Application Layer Multicastを用いたPub/Sub基盤と連携動作するOpenFlow Multicastの設計および実装とメンバ管理オーバヘッドの評価

Design and Implementation of OpenFlow multicast cooperating with Application Layer Multicast and its evaluation of membership management performance

藤田 雅浩 † 秋山 豊和 †
Masahiro Fujita Toyokazu Akiyama

京都産業大学 †
Kyoto Sangyo University

概 要

IoTやSNSなどの近年のインターネットアプリケーションではPub/Sub通信モデルが広く利用されている．Pub/Sub通信ではPublisherが送信した情報をSubscriberに届けるBrokerが存在する．Brokerはスケーラビリティや負荷分散の観点から複数台利用される．その場合，Broker間の通信方式が通信遅延等に影響する．必要なBrokerのみにメッセージを配信する方式としてApplication Layer Multicast(ALM)の利用が考えられるが，ALMでは実際のネットワーク構成を把握できないため，ネットワーク構成を無視した転送経路が生成され，その結果同一機器上を複数回往復するといった無駄が発生する可能性がある．この課題を解決するために本研究ではOpenFlow技術を用いたマルチキャスト(OFM)を利用したアプローチを取る．ただし，OpenFlowにもスケーラビリティ等の問題があるため，本研究ではALMとOFMの両方を連携させる．既存研究において，OFMによるトラフィックおよび転送遅延削減の効果はシミュレーションにより確認できているが，OFM切り替え後のメンバ管理におけるオーバヘッドが明らかになっていなかった．本稿では連携にあたりアプリケーションからOFMを制御可能にするためのOpenFlow Controllerの設計および実装を行い，OFM利用時のメンバ管理オーバヘッドについて性能評価を行った．評価の結果，OFMはALMに比べてSubscriberの参加・脱退処理に時間がかかり，ネットワークに反映されるまでに時間がかかるが，参加・脱退処理の時間は実装を改良することで短縮が見込まれる．また，負荷集中点があることもわかり，OFM利用時の参加・脱退頻度を考慮する必要があると考えられる．

1 はじめに

近年メッセージ配信ツールとして普及しているSNSやIoTにおけるセンサアクチュエータネットワークの通信において，様々な情報の送受信に対応するためにトピックベースのPub/Sub通信モデルが広く用いられている．Pub/Sub通信モデルとは，送信者であるPublisherは受信者を想定せずに情報を送信でき，受信者であるSubscriberは，必要な情報を表すトピックを選択購読することで情報を取得可能な非同期メッセージングモデルである．Pub/Sub通信ではPublisherとSubscriberの仲介を行うBrokerが存在し，実際のメッセージ受け渡しを行っている．Brokerはスケーラビリティの問題や負荷分散の観点から複数分散設置し，相互接続される．Broker間の通信方式がトラフィック量や通信遅延に大きく影響する．Pub/Subミドルウェアの実装では，1対1のIP通信（ユニキャスト）によるブロードキャストが利用されているが[1][2]，Application Layer Multicast(ALM)を用いることで，必要なBrokerのみにメッセージを転送する方法が考えられる[3]．ALMでは，Pub/Subミドルウェアが決定・管理を行う経路に基づいて，必要なBrokerにのみ情報を転送する．しかし，Pub/Subミドルウェアは基本的にネットワーク機器間のトポロジを把握することができない．そのため，接続状況を考慮していない経路が決定され．Brokerやスイッチなど同一インスタンス上をパケットが複数回経由する場合がある．この課題の解決方法として，下位層の情報を用いてALMの転送経路を最適化する方法[4]と下位層のマルチキャスト機能を用いて最適経路で配信する方法[5]が考えられる．これらの両方式は後述の文献[5]で提案されている．プログラマブルなネットワークの実現を目指すOpenFlowを活用することで，ポリシーなどを考慮した制御が導入しやすい利点がある．また，アプリケーション層における最適化と下位層の制御による最適化は併用することが可能であり，文献[5]ではその併用とシミュレーションによる評価について述べている．本稿では，文献[5]の方式において，アプリケーションから制御可能なOpenFlowを用いたMulticast(OFM)の実装を行う．特にOFM網の構築機能ならびにマルチキャストにおけるメンバの参加・脱退機能を提供するためのOpenFlow Controller(OFC)およびそのNorthbound APIの設計ならびに実装を行い，構築したプロトタイプシステムの評価を行う．

以下，2節でBroker間通信の課題，3節および4節でALMとOFMの連携機構の設計および動作について述べる．4節は連携機構の実装について述べ，5節で評価実験について述べる．最後に6節でまとめと今後の課題について述べる．

2 Pub/Sub通信におけるBroker間通信の課題

IoTアプリケーションではホームゲートウェイなど，多数のBrokerが相互接続されるため，Broker間通信の削減が課題となる．Broker間通信の削減方法としてALMの導入が考えられる．ALMでは物理ネットワークすな

わちアンダーレイネットワーク上に仮想ネットワークすなわちオーバレイネットワークによる配信用の論理トポロジを形成する．本研究では PIAX[6] の利用を想定しており，PIAX が提供している P2P 構造化オーバレイネットワークにより ALM を実現する．PIAX では Multi Key Skip Graph を用いて Broker 間のオーバレイを構築している．ALM においてオーバレイを構築する際，アンダーレイが隠蔽されている場合，アンダーレイの構造を知ることができないため，Broker 間の接続時にアンダーレイの構造を参考にして構築することができない．その結果，オーバレイとアンダーレイのトポロジにずれが生じる．このずれによる性能低下を回避する方法として，2.1 節でアンダーレイの情報を活用して ALM を最適化する方法について，2.2 節でアンダーレイを最適化する方法について述べる．

2.1　アンダーレイの情報を活用した ALM の最適化

アンダーレイと異なるトポロジをもつオーバレイを用いて ALM を動作させると，配信元 Broker から遠くにある Broker に情報配信した後，配信元 Broker の近くにある Broker に情報を配信するような遠回りな配信が発生することがある．こうした非効率な配信を改善するために，ISP が持つアンダーレイのトポロジ情報をオーバレイ側に提供する ALTO が考案されている [7]．ALTO を使うと ISP が提供するアンダーレイの情報を利用してオーバレイを生成できるため，下位層の情報を利用したオーバレイのトラフィック量や遅延時間を低減するアプローチが研究されている [4]．しかし，ALM では情報の複製や転送が Broker 上で行われるため，少なくとも Broker がネットワークに接続するインタフェース上において，一旦 Broker が情報を受け取るための経路の重複が発生する．その結果，配信先が増えるごとに経由する Broker 数が増大し，トラフィックの重複が増加する．また，情報の伝達遅延時間についても経由する Broker 数に比例して大きくなる．これらはオーバレイの規模が大きくなるほど深刻化すると考えられる．

2.2　OpenFlow による Multicast を用いたアンダーレイの最適化

ALM と異なる配信方式として OpenFlow による Multicast を用いる方法がある．OpenFlow では OpenFlow Switch(OFS) を OFC で集中管理するため，ネットワークのトポロジ情報の取得が容易であり，Broker 間の最短経路を求めることも可能となる．また，OpenFlow では OFC が生成したフローエントリに従って OFS がパケットの転送をおこなうため，OFS に登録するフローエントリを適宜変更することでアンダーレイの制御が可能となる．フローエントリは，どのようなパケットに適用するかを示すマッチングルール，適用したパケットをどう処理するかを示すアクションから成っており，OFS にあるフローテーブルに登録される．マッチングルールには，MAC アドレスや IP アドレスといったパケットヘッダの情報を任意に指定でき，アクションには OFS における出力ポート，送信元・宛先 IP アドレスおよび MAC アドレスの書き換え，パケットの破棄などを指定でき，柔軟な配信が可能となる．しかし，OpenFlow にも制限がある．まず OpenFlow に対応したネットワーク機器が必要となる．また，管理ドメインドメインを超えた通信においてマルチキャストアドレスを使用するのは現実的ではないため，マルチキャストで自由にドメイン間通信を行うことは現状不可能である．さらに，製品として販売されているハードウェアの OFS のフローテーブルに登録可能なフローエントリ数には 10 万件程度の上限があり，1 トピックあたりのフローエントリ数を考えると大規模 Pub/Sub 環境の場合に全てを OpenFlow で対応するのは困難である．

3　ALM と OFM の連携機構

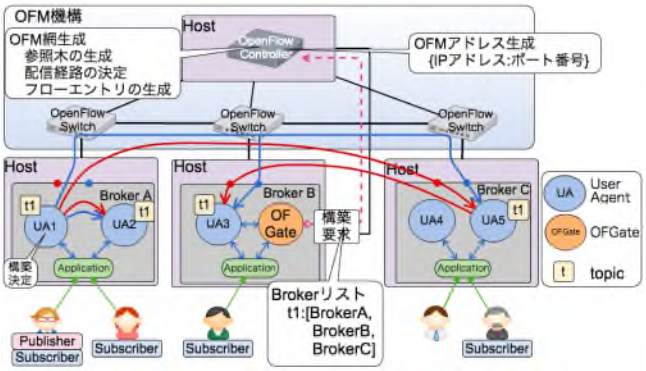

図 1　ALM と OFM の連携機構

本稿では前述で述べた 2 つの最適化方式の課題を解決するために，ALM と OFM を連携動作させる機構を構築する．以下に連携機構の詳細について述べる．ALM の基礎として ID Transport(IDT) と Locator Transport(LT) が分離可能な P2P エージェントプラットフォーム PIAX を利用する．PIAX はエージェント機構を有しており，ユーザと対応するエージェント間で通信することでユーザ間通信を実現している．つまり ALM による通信はエージェント間で通信が行われる．図 1 で UA と書かれた青丸がユーザエージェントを示す．また，PIAX と OFM を実現する OFC が連携する必要があり，そのための機構としてユーザと直接関係しない特殊なエージェント，OFGate を用意している．図 1 では OFGate と書かれた橙色の丸で示す．OFGate は図 1 中の赤点線で示すように，OFGate，OFC 間の制御用チャネルを経由して OFC の Northbound API にアクセスする．以下連携動作時の流れを示す．最初は図 1 の赤実線で示すように ALM を利用した Pub/Sub 通信をしており，Publisher と対応するエージェントが一定の条件を満たしたと判断すると OFGate 経由で OFC に対して OFM による通信網 (OFM 網) の構築要求を行い，OFC が OFM 網を構築することで図 1 の青実線で示したような OFM による Pub/Sub 通信が可能となる．つまり，基本は ALM による通信が行われ，必要に応じて OFM に切り替える．ただし，ALM か OFM かのどちらか一方しか利用できないわけではない．PIAX では ID Transport(IDT) と Locator Transport(LT) が分離されているため，エージェントは ALM と OFM の両方の LT で待ち受けることが可能と

なる．図 1 の各 Broker にある赤点が ALM の待ち受けポートを示し，青点が OFM の待ち受けポートを示している．異なるポートで待ち受けが可能なため，あるトピックは ALM で通信し別のトピックは OFM で通信を行うといったことが可能であり，さらに 1 つのトピック内においても ALM と OFM の通信を共用することも可能になると考えられる．このように ALM と OFM の併用を実現するため，OpenFlow の Hybrid モードを活用する．具体的には OFM 用のフローエントリとともに ALM 用のフローエントリを OFS のフローテーブルに設定する．ALM 用のフローエントリのアクションには，OpenFlow の仕様で仮想ポートとして定義されている“Normal”を指定する．Normal は OFS に既存 L2 スイッチと同じ動作をさせることができる．ただし，OFM で必要となる Broker のトポロジにおける位置を OFC が把握するために，Broker が最初に送信したパケットはパケットインし，その後 Normal フローエントリを生成する．これにより，ALM による通信および OFM による通信の両立が可能となっている．以上が連携機構の概要となる．

以下 3.1 節では OFM の構築，およびマルチキャストのメンバ参加，脱退の処理について述べる．3.2 節では連携機構での管理データについて述べる．

3.1 OFM 網の構築および参加・脱退

ALM から OFM への切り替えは Publisher と対応するエージェントによるモニタリング情報から判断する．OFM への切り替えを決定した場合，図 1 の中央下に示すように切り替えるトピックに関連する Broker のリストを OFC に通知する．メンバの把握は，メンバが ALM に Subscribe するときに OFGate にも通知するため OFGate によって行われる．この情報は Json 形式で OFC への OFM 網の構築要求に付加されており，OFM 網の構築要求は Northbound API を使って行われる．構築要求のための REST API は表 1 の 1 行目とした．OFC 側はこの URI で構築要求を受理する．OFC は構築要求を受理したら構築要求があったトピックと対応する OFM アドレスを割り当て，OFGate に返送する．この OFM アドレスは図 1 の右上に示しているような <IP アドレス:ポート番号 > の形式とし，OpenFlow の管理領域において利用可能なアドレス範囲を事前に指定しておく．これにより，固定的なマルチキャストアドレスを利用する場合に比べて，アドレス範囲が柔軟に設定できるようになる．OFM 網の構築完了後は，Publisher は OFM アドレス宛にパケットを送信することで，対応するトピックのメンバにメッセージを送信することが可能になる．構築要求を行った Publisher 以外が Publish する場合でも，Publish 時に OFGate に OFM 網の構築状況を問い合わせることで OFM を利用したメッセージ配信が可能となる．OFM を利用したメッセージ配信では，OFM 網が本当に構築されているか確認可能とするために，送信者にもパケットが送られるようになっている．もし，OFM アドレス宛にパケットを送信したにもかかわらず，送信者にパケットが送られてこなければ送信者は OFM 網が構築されていないと判断し，ALM によってパケットを再送する．

また，トピックにおけるメンバは動的に参加・脱退する．そのため，メンバの参加・脱退に応じて ALM 網や OFM 網の更新が必要となる．ALM 利用時には，メンバの参加および脱退は近隣 Broker にのみ通知され，必要に応じてオーバレイネットワークの再構築が行われる [6]．OFM 網構築後のメンバの参加および脱退は，構築同様 Northbound API を用いて OFGate 経由で OFC に通知される．参加要求のための REST API は表 1 の 2 行目とし，脱退要求は表 1 の 3 行目とした．このように要求ごとに REST API を用意している．本稿では REST API の要求は“POST”で行っているが，参加は“PUT”，脱退は“DELETE”に変更することも可能である．また，要求に対する応答として OFM アドレスを返送する．

表 1 Northbound API の定義

	REST API	クエリ	応答
構築	http://example.com/api/setupOFM	POST	OFM アドレス
参加	http://example.com/api/addOFMMembers/OFM アドレス	POST (PUT)	OFM アドレス
脱退	http://example.com/api/removeOFMMembers/OFM アドレス	POST (DELETE)	OFM アドレス

3.2 保持データ

OFC では OFM 網を構築するために情報を保持する必要があり，表 2 に管理形式を示し，図 2 に OFM 網の構築処理におけるデータの保持タイミングを示す．また，2.6.1 節から 2.6.5 節にて保持する情報について述べる．

表 2 データ管理形式

	Key	Value
トポロジ情報	datapath_id, ポート番号	接続先 datapath_id, 接続先ポート番号
参照木	datapath_id	node, edge
FDB	MAC アドレス	datapath_id, ポート番号
メンバ情報	OFM アドレス	MAC アドレス, IP アドレス, ポート番号
経由スイッチ情報	OFM アドレス	datapath_id
経由ポート情報	datapath_id	ポート番号

3.2.1 トポロジ情報

OFC による管理領域の把握や OFM のための配信木を算出するために，ネットワーク構成 (トポロジ) の情報を保持する必要がある．トポロジを把握するにはスイッチ間のリンクを全て検出しなくてはならないが，OpenFlow の仕様では定義されていないため Link Layer Discovery Protocol(LLDP) パケットを使って各スイッチ間のリンクを取得し，トポロジ情報として保持する．

3.2.2 参照木

OFM による通信を行うための通信網を構築する方法として，毎回送信元から最短木を構築する方法と，共通の参照木を構築して，新規配信木を構築する際に参照木

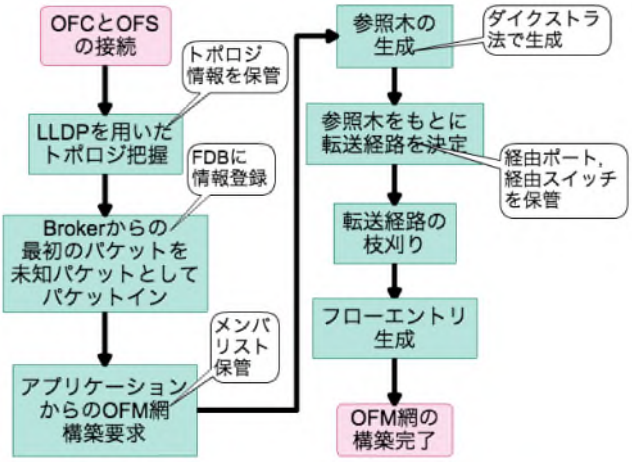

図 2　OFM 網の構築におけるデータ保持タイミング

を用いる方法が考えられる．本稿ではダイクストラ法によって参照木を 1 つ生成し，それを全ての OFM 網において共有する方法で実装した．全てのスイッチ間でのコストを算出した参照木を生成し，そのコストを用いて最短経路で OFM 網を構築する．ただし，トポロジの更新があった場合，参照木を再計算する必要がある．

3.2.3　FDB

LLDP ではスイッチ間のリンクしか検出できないため，トポロジにおける Broker の接続先は別途取得して保管しておく必要がある．L2 スイッチにおける MAC アドレステーブルのように，ある Broker がどのスイッチのどのポートの先にいるかを保持する．

3.2.4　メンバ情報

どのトピックにどのようなメンバがいるか把握するためにそれらの情報を保持しておく必要がある．保持する情報は OFM 網の構築要求に付加されるメンバリストの情報であり，これは OFS から各メンバに対してパケットを送信する際，宛先として指定されている OFM アドレスを各メンバの IP アドレス等に書き換えるために使われる．また，参加・脱退が発生した場合は，適宜メンバ情報の追加・削除を行う．

3.2.5　配信経路情報

OFM 網のための配信経路は木の形で情報を保持しておらず，複数の情報を用いて配信経路とする．その情報として OFM 網で経由する OFS や経由するポートを記録しておくことで，フローエントリを登録する OFS およびアクションにおけるパケットの入出力先ポートの決定に利用する．さらに各メンバに転送する際は，メンバ情報を利用してヘッダ情報の書き換えを行う．

4　連携機構の実装

本研究では OFC 開発フレームワークである Trema を用いて OFC の実装を行った．Trema 自体が Northbound API に対応していなかったため，本研究では図 3

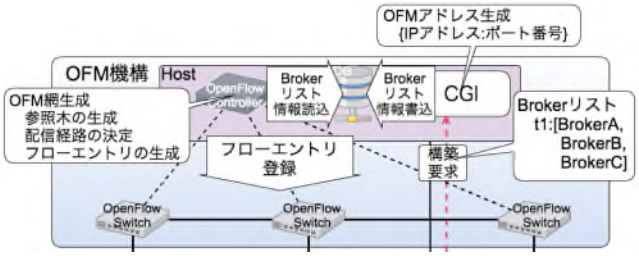

図 3　実装した OFM 機構

のように CGI を利用して Northbound API を実現している．また，OFC と CGI 間のメンバ情報や OFM 網の構築状況の同期を行うためにデータベースを用いている．

OFM は，OFC においてアンダーレイ上の OFM 網を論理的にトピックに対応させることで利用可能となる．OFM は IP マルチキャスト同様，1 つの情報をネットワーク機器上で複製・転送して配信される．そのため，ALM のようなネットワーク機器上を同一情報が何度も往復する問題が解消される．Broker は ALM と OFM の両方の LT で待ち受けており，送信時に適切な LT を選択することで ALM と OFM の切り替えならびに併用を可能にしている．これにより，あるトピックでは ALM，別のトピックは OFM を利用したり，1 つのトピックにおいてもメンバ全てを OFM に切り替えなくとも，一部だけを OFM に切り替え，残りは ALM を引き続き使用するといったことも可能になる．

OFM 管理機構は，OFGate からの要求を受け付ける NorthboundAPI モジュールと OFM 網の構築，更新，削除を行う OFM 網管理モジュールおよび，これらモジュールの連携を行うためデータベースから構成されている．

4.1　Northbound API モジュール

OFGate から送られてきた OFM 網の構築依頼の REST 要求を受理し，OFM 網管理モジュールと連携して構築を行う．REST 要求には OFM に移行するトピックのメンバである Broker の情報が付随しており，この情報は OFM 網管理モジュールが参照できるように DB の方に保管しておく．その際，トピックと対応する OFM アドレスを生成して一緒に保管する．この OFM アドレスは IP アドレスとポート番号から構成されており，これらを宛先に指定してパケットを送信することで OFM を用いた通信が可能となる．また，OFM 網の構築が完了するまでは応答を返さずに OFGate を待機させ，OFM 網の構築状況を OFGate と OFC の間で同期させる．もし，応答が遅すぎて OFGate がタイムアウトしたら，OFM を利用せずに ALM を引き続き利用する．待機により Publisher から見ると，OFM 網の構築のために遅延が発生することになる．そのため，待機による遅延を含め，その後の OFM 網によるトラフィックおよび遅延時間の削減によるメリットを考慮して，OFM 網切り替えの判断を行う必要がある．OFM 網の構築状況は，フラグを使ってデータベースで管理しており，CGI は 1 秒ごとに確認している．もし確認した際に構築完了フラグが立っていれば REST 要求への応答として OFM

アドレスを返す．OFM アドレスは配信メッセージの中にも含まれており，OFM アドレスを利用したメッセージを受信した Broker は OFM 網構築完了を知ることができ，OFM 網をすぐに活用することが可能となる．参加・脱退についても，トピックから参加・脱退するメンバの情報を受け取り，データベースの方に保管し，構築時と同様に OFM 網の更新状況のフラグを 1 秒ごとに確認している．更新完了フラグが立っていれば REST 要求の応答として OFM アドレスを返す．

4.2 OFC モジュール

REST API モジュールで OFM 網構築要求を受理した際のデータベースの更新部分を 1 秒ごとに確認した際に読み込むと構築処理を開始する．まず初めに，LLDP を使って予め把握していたトポロジ情報をもとに参照木を生成する．この参照木の生成にはダイクストラ法を用いており，生成の基準点となる Root node が必要となる．今回は Root node はランダムに決定した．参照木は 1 つだけしか生成せず，2 回目以降の OFM 網構築要求時には，生成済みの参照木を利用する．つまり，最初に生成した参照木を全ての OFM 網において共有することになる．これは，任意の Broker が配信元になることを想定し，計算時間を短縮するためであるが，配信効率を考えた場合，共有しない方が望ましい場合も考えられる．他の配信木構築手順の評価については今後の課題とする．

参照木の生成後は，参照木から Root node から各 Broker までの経路を取り出し，その情報を配信木として整理する．その後，配信木の情報を用いて，経由 OFS に経由ポートの数だけフローエントリを生成する．そのフローエントリのアクションでは，隣接した OFS に向けた出力ポートの他に，OFS から Broker に転送する場合は，データベースから取得した Broker 情報を用いてパケットの宛先 IP や MAC アドレスを各 Broker のものに書き換えるルールを指定している．このようなフローエントリが準備できたら，OFS に登録して OFM 網の構築は完了させる．最後に OFM 網の構築が完了した旨を Northbound API モジュールに知らせるため，データベースの構築状況管理フラグを構築完了にする．

メンバの参加・脱退についても 1 秒ごとに確認しているデータベースの更新されたデータを読み込むことで処理を開始する．処理としては，対象のトピックの構築状況管理フラグを構築完了から参加中，もしくは脱退中に変更する．その後参加または脱退するメンバのトポロジ上の位置を把握し，その位置に応じた参加・脱退処理を行う．参加・脱退するメンバが複数いたとしても処理は逐次的に 1 つずつ処理を行うようになっている．

上述の通り，OFM 網構築時には参照木を用いているが，参照木の構造上，OFM 網を構築する際に枝刈りが必要になるケースがある．以下，4.2.1 節では，新規 OFM 網構築時，4.2.2 節ではメンバの参加・脱退時の枝刈り処理について述べる．

4.2.1 OFM 網の最適化

OFM 網の構築は，ダイクストラ法で求めた参照木を利用しており，Root node を基準にしている．そのため，OFM 網の転送経路は基準点である Root node を必ず経由する．しかし，OFM 網によっては Root node を経由しないものもあり，単純に構築すると OFM 網の転送経路に Root node を経由するような無駄な経路が含まれてしまう．そのため，OFM 網の無駄な経路の刈り込みが必要となる．具体的には Root node から OFM 網に向かって経路をたどりながらその経路が必要かどうかを判定し，必要無い場合は刈り込みを行う．

4.2.2 Subscriber の参加・脱退

Pub/Sub 通信では，トピックに対する Subscriber の参加や脱退が発生し，OFM 利用時は OFC で参加・脱退に合わせて OFM 網の更新が必要となる．参加・脱退は幾つかのパターンに分類することができる．参加・脱退に伴って OFM 網が拡大・縮小するパターンとしないパターン，そして拡大・縮小するパターンでは，処理対象の OFM 網に Root node が含まれるパターンと含まれないパターンが考えられる．後述のものほど必要な処理が多くなる．参加処理は，参加する Subscriber から Root node までの経路を Root node に向かいつつ OFM 網に加えていき，参加対象の OFM 網にたどり付いたら処理を終える．ただし，Root node に到達しても参加対象の OFM 網にたどりつかなかった場合，OFM 網側から Root node までの経路を OFM 網に加えていく処理を行う．脱退処理は，参加時と同様，脱退する Subscriber から Root node までの経路を OFM 網から削除していく．ただし，Root node に到達しても削除を停止する条件を満たさなかった場合，保管しているトポロジ情報を用いて脱退対象の OFM 網に向かって削除処理を続行する．

5 性能評価実験

提案している ALM と OFM の連携機構は既に文献 [5] のシミュレーション評価においてトラフィック量および転送遅延の削減効果を確認しているが，OFM 切り替えによって生じるコストが明らかになっていない．本稿ではスペースの都合上詳細な手順は省略しているが，OFM 網利用時の参加・脱退処理として，ALM 網への参加・脱退と同等なコストを要する方式と，さらに OFGate を経由した集中管理による参加・脱退コストを要する方式の 2 種類を実装している．初期評価として，ALM での参加・脱退コストと OFGate による参加・脱退コストを比較し，今後はさらにキャッシュを用いた方式，ALM 網を用いてあメンバ管理方式と比較して評価をおこなう予定である．また，上記の通り，実際には実装されている方式について，キャッシュ等の機能も考慮した上で評価を行っていく必要があるが，本稿ではまず基礎調査として，ALM 網での参加・脱退コストを比較した．今後，ALM と OFM によるトラフィック量，転送遅延の削減効果と，これらのメンバ管理コストなどを総合して，切り替えの優劣を判断できる指標を確立していく予定であ

る．評価は図5のようにOFC1台，OFS8台，Broker17台で構成された環境を用いる．評価環境はネットワークエミュレータであるMininetを用いて仮想ネットワークを構築している．OFSに関してはソフトウェアスイッチであるOpen vSwitchを用いており，BrokerはLinuxのNetwork NameSpace機能を用いている．評価は2段階で行う，まず5.1節では，単一のSubscriberによる参加・脱退にかかる処理時間の計測した結果について述べる．この評価は実装したOFCにおいて，前述したパターンによって処理時間が異なるため，その影響を確認するために行った．5.2節では参加・脱退を大量に発生させた時の参加・脱退に必要な時間，各BrokerやOFSにおけるCPUの使用率を計測した結果について述べる．

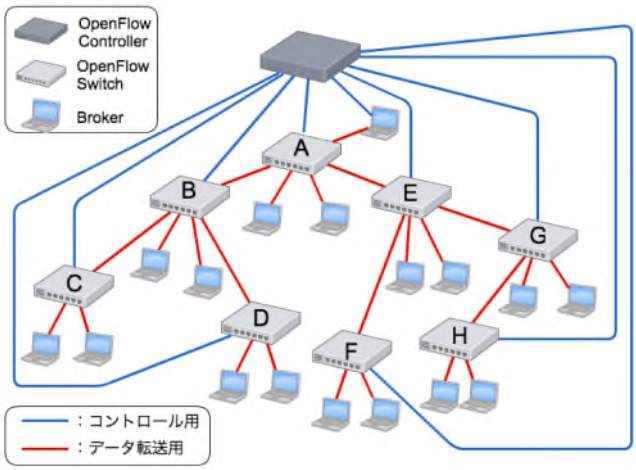

図4　評価環境

5.1　単一Subscriberの参加・脱退における処理時間

ALMによる参加・脱退処理では，オーバレイネットワーク上の近隣ノードとの論理的な接続を変更するだけだが，OFMの場合はSubscriberはOFGateを通じてOFCへ参加・脱退の要求を行い，OFCがそれに応じてOFM網の更新処理を行って，さらにOFCが返す応答をSubscriberが受信する必要があるため，ALMと比較すると処理時間は長くなると考えられた．それを確認するためにALMとOFMの双方において，OFCに実装した参加・脱退処理の各パターンの所要処理時間の計測評価を行った．予め各パターンにおける参加・脱退ができるようなOFM網を生成しておき，そのOFM網に対して参加と脱退を処理の間に数秒の間隔を空けて50回繰り返す．簡易的に行うために，同一のSubscriberの参加と脱退を交互に行っており，処理時間はSubscriberが参加または脱退要求を出してから応答が返ってくるまでの時間を計測する．なお，OFMに関してはOFCの実装上ポーリング処理を行っているため，参考としてCGIでの処理時間から1秒間のポーリング時間を除いた時間も出している．ただし，SubscriberとCGI間の通信時間が含まれておらず，OFCにおけるデータベースの読み込みにおけるポーリング時間が引かれていないことに注意が必要である．

計測した処理時間を平均した結果を図5に示す．赤系統のグラフはALMの平均処理時間，青系統のグラフはOFMの平均処理時間を示している．OFMに関してはポーリング処理を含む時間と含まない時間の2つ示している．図5を見ると，参加・脱退の両方においてALMの処理時間よりもOFMの処理時間の方が長いことがわかる．ポーリングを含む処理時間に関しては3倍以上の差が開いており，ポーリングを除いた場合でも差は縮めているがそれでもALMに比べて処理時間が長い．これは予想していた通りであり単純な参加・脱退処理にかかる時間はALMの方が短いことがわかった．つまり，OFMはALMと比べてSubscriberの参加・脱退が通信に反映されるまでに時間がかかることになり，OFM切り替え時には参加・脱退頻度を考慮する必要がある．しかし，OFCの実装をポーリング処理からイベント駆動にしたり，CGIとOFCの2つのモジュールをOFCとして実装することができれば差をより縮めることができると考えられる．

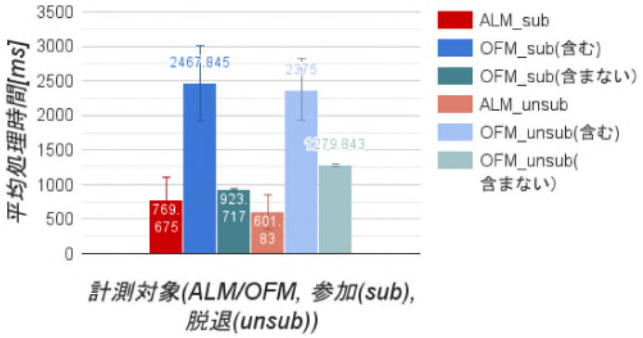

図5　ALMとOFMによる参加・脱退時間

5.2　参加・脱退におけるオーバヘッドの評価

5.1節では単発的な参加・脱退にかかる時間を計測したが，実際は多くの参加・脱退が発生する．その場合，ALMでは参加・脱退に応じてオーバレイネットワークを更新する各Broker，OFMでは参加・脱退処理時に必ず経由されるOFGateに負荷が集中すると考えられる．また，OFMでは必ずOFGateと通信を行うため，OFGateが繋がっているOFSの負荷が高くなると考えられ，さらに，フローエントリの更新処理も行われるため，各OFSの基本的な負荷は高くなると考えられる．それらを確認するためにALMとOFMの双方において，参加・脱退が多く発生した場合のOFGate，Broker，OFSのCPU使用率を計測を行った．計測は，各Subscriberの参加・脱退を30回繰り返し，その間の各BrokerやOFGateおよび各OFSの1秒ごとのCPU使用率を“ps”コマンドを利用して取得しておこなった．計測結果として，BrokerとOFGateの平均CPU使用率を図6，各OFSの平均CPU使用率を図7に示す．図6を見ると，予想していた通りOFGateの平均CPU使用率はOFMの方が高く，Broker使用率はALMの方が高い．また，各OFSにおいてもOFMのほうが全体的にCPU使用率が高く，特に参照木のRoot nodeに近づくほど高くなって

いることがわかる．つまり，OFM を利用する際，ALM と比べて Broker の負荷は低減できるが，OFGate および各 OFS の負荷は高くなり，特に OFGate の負荷が高くなっていることがわかる．本稿では Broker 数が少数であるため動作していたが，Broker 数が増加した場合，OFGate に対する負荷はより高くなると考えられる．そのため，OFM における参加・脱退の頻度を考慮する必要がある．今回計測した結果だけでは，ALM 単体に比べて OFM と連携した場合のオーバヘッドが大きいと判断できるが，文献 [5] などの既存研究では ALM 単体に比べ OFM との連携時に Pub/Sub 通信に関するトラフィック量や転送遅延時間の大幅な削減できることを示している．つまり，今回の結果と合わせて考えると参加・脱退の頻度が少ない場合においては ALM 単体よりも OFM と連携した場合に利点があると考えられる．

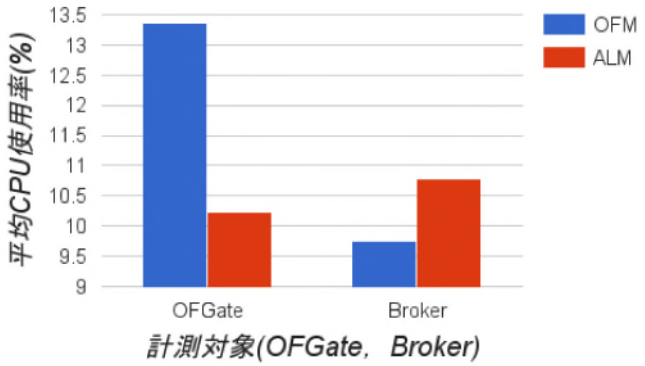

図 6　OFGate と Broker の平均 CPU 使用率

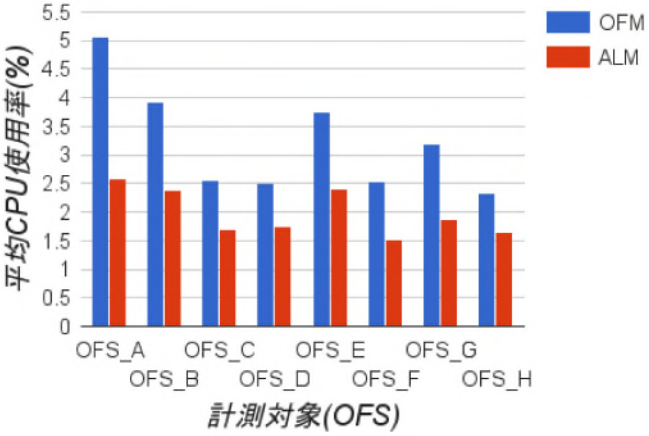

図 7　各 OFS の平均 CPU 使用率

6　まとめ

本稿では，ALM を用いた Pub/Sub 基盤と連携し，Broker 間通信の最適化を行うための OFM の設計および実装と評価について述べた．その結果，ALM と比べて OFM の方が Subscriber の参加・脱退の処理に時間がかかり，ネットワークに反映されるまでに時間がかかることがわかった．また，OFM では OFGate に負荷が集中するため，OFGate がボトルネックになる可能性が高くなると考えられる．そのため，OFM における Subscriber の参加・脱退頻度等を考慮する必要があると考えられる．今後は，ツリーの計算時間，フローエントリ数など OFM 側の性能に影響する部分があるためこの点に関しても評価していく予定である．さらに，今回は ALM か OFM かのどちらかを使う場合のみの評価を行ったため，ALM と OFM の併用時における OFC の動作確認や評価を行う必要がある．また，現在の実装では ALM に比べて処理に時間がかかっており，非効率な実装を行っている部分もあったためその改修を行う必要がある．ただし，OFC 実装のフレームワークとして利用している Trema の最新版は Pure Ruby の実装になっており，Northbound API への対応や OpenFlow のバージョン 1.3 にも対応しているため，実装の改修とともに C 言語から Ruby への移植を検討するなど ALM と OFM の連携により適した実装に改良していく予定である．

謝辞：本研究の一部は総務省 SCOPE の助成を受けたものである．

参考文献

[1] “Mosquitto,anopensourcemessagebroker”,“http://mosquitto.org/“,Aug2015

[2] “Redis,anopensourceadvancedkey-valuecacheandstore”,“http://redis.io/“,Aug2015

[3] M. Hosseini, D.T. Ahmed, S. Shirmohammadi, N.D. Georganas, “A survey of application-layer multicast protocols,” Communications Surveys & Tutorials, IEEE, vol.9, no.3, pp.58,74, 2007.

[4] Yuuichi Teranishi, Ryohei Banno, Toyokazu Akiyama, “Scalable and Locality-Aware Distributed Topic-based Pub/Sub Messaging for IoT,” Proc. of 2015 IEEE Global Communications Conference (GLOBECOM 2015): Selected Areas in Communications: P2P Networking (Dec. 2015).

[5] Toyokazu Akiyama, Yuuichi Teranishi, Ryohei Banno, Katsuyoshi Iida, Yukiko Kawai, “SAPS: Software Defined Network Aware Pub/Sub - A Design of the Hybrid Architecture utilizing Distributed and Centralized Multicast,” The 39th Annual International Computers, Software & Applications Conference (COMPSAC 2015) (Jul. 2015).

[6] Y. Teranishi, “PIAX: Toward a framework for sensor overlay network,” Proc. of 6th Consumer Communications and Networking Conference (CCNC 2009), 2009.

[7] J.Seedorf, E.Burger, “Application-Layer Traffic Optimization (ALTO) Problem Statement”, “https://tools.ietf.org/html/rfc5693”, Aug 2015