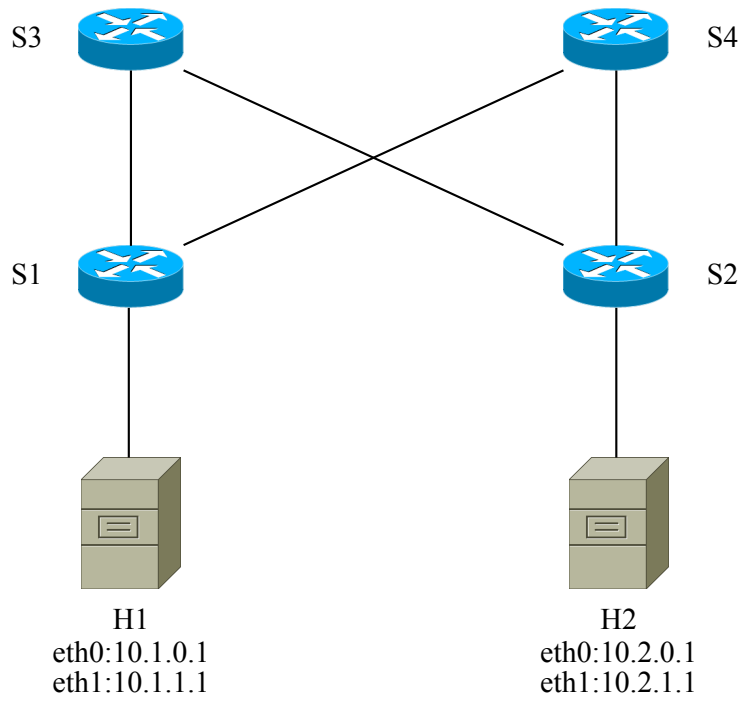

Fig.7.7　Techinical problem of creating subflow without complete paired IP addressses

7.4.1　Incast 問題に対する考察

データセンターが抱える性能障害の一つに Incast 問題がある [36]．これは，一つのインタフェースに大量のショートフローが同時に集中することにより遅延が生じるという，データセンターでの並列分散処理が生み出す特有の問題である．今回の提案手法では，Queue buildup に対するアプローチであるために，この Incast 問題に対して有効に働く手立てはない．しかし，データセンターレーンモデルによる通信の切り分けによって，この問題を解決できる可能性がある．今の MPTCP 実装では，サイズの小さい通信には複数経路を用いずに一つの経路だけで通信が完了するため，今回の提案手法のような経路切替手法は有効に働かない．そのため，今回のような複数経路への負荷分散を考えた場合，あらかじめ通信する経路を分散させる必要がある．このような静的な負荷分散として，ラウンドロビンやハッシュベースを用いた負荷分散手法がある．しかし，これらの手法ではその不完全な負荷分散によって，Fig.2.4 のようにロングフローとショートフローが同じ経路を使って通信する場合がある．今回提案したデータセンターレーンモデルはではそれらのフロー区別して制御することができ，SL を利用するショートフローに対してのみ，負荷分散の効果を与えることができる．例えば，エンドノード間の通信経路が 3 本存在している時には，SL に 2 経路，LL に 1 経路割り当てた時，SL に対してラウンドロビンやハッシュベースを用いた負荷分散手法が有効であると考える．そこで，3 経路のうち，SL に 2 経路，LL に 1 経路割り当てた時の Incast トラフィックに対する負荷分散の予備実験

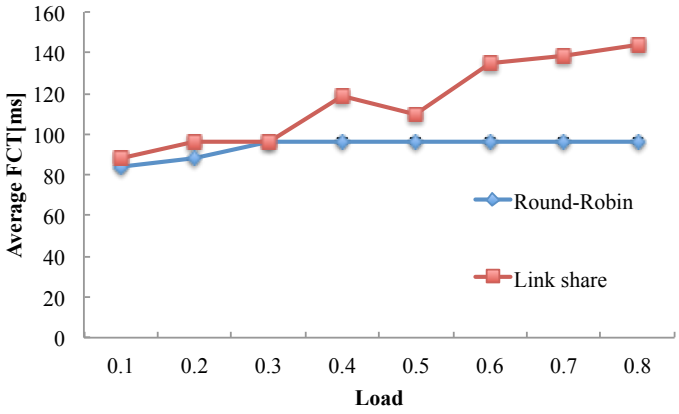

Fig.7.8　Effect of the load balancing by simple idea with Round-Robin alghorithm

を行った. 検証環境は, Fig.5.3 のような SL がに 2 経路割り当てられているトポロジーを考える. このトポロジーでは, 左右に 20 ノードずつ計 40 ノードが接続されている. ベンチマークトラフィックとして, 70KB フローに対し平均 200ms のポアソン生起を行う. 通信を行うノードの数の割合を大きくしていき, incast トラフィックによる通信負荷を大きくする. この時, ラウンドロビンによる負荷分散と一つの経路を占有する状況を比較する. Fig.7.8 に結果を示す. 縦軸が incast トラフィックに対する平均 FCT, 横軸は通信ノードの割合を示している. この結果はラウンドロビンが理想的に働いた時の結果を意味しているが, 提案したデータセンターレーンモデルでは, こうした同地に扱って良いフローに対してのみ, 負荷分散を適用することができるため, その効果は大きいと考えられる.

7.4.2　実環境における適用, 運用に関する考察

本提案手法では, 各ノードが複数のインターフェースがあり, 等価なコストの経路がインタフェース数だけ用意されているマルチパスなネットワーク環境での利用を想定しており, エンドノード OS の変更のみで実環境への適用が可能となる. この時, 各インタフェースに割り当てられている IP アドレスが独立して通信を行えるようなルーティング設定が必要となる. 今回の評価実験では, ルーティングについては各スイッチでのスタティックルーティングにより環境を構築した.

今日の大規模データセンターにおいては, 機器の故障やメンテナンス等で, 一部のリンクがダウンすることは比較的頻繁に起こり [65], 常に完全対称なトポロジーの運用は困難である. 特に本提案手法では, 宛先 IP アドレスによる経路負荷分散がなされている事が前提であるために, 局所的に通信ができない部分があると, 提案手法による効果が小さくなる. 今回の実装では, そういった一部の経路が利用できなくなった際のフォールバック MPTCP による複数経路利用に一任するという点で課題がある. 実際, ネットワーク障害の度に, 適切なルーティングを

考慮し, マニュアルでルーティングとエンドノード OS へ設定の修正を加えるのは現実的ではない. こういった運用の課題については近年, OpenFlow という技術によって, ネットワークの柔軟な制御を実現することが可能となってきており, 今後の活用が期待されている [66]. 現時点ではデータセンター内のネットワーク機器で OpenFlow に対応している機器は少ないが, 今後, こういった小規模な障害, トラブルの対応に対し, 運用面での活用できる可能性をを秘めている.

第 8 章

結論

本研究では, エンドノード間通信において複数のインタフェースを用いて複数の経路がある今日のデータセンターネットワークでのネットワーク内通信における, ショートフローの遅延問題について, MPTCP を用いた経路切替手法によってフロー完結時間の短縮化を行う手法を提案した. 遅延の主な原因の一つであるキューイング遅延が発生しているスイッチに対して直接変更を加える従来手法では, 実環境への適用に大きな課題があった. そこで, 提案手法では, エンドノードの OS に対してのみ変更を加えることで, 通信性能の改善を目指した. 併せて, 実際のデータセンターが生成するトラフィックの一例として, データセンターで広く利用されている並列分散処理アプリケーションのトラフィックを観測することで, サイズの大きいスループット指向なロングフローとサイズの小さいレイテンシ指向なショートフローが一つのリンクに混在して通信を行っていることを明らかにし, それによる通信性能障害が生じる技術的背景を考察した. そして, 指向毎に異なる通信経路を利用するレーンモデルを提案した. また, OS レベルから一つの通信を見た際に, そのフローの指向をフロー持続時間を用いて分類し, 経路の状況に合わせた経路切替手法を提案した.

提案手法を OS に実装し, シミュレーションを用いた評価を行い, ロングフローとショートフローが混在する環境下で, それぞれの指向毎に経路を切り替える制御を行うことで, ショートフロー通信を短縮化することを示した.

今後の課題としては, 提案手法の適用範囲の拡大とエンドノードに対する通信性能の向上が考えられる. 適用範囲の拡大としては, Incast 問題への対応が考えられる. 並列分散処理を運用するデータセンターにおいて, Queue buildup と並ぶ大きな課題である Incast 問題に対し, 本提案手法では改善することはできない. これは指針として, エンドノードのみに対するアプローチを行う以上, 予め経路の状況を把握しておき, それに応じて最適な経路制御を行うことが困難であるためである. また, 現状 MPTCP 実装がサイズの小さい通信には複数経路を利用せず, 初めにコネクションを確立した経路のみで通信を完了するためである. この課題の解決として, エンドノードへのアプローチとして OS 側で通信履歴を保持しておき, 経路を切り替える手法, スイッチに対するアプローチとして SDN を用いて全ルーティングを管理する手法が考えられる. 前者については, 通信の発生頻度に依存する経路決定の正確さ, 後者では, スケーラ

ビリティの問題があり, それらを解決するためのフロー制御を考えていきたい．また近い将来の通信帯域の広帯域化によるエンドノードのボトルネック問題があり, NIC 負荷分散手法として, MPTCP による経路切替による, インタフェース切替手法が期待されている．このように, 今後ネットワークのみならずエンドノードの性能まで拡大した改善手法が求められていると考えられ, 上記の課題を解決することでデータセンター性能の改善が実現できると考えられる．

発表文献

国内会議 (査読あり)

1. 藤居 翔吾, 田崎 創, 関谷 勇司, ”MultiPath TCP 適用時のデータセンターネットワークでのフローサイズが与える影響に関する一考察”, インターネットカンファレンス 2014

国内会議 (査読なし)

1. 藤居 翔吾, 田崎 創, 関谷 勇司, ”データセンター環境におけるショートフロー通信改善手法の一提案”, 電子情報通信学会, 信学技法, vol. 113, no. 364, IA2013-65, pp. 47-52, 2013.

受賞

1. 学生研究奨励賞:藤居 翔吾, 田崎 創, 関谷 勇司, ”データセンター環境におけるショートフロー通信改善手法の一提案”, 電子情報通信学会, 信学技法, vol. 113, no. 364, IA2013-65, pp. 47-52, 2013.
2. 藤居翔吾, 田崎創, 関谷勇司, ”MultiPath TCP 適用時のデータセンターネットワークでのフローサイズが与える影響に関する一考察”, 日本ソフトウェア科学会研究会資料シリーズ No.74 ISSN1341-870X, インターネットコンファレス 2014 論文集, pp33-42

謝辞

本研究および大学院修士課程 2 年間の生活においてご指導ご鞭撻していただいた東京大学の関谷勇司准教授と田崎創特任講師に深く感謝いたします．また, 本研究を進めるにあたり多くの建設的な意見を与えてくださった東京大学の若原恭教授，中山雅哉准教授，小川剛史准教授，妙中雄三助教，宮本大輔助教に心より感謝いたします． 研究室での生活においては，Jia Xiazhou 氏をはじめ，多くの心優しい同期，先輩，後輩に恵まれ，大変充実した 2 年間を送ることができました．ここに感謝の意を表します．最後に，長い学生生活を支えてくださった両親，兄弟に深く深く心より感謝を致します．

参考文献

[1] 日本アイ・ビー・エム株式会社. IBM 第 1 章大容量データのバックアップ, http://www-06.ibm.com/systems/jp/storage/column/backup/01.html

[2] Jim Liddle. Amazon found every 100ms of latency cost them 1% in sales, August 2008. http://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/

[3] R. Kohavi et al. Practical Guide to Controlled Experiments on theWeb: Listen to Your Customers not to the HiPPO. KDD, 2007.

[4] J. Hamilton. On designing and deploying Internet-scale services. In USENIX LISA, 2007.

[5] Facebook. Presto: Interacting with petabytes of data at Facebook, https://www.facebook.com/notes/facebook-engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920

[6] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[7] Al-Fares, Mohammad, Alexander L oukissas, and Amin Vahdat. "A scalable, commodity data center network architecture." ACM SIGCOMM Computer Communication Review. Vol. 3

[8] Alizadeh, Mohammad, et al. "Data center tcp (dctcp)." ACM SIGCOMM Computer Communication Review 40.4 (2010): 63-74.

[9] Raiciu, Costin, et al. "Improving datacenter performance and robustness with multipath TCP." ACM SIGCOMM Computer Communication Review. Vol. 41. No. 4. ACM, 2011.

[10] Zats, David, et al. "DeTail: Reducing the flow completion time tail in datacenter networks." ACM SIGCOMM Computer Communication Review 42.4 (2012): 139-150.

[11] Alizadeh, Mohammad, et al. "pfabric: Minimal near-optimal datacenter transport." Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM. ACM, 2013.

[12] Kohler, Eddie, et al. "The Click modular router." ACM Transactions on Computer

Systems (TOCS) 18.3 (2000): 263-297.

[13] Ford, Alan, et al. TCP Extensions for Multipath Operation with Multiple Addresses: draft-ietf-mptcp-multiaddressed-03. No. Internet draft (draft-ietf-mptcp-multiaddressed-07). Roke Manor, 2011.

[14] Benson, Theophilus, Aditya Akella, and David A. Maltz. "Network traffic characteristics of data centers in the wild." Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010.

[15] J. Salim, When NAPI Comes to Town, Proceedings of Linux 2005 Conference, UK, August 2005.

[16] Microsoft corporation. scalable networking with rss, 2005.

[17] Herbert, T. rfs: receive flow steering, september 2010. http://lwn.net/Articles/381955/.

[18] Herbert, T. rps: receive packet steering, september 2010. `http://lwn.net/Articles/361440/.`

[19] ip networking lab「MultiPath TCP - Linux Kernel implementation」`http://mptcp.info.ucl.ac.be/`

[20] Vasudevan, Vijay, et al. "Safe and effective fine-grained TCP retransmissions for datacenter communication." ACM SIGCOMM Computer Communication Review. Vol. 39. No. 4. ACM, 2009.

[21] 藤居 翔吾, 田崎 創, 関谷 勇司, "MultiPath TCP 適用時のデータセンターネットワークでのフローサイズが与える影響に関する一考察", 電子情報通信学会, 信学技法, vol. 113, no. 364, IA2013-65, pp. 47-52, 2013.

[22] 藤居翔吾, 田崎創, 関谷勇司, "MultiPath TCP 適用時のデータセンターネットワークでのフローサイズが与える影響に関する一考察", 日本ソフトウェア科学会研究会資料シリーズ No.74 ISSN1341-870X, インターネットコンファレス 2014 論文集, pp33-42

[23] P. Agarwal, B. Kwan, and L. Ashvin. Flexible buffer allocation entities for traffic aggregate containment. US Patent 20090207848, August 2009.

[24] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. IEEE/ACM ToN, 1994.

[25] A. Walid, et al. Balanced Linked Adaptation Congestion Control Algorithm for MPTCP draft-walid-mptcp-congestion-control-00, 2014.

[26] tcpdump, `http://www.tcpdump.org/`

[27] intel, `http://www.intel.com/content/www/us/en/network-adapters/gigabit-network-adapters/ethernet-server-adapters.html`

[28] Lu, Guohan, et al. "Using cpu as a traffic co-processing unit in commodity switches." Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012.

[29] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335.

[30] Iyengar, Janardhan R., Paul D. Amer, and Randall Stewart. "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths." Networking, IEEE/ACM Transactions on 14.5 (2006): 951-964.

[31] Iyengar, Janardhan, et al. "Concurrent multipath transfer using SCTP multihoming." SPECTS 2004 (2004).

[32] IEEE Standards Association. "IEEE 802.3 LAN/MAN CSMA/CD Access Method." (2008).

[33] Fukunaga, Takafumi, and Hidenori Umeno. "Implementation and evaluation of improvement in parallel processing performance on the cluster using small-scale SMP PCs." IEEJ Transactions on Electronics, Information and Systems 128 (2008): 1842-1851.

[34] Umeshima S, Higaki H. Extended ARP for high-per- formance LAN communications with multiple kinds of NICs. 2003-DSM-029, Vol. 2003, No. 38, p 31 – 36, 2003.

[35] Meisner, David, et al. "Power management of online data-intensive services." Computer Architecture (ISCA), 2011 38th Annual International Symposium on. IEEE, 2011.

[36] Chen, Yanpei, et al. "Understanding TCP incast throughput collapse in datacenter networks." Proceedings of the 1st ACM workshop on Research on enterprise networking. ACM, 2009.

[37] Floyd, Sally, and Van Jacobson. "The synchronization of periodic routing messages." Networking, IEEE/ACM Transactions on 2.2 (1994): 122-136.

[38] Iyer, Sundar, Ramana Rao Kompella, and N. McKeowa. "Analysis of a memory architecture for fast packet buffers." High Performance Switching and Routing, 2001 IEEE Workshop on. IEEE, 2001.

[39] Barroso, Luiz Andr é, Jeffrey Dean, and Urs Holzle. "Web search for a planet: The Google cluster architecture." Micro, Ieee 23.2 (2003): 22-28.

[40] Wilson, Christo, et al. "Better never than late: Meeting deadlines in datacenter networks." ACM SIGCOMM Computer Communication Review. Vol. 41. No. 4. ACM, 2011.

[41] Vamanan, Balajee, Jahangir Hasan, and T. N. Vijaykumar. "Deadline-aware datacenter tcp (d2tcp)." ACM SIGCOMM Computer Communication Review 42.4 (2012): 115-126.

[42] Beaver, Doug, et al. "Finding a Needle in Haystack: Facebook's Photo Storage." OSDI. Vol. 10. 2010.

[43] Hastorun, Deniz, et al. "Dynamo: Amazon ' s highly available key-value store." In

Proc. SOSP. 2007.

[44] Isard, Michael, et al. "Dryad: distributed data-parallel programs from sequential building blocks." ACM SIGOPS Operating Systems Review. Vol. 41. No. 3. ACM, 2007.

[45] Liu, Shuhao, et al. "RepFlow on node. js: Cutting Tail Latency in Data Center Networks at the Applications Layer." arXiv preprint arXiv:1407.1239 (2014).

[46] Greenberg, Albert, et al. "VL2: a scalable and flexible data center network." ACM SIGCOMM Computer Communication Review. Vol. 39. No. 4. ACM, 2009.

[47] Curtis, Andrew R., et al. "Devoflow: scaling flow management for high-performance networks." ACM SIGCOMM Computer Communication Review. Vol. 41. No. 4. ACM, 2011.

[48] Clos, Charles. "A Study of Non - Blocking Switching Networks." Bell System Technical Journal 32.2 (1953): 406-424.

[49] U.S. Department of Commerce, Bureau of Public Roads(1964), Traffic Assignment Manual

[50] Davidson, K. B. "A flow travel time relationship for use in transportation planning." Australian Road Research Board (ARRB) Conference, 3rd, 1966, Sydney. Vol. 3. No. 1. 1966.

[51] Tazaki, Hajime, et al. "Direct code execution: Revisiting library os architecture for reproducible network experiments." Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. ACM, 2013.

[52] McKeown, Nick. "A fast switched backplane for a gigabit switched router." Business Communications Review 27.12 (1997): 1-30.

[53] Priority flow control: Build reliable layer 2 infrastructure. `http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/white_paper_c11-542809.pdf`

[54] Hong, Chi-Yao, Matthew Caesar, and P. Godfrey. "Finishing flows quickly with preemptive scheduling." ACM SIGCOMM Computer Communication Review 42.4 (2012): 127-138.

[55] Liu, Shuhao, Hong Xu, and Zhiping Cai. "Low latency datacenter networking: A short survey." arXiv preprint arXiv:1312.3455 (2013).

[56] Alizadeh, Mohammad, et al. "Less is more: trading a little bandwidth for ultra-low latency in the data center." Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[57] Zarifis, Kyriakos, et al. "DIBS: just-in-time congestion mitigation for data centers." Proceedings of the Ninth European Conference on Computer Systems. ACM, 2014.

[58] Zats, David, et al. FastLane: An agile congestion signaling mechanism for improving

datacenter performance. No. UCB/EECS-2013-113. CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE, 2013.

[59] Cheng, Peng, et al. "Catch the whole lot in an action: Rapid precise packet loss notification in data centers." Proc. USENIX NSDI. 2014.

[60] Chen, Li, et al. "Towards minimal-delay deadline-driven data center tcp." Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks. ACM, 2013.

[61] Guo, Chuanxiong, et al. "BCube: a high performance, server-centric network architecture for modular data centers." ACM SIGCOMM Computer Communication Review 39.4 (2009): 63-74.

[62] Raiciu, C., M. Handley, and D. Wischik. "Coupled congestion control for multipath transport protocols." draft-ietf-mptcp-congestion-01 (work in progress) (2011).

[63] Jacobsson, Krister, et al. "Round trip time estimation in communication networks using adaptive kalman filtering." (2004).

[64] Angrisani, Leopoldo, et al. "Measurement of processing and queuing delays introduced by an open-source router in a single-hop network." Instrumentation and Measurement, IEEE Transactions on 55.4 (2006): 1065-1076.

[65] DEAN, J. Software engineering advice from building large-scale distributed systems. http://research.google.com/people/jeff/stanford-295-talk.pdf.

[66] McKeown, Nick, et al. "OpenFlow: enabling innovation in campus networks." ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74.

付録 A　リンクパフォーマンス関数:Davidson 関数

リンクパフォーマンス関数は, ネットワークを構成する個々のリンクのサービス水準をリンク交通量とリンク属性の関数として表したものである．本研究では, リンクのサービス水準には, フロー完結時間 (旅行時間) を用いている．リンクパフォーマンス関数は, 狭義の増加関数である必要があり, このことは待ち行列理論を用いて説明することができる．

今, M/M/1(∞) の待ち行列理論を考える．この時, 到着率 λ, サービス率 μ を用いて利用率 ρ は以下のように表される．

$$\rho = \frac{\lambda}{\mu} \tag{1}$$

この利用率 ρ と平均サービス時間 T_s を用いて, 待ち時間 T_w は以下のように表される．

$$T_w = \frac{\rho}{1-\rho} \times T_s \tag{2}$$

次にあるリンクにおけるフロー完結時間 t を導出する．このリンクにおいて, 他のフローが存在しない時のフロー完結時間を t_0 とする．この時, 遅れのパラメータ $J(>0)$ と式 1, 2 を用いて以下のように表すことができる．

$$Jt_0\frac{\rho}{1-\rho} = Jt_0\frac{q}{C-q} \tag{3}$$

ここで, q はリンク利用率, C はリンク容量である．これにより, フロー完結時間 $t(q)$ は次のように定義される．

$$t(q) = t_0 + Jt_0\frac{\rho}{1-\rho} = t_0\Big\{1 + J\big(\frac{q}{C-q}\big)\Big\} \tag{4}$$