SDNソフトウェアスイッチ「Lagopus」

SDNとOpenFlowの基礎から、最新のソフトウェアスイッチ「Lagopus」の性能にフォーカスして詳細を解説します。

3.1 SDNとOpenFlow

ネットワーク関連の業務に就かれている方であれば、SDNという言葉を耳にしたことはあるかと思います。ネットワークの構成要素として設置された、ルータやスイッチなど各種専用ハードウェア機器に人手で設定を施す従来の構築作業と比較して、「オーケストレータ」と呼ばれるシステムにより集中管理された情報を用い、ソフトウェアによってネットワークの構成要素に対して設定を投入する、それがSoftware Defined Networking（SDN）の基本的な考え方です。

SDNとOpenFlowを広く推進しているOpen Networking Foundation（ONF）によると、SDNの構成要素はおおよそ3つのレイヤーに分けることができるとしています（図-1）。

- **アプリケーションレイヤー**
- **コントロールレイヤー**
- **インフラストラクチャレイヤー**

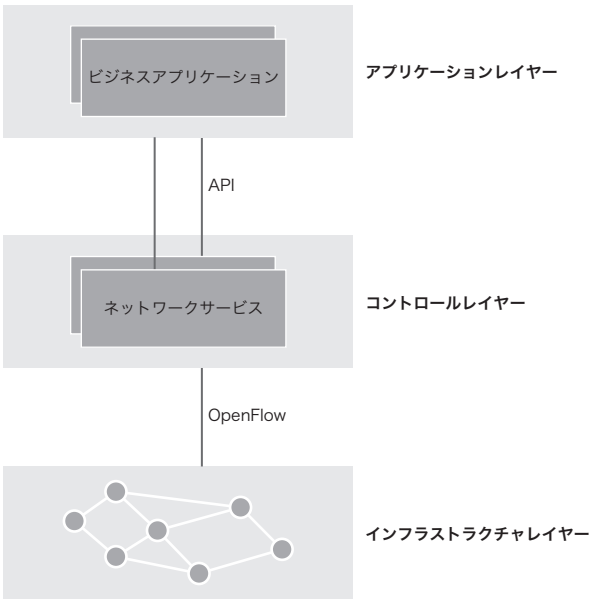

図-1 SDNの構成要素

アプリケーションレイヤーは、SDNの中核をなす集中管理とユーザインタフェースをつかさどるレイヤーで、ここに位置するものはSDNのみならずクラウド、すなわち仮想マシンについても管理を一手に引き受けます。ソフトウェアで言えばOpenStack、CloudStackといったオーケストレータが該当します。

コントロールレイヤーは、アプリケーションレイヤーとインフラストラクチャレイヤーの橋渡しの役目を担うレイヤーで、アプリケーションレイヤーから見るとネットワークサービスを提供するレイヤーとなります。アプリケーションレイヤーからの処理要求を、具体的な設定指示に置き換えて、インフラストラクチャレイヤーの各コンポーネントに設定を投入するのが主な役割です。ソフトウェアでいえばFloodlight、Ryuといったフレームワークを用います。

インフラストラクチャレイヤーは、文字どおりネットワークの基盤をつかさどるレイヤーで、顧客ネットワークのパケットを送受信するコンポーネントが該当します。広義ではコントロールレイヤーからソフトウェア制御が可能なルータ・スイッチの類となりますが、狭義では「ベンダニュートラルなAPI・プロトコルによって」制御可能なものを指すことになります。

ベンダーニュートラルなプロトコルによって制御可能なスイッチの仕様として広く知られているものとして、OpenFlow Switch Specificationがあります。この仕様ではコントローラとスイッチ間の制御プロトコルも定義されていて、OpenFlow Protocolと呼ばれます。OpenFlowはオープンな標準として定義され、自由に仕様を参照してハードウェアやソフトウェアを実装することができます。OpenFlowに対応するインフラストラクチャレイヤーのコンポーネントとして、ハードウェアスイッチ製品の他にソフトウェアとして実装されたソフトウェアスイッチがいくつも存在しています。

3.2 OpenFlow

OpenFlowは2008年にスタンフォード大学で策定され、その後仕様の拡張・修正がなされ、現在はONFによって定義されているオープンな技術仕様です。仕様にはバージョンが振られていて、現在最も多く使われているのはOpenFlow 1.0です。その後いくつかの拡張がなされ、1.1や1.2が策定されましたが、普及前にバージョン更新されたためにあまり使われず、長期サポートが謳われているOpenFlow 1.3の実装がよく見られます。執筆時点での最新バージョンはOpenFlow 1.4ですが、仕様の複雑化による実装の困難さから、対応した実装はほとんどないというのが現状です。OpenFlowでは、バージョン間の互換性を考慮しない仕様となっているために、コントローラとスイッチの間で利用するバージョンが一致していなければなりません。

OpenFlowの考え方は、制御プレーンとデータプレーンを分離するところから始まっています。データプレーンは、与えられたルールに従ってパケットを高速に処理・転送する役割、制御プレーンはルールを与える役割になります。この場合のルールとは、受信したパケットがどのようなものかを判別するものであり、OpenFlowでは1つのルールとそれに対応した処理を一組として、フローエントリと呼んでいます。例えば、「LANポート1で受信した、宛先IPアドレス192.168.0.1の、TCP80番ポート宛のパケット」といったルールを与えることができます。複数のフローエントリをまとめたものをフローテーブルと呼び、OpenFlowで処理されるパケットはフローテーブルの中から該当するフローエントリ1つを見つけ出し、関連付けられているアクションを実行することになります。アクションには、単純に指定されたLANポートへパケットを送信するといったもののほか、宛先IPアドレスの書き換え、VLANタグの挿入といったパケットヘッダの変更などもあり、これらは仕様に定められた範囲で自由に指定することが可能です。フローテーブルを複数用意しておき、アクションとして次のフローテーブルの処理をせよ、と指定することもできます。

また、アクションにはパケットを制御プレーンつまりコントローラに送信するという指定が可能です。コントローラはパケットをスイッチから受け取り、より複雑な処理を実行することができます。更に、コントローラはスイッチに対して、任意のパケットを送出させることもできます。これらをまとめると図-2のようになります。

このように、OpenFlowではコントローラとスイッチの連携によって、高速性を保ったまま複雑なパケット処理を実現することを考慮して考案されました。しかし、OpenFlow仕様自体の拡張によって、データプレーンは複雑化の一途をたどっています。

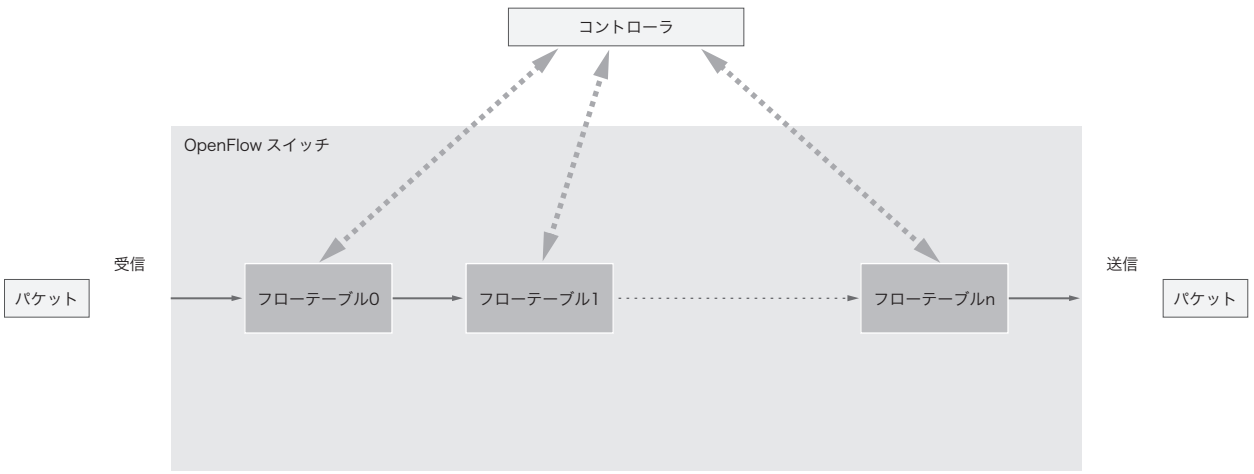

図-2 OpenFlowの考え方

3.3 ソフトウェアスイッチ

OpenFlowに対応するEthernetスイッチの実現方式としては、大まかに3つに分類できると言えます。

1つ目は、ASICを用いたハードウェアスイッチ製品です。スイッチ容量は大きく、大規模ネットワークに対応できる性能を持てる半面、扱えるフローエントリ数やテーブル数に相当な制限がかかり、かつ価格も高価です。また、OpenFlow仕様の項目には必須なものとオプションとして省いていいものとがありますが、ハードウェアスイッチ製品では実装の難しい処理が省かれていることが多い点に注意が必要です。

2つ目は、ネットワークプロセッサ(NPU)を用いた製品です。ネットワーク処理のための特殊な機能を内蔵したプロセッサを利用し、その機能を活かすファームウェアを内蔵したものです。ASICを用いたスイッチと比較すると、フローエントリ数、テーブル数、実装項目共に制限は緩く、また速度もハードウェアスイッチには及ばないことがあるものの、大規模ネットワークにも適用可能と言えます。やはり価格は高価であり、中規模あるいは小規模ネットワークに適用するにはコストが見合いません。

3つ目は、ソフトウェアスイッチです。サーバの汎用OS(例えばLinux)で動作するプログラムによってOpenFlowの機能を実現します。仮想マシン上で動作させることもできることから、クラウドとの相性がいいという利用面の性質があります。機能的には実装次第でOpenFlowの全機能を提供可能な半面、性能についてはASICやNPUの足元にも及ばないというのがこれまでの現状であり大きな課題でした。

10GbEネットワークインタフェースで64バイトのパケットをワイヤーレートで処理すると、14.88Mフレーム/秒(fps)となり、1フレームあたりにかけることのできる時間は67ナノ秒という計算になります。プロセッサクロックが2GHzとすると、1クロックは0.5ナノ秒になります。大雑把にいえば、1クロック1命令と仮定した場合、134命令で1パケットの処理を完了させなければワイヤーレートでのパケット転送は実現できないということになります。2012年当時、既存のソフトウェアスイッチ実装ではせいぜい数百K〜1Mフレーム/秒であったことから、性能面でのボトルネックを解消すべく、新しいソフトウェアスイッチの実装が検討されました。そうして開発されたのがSDNソフトウェアスイッチLagopus(ラゴパス)です。

3.4 SDNソフトウェアスイッチLagopus

Lagopusは、OpenFlow 1.3仕様に準拠したソフトウェアスイッチです。2013年に開発が始まり、2014年7月にApache Licenseでソースコードが公開されました。2014年10月現在の最新バージョンは0.1.1で、現在も開発が続いています。Lagopusの主な特徴を列挙すると下記のとおりになります。

- **64 Linux用ソフトウェアスイッチ**
- **1プロセス、マルチスレッド動作(2コア以上必要)**
- **開発言語は主にC言語**
- **OpenFlow 1.3.4準拠**
- **OpenFlow仕様に記述されているオプションをほとんど実装**
- **MPLS - VLAN変換を伴うパケット転送で10GbEワイヤーレートの性能**
- **フローエントリ数:100万、テーブル数:254の大規模設定に対応**
- **Intel DPDKを利用したユーザスペースによる高速パケット処理**
- **Apache Licenseによりソースコード公開、非商用・商用を問わず利用可能なオープンソースソフトウェア**

本稿では、これらの特徴のうち性能にフォーカスして、詳細を解説します。まず、ソフトウェアスイッチにおいて性能に大きく影響の出る主なポイントについて書き出してみます。

3.5 ソフトウェアスイッチのボトルネック

3.5.1 受信割り込み、送信完了割り込み処理

CPUが様々な処理を実行している最中に、ネットワークインタフェースカード(NIC)がパケットの受信を検知すると、NICはパケットのバイト列をバッファメモリに格納した上で、CPUに対して割り込みを発行します。CPUはNICからの割り込みを検知すると、処理中の状況(コンテキスト)を一時的に退避して、割り込み処理に移ります。割り込み処理では、バッファメモリに格納されたパケットの情報を取り込み、ネットワークスタックの受信キューに詰めるなど最低限の処理が実行され、処理が完了すると退避していたコンテキストを復帰させ、割り込みがかかる前の処理を続行します。このコンテキストの切り替わりのことをコンテキストスイッチと呼ぶのですが、高速なパケット転送をする上ではこれが大きな負荷となります。1パケットずつ割り込みを発生させていたのでは性能に著しい影響があるため、NICの機能としてある程度まとまったタイミングで割り込みを発生させるといった負荷を軽減する工夫があります。

3.5.2 パケットのメモリコピー

また、受信したパケットはまずOSのカーネル空間で管理されるメモリに記録されますが、通常これを直接ユーザスペースのプログラムから読み書きすることはできません。システムコールread(2)あるいはrecv(2)は、ユーザプログラム側であらかじめ確保したメモリ空間に、受信したパケットの内容をコピーします。逆に、システムコールwrite(2)あるいはsend(2)は、ユーザプログラムが用意した内容をカーネル内のメモリ空間にコピーします。10GbEワイヤーレートであれば1秒間に14.88Mフレームを送受信することになりますので、64バイトのパケットであれば秒間14.88M×64×2=1.9GBのメモリコピーが発生することになります。DDR3-1600のメモリ転送速度の理論値は12.8GB/秒ですので、単純に当てはめたとしてもメモリコピーだけで全体時間の約15%を消費することになります。

3.5.3 OSによるプロセススケジューリング

OSのユーザスペースで動作しているプロセスは複数ありますが、psコマンドなどで見れば分かるとおりマルチコア全盛の現在においても搭載されているCPUのコア数で賄える数にとどまることはありません。そのため、OSは適宜動作させるプロセスを切り替えて、見た目上同時に動いているように見せています。ミリ秒単位で切り替わっていくため通常人間が触っている限りでは気付かないほど高速ではあるのですが、ソフトウェアスイッチの動作中にこのようなOSによるプロセススケジューリングが発生すれば、当然ながら性能に大きく影響します。

3.5.4 メモリブロックの確保と解放、ページフォルト

読み書きをする際に使用するメモリブロックについては、受信のたびに確保し、送信完了のたびに解放するのは効率的ではなく、すぐにまた次のパケットを受信するということを考えると領域を再利用することにより性能の低下を軽減できます。しかし、ユーザスペースにあるメモリはすべてが実際のメモリ上に置かれているとは限りません。OSは必要に応じてユーザスペースのメモリと実際のメモリをマッピングします。必要に応じてというのは、プログラムがそのメモリにアクセスしようとした瞬間です。この仕組みをオンデマンドページングと呼びますが、処理としてはプログラムがメモリにアクセスしようとした瞬間に例外が発生し、例外処理の中でメモリのマッピングを実行し、復帰後プログラムは何事もなかったかのように処理を実行し続けます。この例外をページフォルトと呼びます。ページフォルトが発生してメモリのマッピング処理が実行される場合と、既にメモリがマッピング済みである場合とで、処理時間に大きな差があることは自明です。

3.5.5 リソースへの排他アクセス

OpenFlowではフローテーブルを参照してパケット処理を実行することは説明しましたが、当然ながらパケット処理中にもフローテーブルへのエントリの追加・変更・削除が可能です。しかし、受信したパケットにマッチし今まさに参照しているフローエントリが削除されることがあってはなりません。そのため、パケット処理とフローテーブルへのエントリ追加・変更・削除は互いに排他的に実行できなければなりません。排他アクセスが必要なリソースはフローテーブルのみではなく、例えばLANポートのカウンタ情報などがあります。通常このような用途にはリーダ・ライタロックと呼ばれる仕組みを使うのですが、ロックは比較的重い処理で、1パケットごとにロック・アンロックを繰り返していては転送性能に大きな影響が出ます。

3.6 ボトルネックを解消するIntel DPDK

前出のボトルネックのうちのいくつかを解消しようという試みは、既にいくつかありますが、ここではLagopusで利用しているIntel DPDKの取り組みを紹介します。Intel DPDK (Data Plane Development Kit)は、データプレーンをプログラミングするにあたって有用なAPIを提供するライブラリであり、高速なパケットI/Oを実現するフレームワークとなっています。執筆時点での最新バージョンは1.7.1です。Intel DPDKの大きな特徴は下記のようなものです。

- **x86、x86_64 Linux及びFreeBSDで動作**
- **BSD Licenseで提供されているため、商用・非商用を問わず利用可能**
- **割り込みを用いないパケット送受信を実現するPoll Mode Driver(PMD)**
- **ユーザスペースI/Oによるゼロコピー**
- **マルチコア前提、コア固定によるOSのスケジューリング回避**
- **hugepageを利用したページフォルトの削減**
- **Longest Prefix Match、hash、リングバッファなど最適化されたライブラリ**
- **サンプルアプリケーションを多数同梱**

3.7 Lagopusにおける性能向上策

Lagopusでは、Intel DPDKを利用することで、性能ボトルネックの多くを解消しています。とりわけ、PMDとコア固定は大きく性能向上に貢献しています。また、Lagopusはスレッド構成が特徴的と言えます(図-3)。

Lagopusのデータプレーンは、パケットI/Oを担当するスレッドとOpenFlow処理を担当するスレッドが分かれています。更に、I/Oは受信(RX)と送信(TX)に、OpenFlow処理(worker)は複数のスレッドとなるよう起動時に指定が可能です。これらのスレッドは、それぞれ他の処理が割り当てられないよう別々のコア上で動くよう固定させることで、性能を確保しています。また、スレッド間の通信には、DPDKが提供しているロック処理不要のリングバッファを用いています。

パケットI/Oの基盤部分は主にIntel DPDKを利用することでボトルネックを解消していますが、OpenFlow処理部分ではLagopus独自の性能向上策が見られます。Lagopusのフローテーブルの内部構造を図-4に示します。

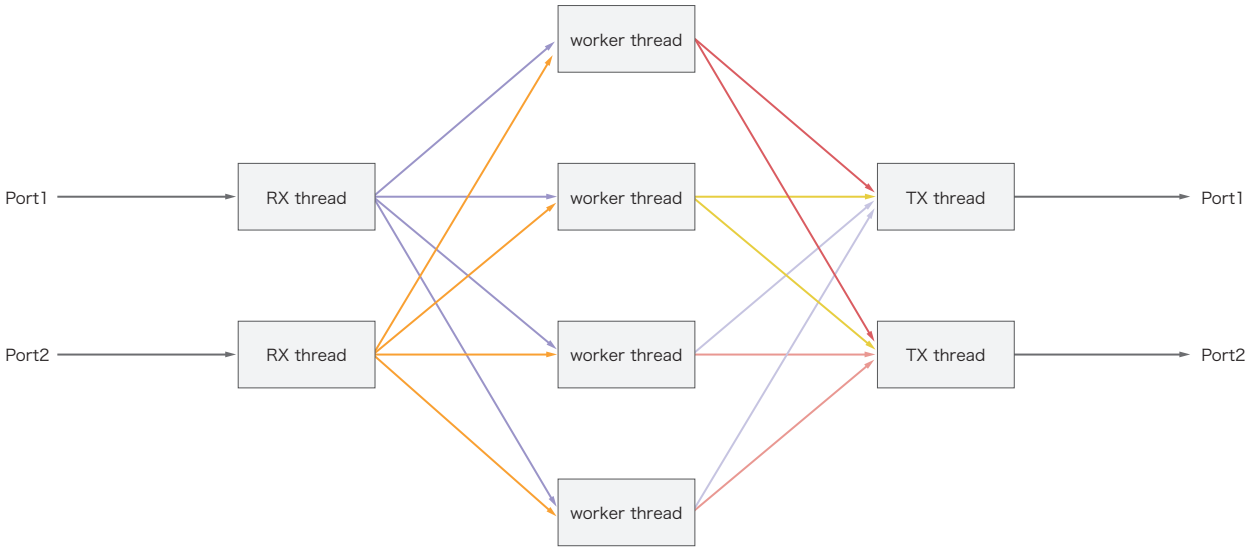

図-3 Lagopusのスレッド構成

コントローラからの要求に応じてフローテーブル内容を出力する必要があるため、OpenFlow仕様を素直に表現したフラットなテーブルが用意されていますが、Lagopusではそれとは別にフロー探索に適した構造のテーブルが用意されています。この構造は現在のところ固定的で、どんなフローテーブルでも最適な探索ができるというものではありませんが、VLAN IDによる分岐、MPLSラベルによる分岐、IPv4宛先アドレスによる分岐、といったように、典型的に使われるであろうパケットヘッダに対して高速な探索を実現しています。また、一度探索したパケットの情報をキャッシュし、同じパケットを受信した際には、キャッシュからフローエントリ情報を取得し探索自体を省略するという、フローキャッシュの仕組みも実装されています。

以上の性能向上策の相乗効果により、Intel Xeonを用いた試験環境において、10万エントリのフローテーブルを投入した状態で、OpenFlowによるパケットヘッダ書き換えを伴うパケット転送にて、Lagopusは10GbEワイヤーレート（14.88Mfps）を達成することができました。そして現在Lagopusは、より複雑なフローエントリでの処理速度向上や、より多くのポートを同時に処理する際の性能の維持、更に、より高速なインタフェースへの対応を順次進めています。

3.8 まとめ

SDNソフトウェアスイッチLagopusは、OpenFlow仕様への準拠度も高く、従来のソフトウェアスイッチと比較して飛躍的な性能向上を実現しています。現在Lagopusは試験環境以外でのテストや動作確認が不十分なプロトタイプ品質のソフトウェアですが、実環境にて幅広く使用されることを想定して開発が続いています。もしご興味がありましたら是非ホームページ（http://lagopus.github.io）をご覧ください。

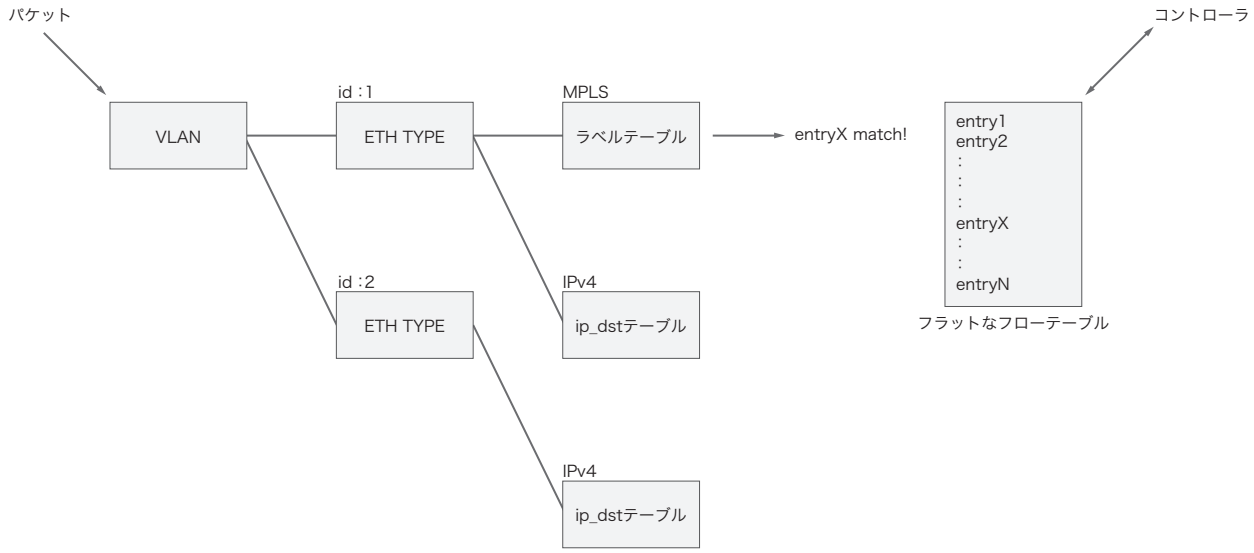

図-4 Lagopusのフローテーブルの内部構造

執筆者：

沖 勝（おき まさる）
株式会社ストラトスフィア 研究開発部。