

1.2.3 研究開発項目①-2 「光電ハイブリッドスイッチ制御アルゴリズム」

【成果の概要】

光電ハイブリッドスイッチシステムプロジェクトにおいて、本研究開発項目ではディスアグリゲーション型次世代データセンタ向けアーキテクチャに有効なトラフィック振り分けを実現するためにトラフィック振り分けアルゴリズム及びネットワークコントローラの高速制御技術の開発を進めてきた。本項では試作した光電ハイブリッドスイッチシステムを用いた実証・評価について報告する。

主な成果は以下の通りである。(1) ネットワークコントローラに組み込む振り分けアルゴリズムとして、サービス駆動方式とデータ駆動方式の2つのアプローチから検討を行い、光電ハイブリッドスイッチシステムと親和性があり、分散深層学習に効果がある振り分けアルゴリズムを提案した。

(2) 1000 ラック規模まで拡張可能、かつ 100 マイクロ秒で光スイッチ制御が可能な高速コントロールプレーン技術を提案した。(3) 多様な光部品の機能・状態・構成をコントローラから統一的に操作可能な表記法を開発した。これらの機能を、ネットワークコントローラプラットフォームのオープンソースモジュールとしてその機能を実装し、光電ハイブリッドスイッチシステムとしての動作を実証した。

【最終目標】

研究開発項目⑤「光電ハイブリッドスイッチシステム実証」と連携して、FPGA を用いたコントローラのサブシステムを含めて、光電ハイブリッドスイッチシステムとしての動作確認を行い、トラフィック振り分けアルゴリズムを確立する。また、開発した成果物をオープンソースソフトウェアとして公開する。

【成果】

<光電ハイブリッドスイッチ制御アルゴリズムの課題設定とアプローチ>

デジタル化社会を支える ICT 基盤であるデータセンタにおいて、今後予想されるデータ処理量、および通信量の増加に対応するため、データセンタにおける光ネットワーク技術への期待が高まっている。例えば、光回線スイッチは、電気パケットスイッチと比較すると、経路の切替に時間を要するという欠点がある一方で、通信帯域の増加に対して消費電力が一定であるという利点がある。そこで光スイッチと電気スイッチを相補的に組み合わせた光電ハイブリッドスイッチシステムは、通信帯域の拡張と消費電力の削減の両面から有望な技術である[1][2]。

佐藤[2]は、空間および波長ルーティングスイッチを効率的に組み合わせた大規模高速光回線スイッチアーキテクチャを提案しており、1000 ラック以上の規模のデータセンタにも適用可能なことを理論的に示している。また、多ポート化が容易であり、切替速度も 100 マイクロ秒を切る PLC (Planar Lightwave Circuit) スwitch[3]やシリコンフォトニクススイッチ技術が実用フェーズに近づいている。

光電ハイブリッドスイッチネットワークの典型的な構成例を図 1.2.3-1 に示す。データセンタでは、サーバが収容された複数のラックが大容量・高速ネットワーク（データセンタネットワーク）を介して相互接続されている。このデータセンタネットワークは電気パケットスイッチから構成される電気ネットワークと、光回線スイッチから構成される光ネットワークからなる。各ラックには ToR (Top of Rack) スwitchが設置され、それは光コアスイッチと電気コアスイッチ双方へのインタフェースを有

している。

一般的にデータセンタを流れるフローには大容量だが数が少ないエレファント（elephant）フローと、小容量だが数の多いマウス（mice）フローが混在していることが知られている。光電ハイブリッドスイッチネットワークでは、エレファントフローは光コアスイッチ経由で、マウスフローは電気コアスイッチ経由でという具合にフローの特性に応じてネットワーク経路を切り替えることで、それぞれのスイッチの利点を活かすことが可能である。なお以下では、光コアスイッチや電気コアスイッチ等の資源管理を担い、経路切替を制御するサーバをネットワークコントローラ（図中では、以下「コントローラ」）と呼ぶ。また、ネットワークコントローラと各種スイッチを結ぶ制御ネットワークを含めてコントロールプレーンと呼ぶ。

本研究開発項目では、ネットワークコントローラに組み込む振り分けアルゴリズムを各種比較検討し、ディスアグリゲーション型次世代データセンタ向けアーキテクチャに有効な方式を明らかにする。またその振り分けアルゴリズムを実装するネットワークコントローラの機能概要と仕様概要を明らかにし、参照実装をオープンソースソフトウェアとして公開する。

また、1000 ラック以上の規模にもスケールし、100 マイクロ秒以下の経路切替が可能になる多ポート高速光スイッチの特性を活かすためには、これを制御するコントロールプレーンのスケーラビリティ及び高速化が必須となる。そこで本研究開発項目では、産業用途のリアルタイム Ethernet 規格（IEC61158）である EtherCAT [4]を活用した高速コントロールプレーン技術を提案する[5][6]。

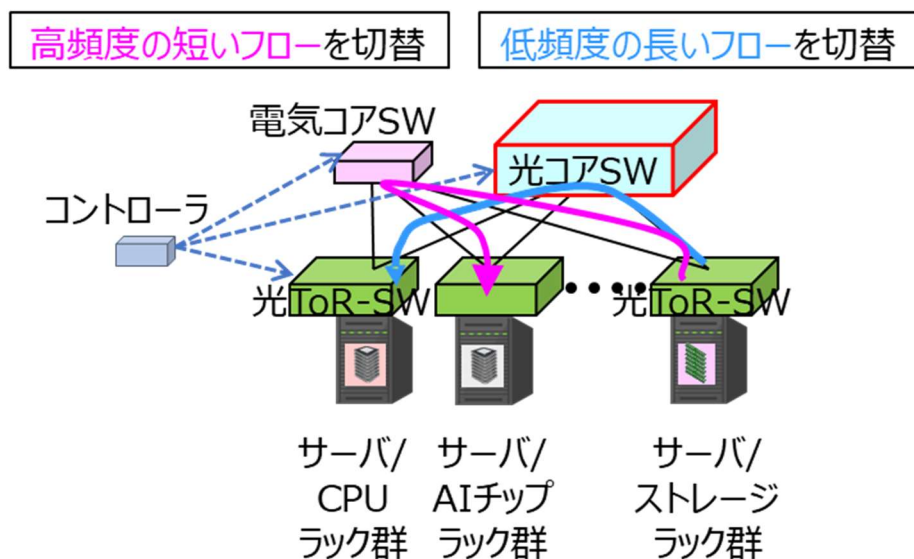


図 1.2.3-1 光電ハイブリッドスイッチネットワーク

<トラフィック振り分けアルゴリズムの開発>

光電ハイブリッドスイッチネットワークの実現方式は、経路の切替点と、経路を切り替えるための制御トリガの2つから整理できる。経路の切替点として、エンドポイントであるサーバで切り替える方法と、サーバラックに設置されたToRスイッチにて切り替える方法が考えられる。本研究開発項目では、ToRスイッチとしてOpenFlow [7]スイッチを用いることで、トラフィックを振り分けることを前提とする。制御トリガは、サービス駆動とデータ駆動があり、トラフィックの性質と用途に応じて使い分けることができる。サービス駆動方式は、アプリケーションが送信パケットに付与するヒント情報（タグ）を基にフローを切り替える方式であり、プロアクティブに素早い（数十マイクロ秒）切

替に有用である。一方、データ駆動方式は、ToR スイッチから得られた統計情報を基にフロー又はフロー群を切り替える方式であり、リアクティブ制御のため、より幅広いシーンでの光スイッチ活用が可能となる。しかし、現行の OpenFlow スイッチ等の汎用スイッチを用いる場合、切替時間が課題となる。

なお、サービス駆動方式とデータ駆動方式は共存が可能であり、個別のデータセンタにおいて、サービスに応じた最適化と自律的なトラフィック最適化を両立できる。本節では、近年のデータセンタにおいて主要なワークロードとなりつつある分散深層学習のトレーニングに適したサービス駆動方式のトラフィック振り分けアルゴリズムを提案する。

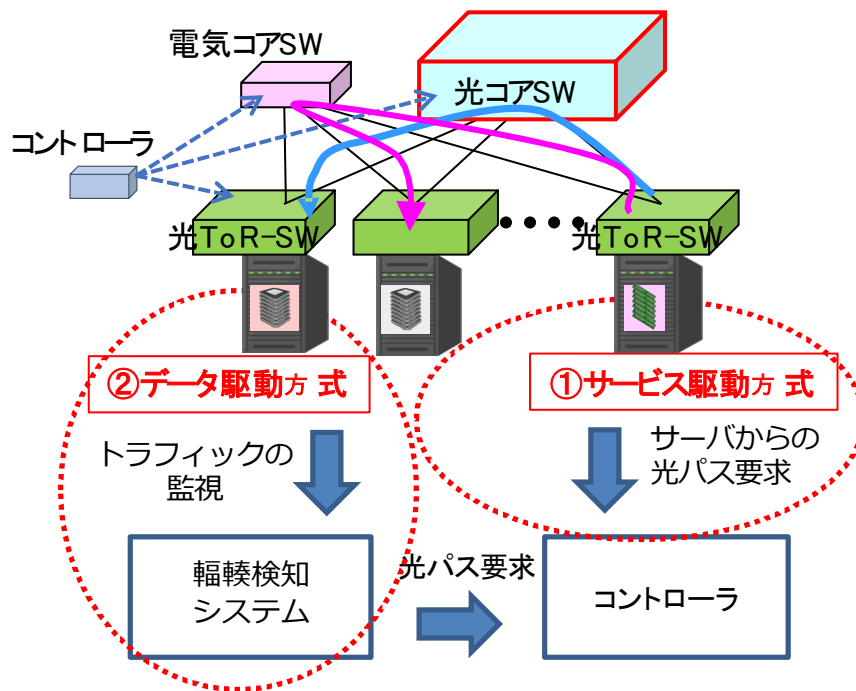
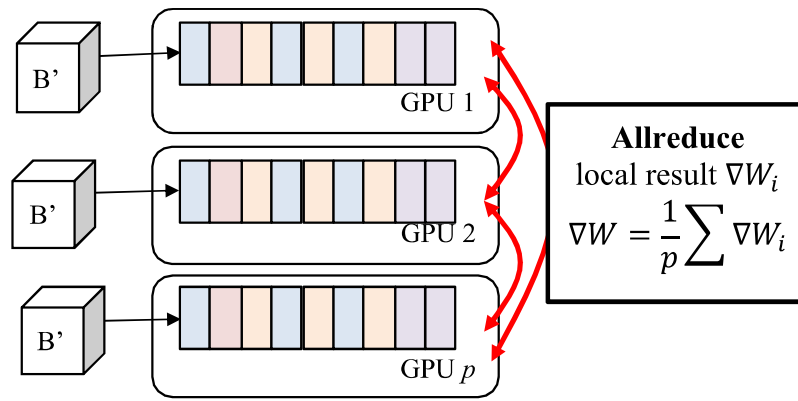


図 1.2.3-2 2種類のトラフィック振り分けアルゴリズム

深層学習や高性能計算シミュレーションで使われる典型的な集合通信アルゴリズムの一つに AllReduce があり、各プロセス（サーバ）に分散されたデータの合計や最大値を求める集約演算を実行するために用いられる。分散深層学習においては AllReduce にかかる通信時間が最も支配的である。図 1.2.3-3 に分散深層学習での AllReduce が使われる様子を示す。深層学習では学習モデルのパラメータ数や学習に用いるデータ量に比例して、学習モデルの性能が向上することが知られている。処理の大規模化に合わせて使用する計算サーバ数も増加の一途をたどっているが、計算サーバ数に比例して、演算時間に対する通信時間が増加し、通信がボトルネックになる傾向にある。したがって、AllReduce を光スイッチネットワークにオフロードすることは、性能面からも消費電力の面からも利点があると考ええる。



- 異なるデータセットを使って各GPUで学習
- 学習結果を**Allreduce通信**で平均化
- 以上を学習が収束するまで繰り返す

図 1.2.3-3 データ並列化分散深層学習の概要

AllReduce の実装方法として様々なアルゴリズムが考案されているが、プロセス間で論理リングを形成し、隣のプロセスとのみデータ交換を行うリングベース AllReduce アルゴリズムは、帯域を効率的に使うことができることが知られている。さらに論理リングは、AllReduce 実行中は不変であるため、経路切替を行う必要がない。換言すると、リングベース AllReduce アルゴリズムは、光回線ネットワークと相性がよい。図 1.2.3-4 に示す例では、アプリケーション中の AllReduce の呼び出しをトリガとして、プロセス 1→プロセス 2→プロセス 4→プロセス 3→プロセス 1 と論理リングを形成し、関数終了時に論理リングを解放する。この際、各プロセス間は光スイッチを介して、光パスを確立することになる。本節では非常に単純なリングの例を示したが、計算サーバが複数の GPU を持っているなど、計算サーバ内の並列度も考慮した階層的な、より大規模なシステムに対応する手法[11]も提案したが、本報告では詳細を割愛する。

提案した AllReduce アルゴリズムの効果を、計算機シミュレータ SimGrid を用いて検証した（図 1.2.3-5）。SimGrid には、MPI プログラムを実際に実行して取得したプロファイル情報を用いて、通信や演算処理をシミュレーションすることが可能であるので、今回はこの機能を用いた。なお、実環境における実装方法は後述する。

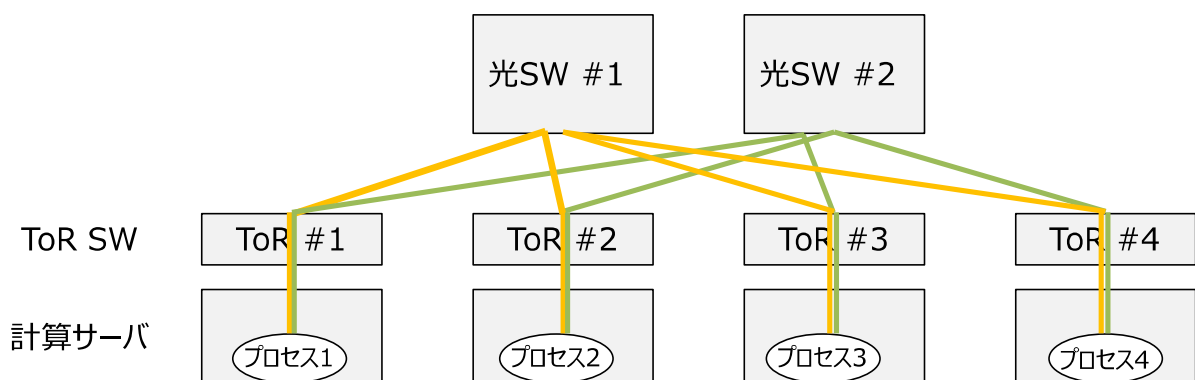


図 1.2.3-4 リングベース AllReduce における論理リング形成。

黄色線は光スイッチ#1 経由、緑色線は光スイッチ#2 を経由するものを示す。この例では、プロセス 1→プロセス 2→プロセス 4→プロセス 3→プロセス 1 と論理リングを形成している。

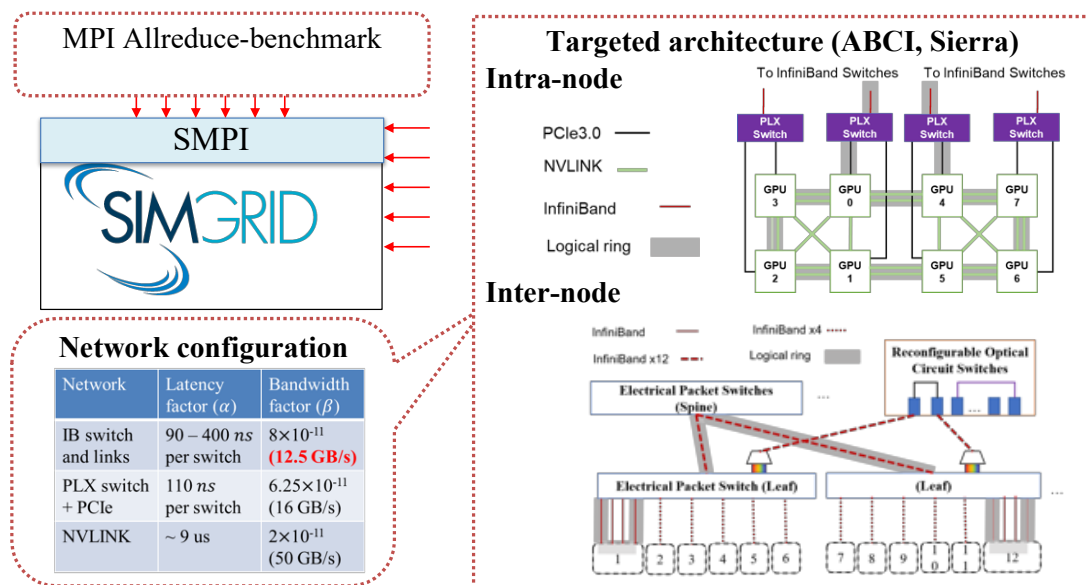


図 1.2.3-5 SimGrid を用いたシミュレーション実験

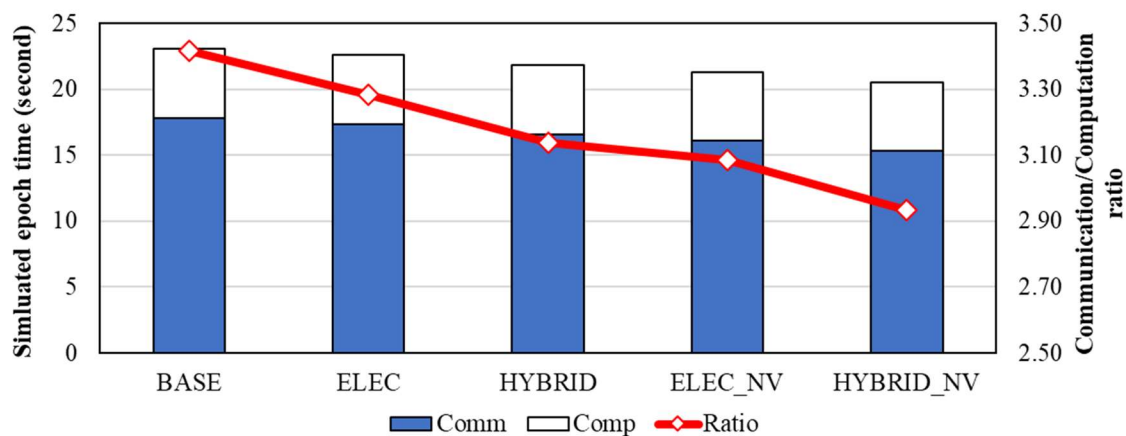


図 1.2.3-6 256GPU 使用時における実行時間の内訳

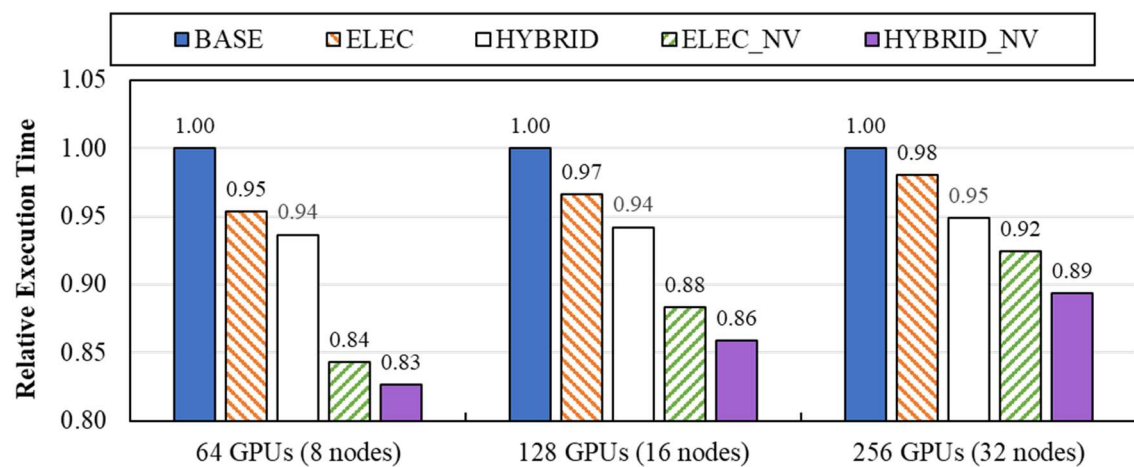


図 1.2.3-7 Flat ring-based Allreduce アルゴリズムの性能評価

256 基の GPU (32 台のサーバ) を用いて ResNet50 という学習モデルのトレーニングを行なった場合のシミュレーション結果を図 1.2.3-6 と図 1.2.3-7 に示す。HYBRID 及び HYBRID_NV が提案手法であり、NV はサーバ内の GPU 通信を高速化した場合の結果である。図 1.2.3-6 では演算時間と通信時間の比率を示している。提案手法により通信時間が削減でき、「通信／演算」比が減少していることがわかる。図 1.2.3-7 はサーバ台数を増やした場合の実行時間の変化を示している。提案アルゴリズムは電気スイッチのみを使った場合にも効果があることがわかるが、ハイブリッドスイッチを用いることでさらに数パーセント実行時間を削減できる。さらに効果はサーバ台数が増えるに従って大きくなる。これは大規模化するほど電気スイッチのホップ数が増えるからである。

この結果より、光コアスイッチの切替時間が数十マイクロ秒から 100 マイクロ秒と十分短ければ、多段の電気スイッチ構成よりも高速化できることを確認できた。

<EtherCAT を用いた光スイッチ制御>

前節で示したように、振り分けアルゴリズムの効果を活かすためには、光スイッチ自体の高速化のみならず、光コアスイッチへ経路切替を素早く通知するコントロールプレーンの高速化が必要になる。

データセンタネットワークは Ethernet 技術に基づいて構築されることが一般的であり、コントロールプレーンのために独自ネットワーク技術を導入することは、コストや運用の面から現実的ではない。また、TCP/IP ベースの制御では数十マイクロ秒で確実な応答性能を実現するのは困難である。そこで、産業用途のリアルタイム Ethernet 規格 (IEC61158) である EtherCAT [4] を利用する着想を得た。EtherCAT は確実な応答性能と高精度同期性能から、半導体製造装置、工作機械といった産業用製造装置の内部通信バスや、工場内通信ネットワーク等に採用されている。

まず EtherCAT ネットワークを構成する要素について簡単に説明する。一つのネットワークセグメントには一つのマスタと複数のスレーブが存在し、ネットワークのトポロジはデジーチェーンやリングが一般的である。モータやアクチュエータ、シーケンサなどがスレーブとなる。概念的には、マスタとスレーブは固定長のメモリ空間を共有し、このメモリ空間はスレーブごとに分割されている。EtherCAT におけるプログラミングは、このメモリ空間の読み書きを介して、機器を制御したり、その状態を監視したりすることになる。メモリ空間は Ethernet フレーム (以下、テレグラムと呼ぶ) にマッピングされ、マスタからスレーブに周期的に送信される。この周期を PDO (Process Data Objects) 周期と呼ぶ。PDO 周期の開始時にマスタからテレグラムが送出され、テレグラムはスレーブ間をバケツリレー方式で転送する。その過程でスレーブはテレグラムに割り当てられた各自のメモリ空間を読み書きする。

提案方式では、空間スイッチや波長スイッチといった光コンポーネントにスレーブ機能を実装することで、マスタからその制御を一括して行う。詳細は後述するが PDO 周期を数十マイクロ秒に設定することが可能であり、サブ 100 マイクロ秒での動作が期待できる。さらにネットワークセグメントを増やして、マスタの並列度を上げることで大規模なデータセンタネットワークにも対応可能である。

なお、現時点で EtherCAT の理論送信レートは 100 Mbps である。現時点ではギガビットイーサネットへの対応もアナウンスされており、以下ではこれを EtherCAT G と呼ぶ。

続いて、EtherCAT 技術を用いて、1000 ラック以上の規模の光ネットワークに対して、100 マイクロ秒以下での経路切替が可能になるか検討する。要求に従って光パスを構築するネットワーク構成時間 CT は、各 EtherCAT スレーブでの処理時間の合計 (ST) とネットワーク伝搬遅延 (D) の和 ($CT = ST + D$) とおおよそ見積もることができる。EtherCAT スレーブの処理時間は、0.8 マイクロ秒で一定と

仮定する。これは EtherCAT スレーブが用いている ASIC の性能を考えると妥当な値と考える。さらに各スレーブのデータサイズを 8 バイト、マスタの並列度を 16 と仮定する。ST はスレーブ数と 0.8 マイクロ秒の乗算から、D はテレグラムサイズ（ヘッダサイズ＋スレーブ数×8 バイト）とデータ転送レートの除算から求めることができる。表 1.2.3-1 に、EtherCAT と EtherCAT G におけるネットワーク構成時間 CT を示す。これより、100 マイクロ秒以内の構成時間の制約を満たしつつ、EtherCAT の場合で 960 スレーブ、EtherCAT G の場合で 1280 スレーブに対応できることがわかる。スレーブは 1 ラックに対応するので、本方式は 1000 ラック規模のデータセンタにも対応できると言える。

表 1.2.3-1 光スイッチ制御性能の見積

	#slaves/master	ST	DT	CT
EtherCAT	60	48 マイクロ秒	40 マイクロ秒	88 マイクロ秒
EtherCAT G	80	64 マイクロ秒	5.3 マイクロ秒	70 マイクロ秒

EtherCAT を用いた光コアスイッチ制御の実現性を確認するために、図 1.2.3-8 に示すように 2 基の 1x2 ファイバスイッチを用いた簡単な実験系を構築した。EtherCAT マスタとしてオープンソースソフトウェアの SOEM (Simple Open EtherCAT Master) を、スレーブとして Microchip 社の EVB-LAN9252-DIGIO 評価ボードを用いた。SOEM は C 言語のライブラリであり、Linux OS 上に SOEM を用いた単純なコントローラを実装した。EVB-LAN9252-DIGIO はデジタル I/O ピンを有しており、このピンを介して 1x2 ファイバスイッチの経路を制御する。

実験で得られたタイミングチャートの一例を図 1.2.3-9 に示す。この場合、変更要求を受信してから光スイッチの切替が完了するまで 144 マイクロ秒程度かかることがわかる。PDO 周期の開始 (③) から設定完了 (④) までの時間が 73 マイクロ秒であり、この処理時間はほぼ一定である。一方で、③から④にかけての時間は、PDO 周期開始のタイミングに依存し、最悪の場合は PDO 周期分待つ必要がある。この結果より、EtherCAT を用いて光ネットワークを制御することは原理的には可能であるが、ソフトウェア実装による性能上の課題が明らかになった。具体的には、(1) PDO の最小値が 200 マイクロ秒である、(2) パケット処理が遅い、という理由から、多ポート高速光スイッチの制御に求められる時間制約とスケラビリティを満たすためには、EtherCAT マスタのハードウェア化が必要であるとの結論に至った。

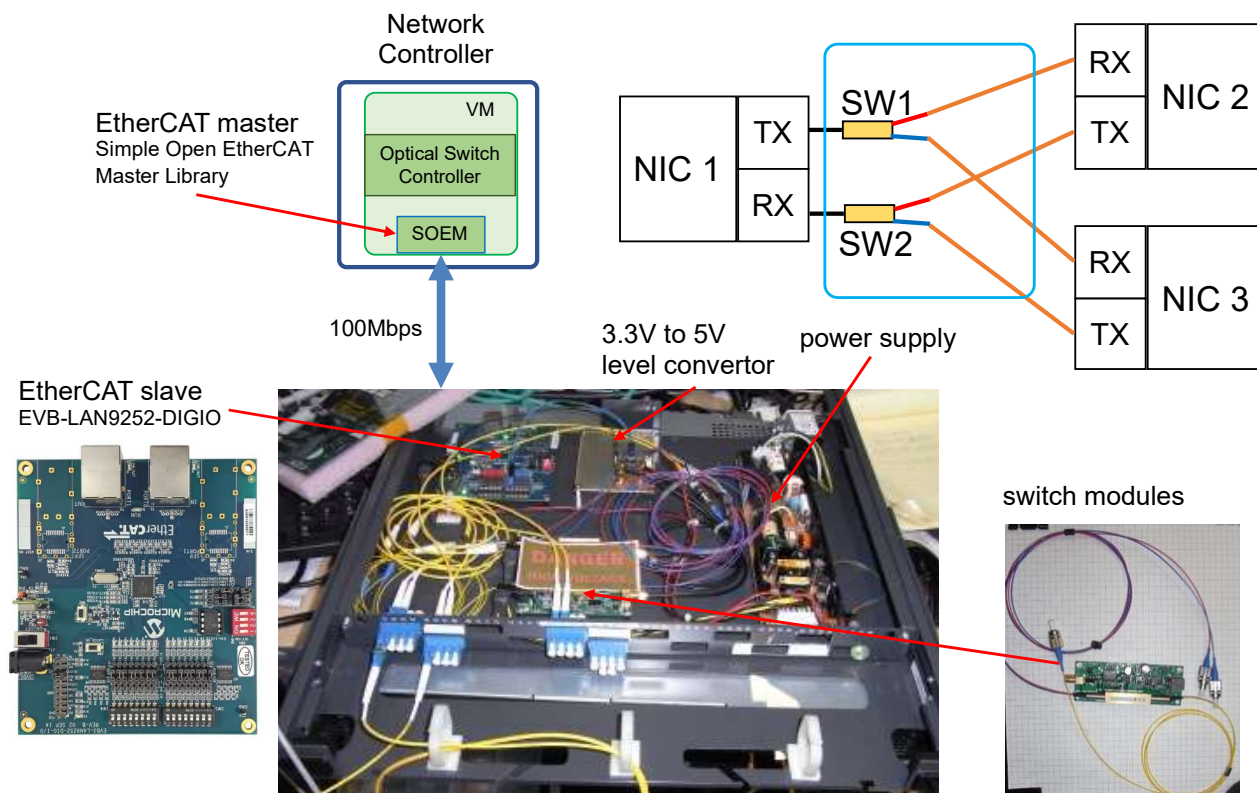


図 1.2.3-8 ソフトウェアベース EtherCAT マスタを用いた実験構成

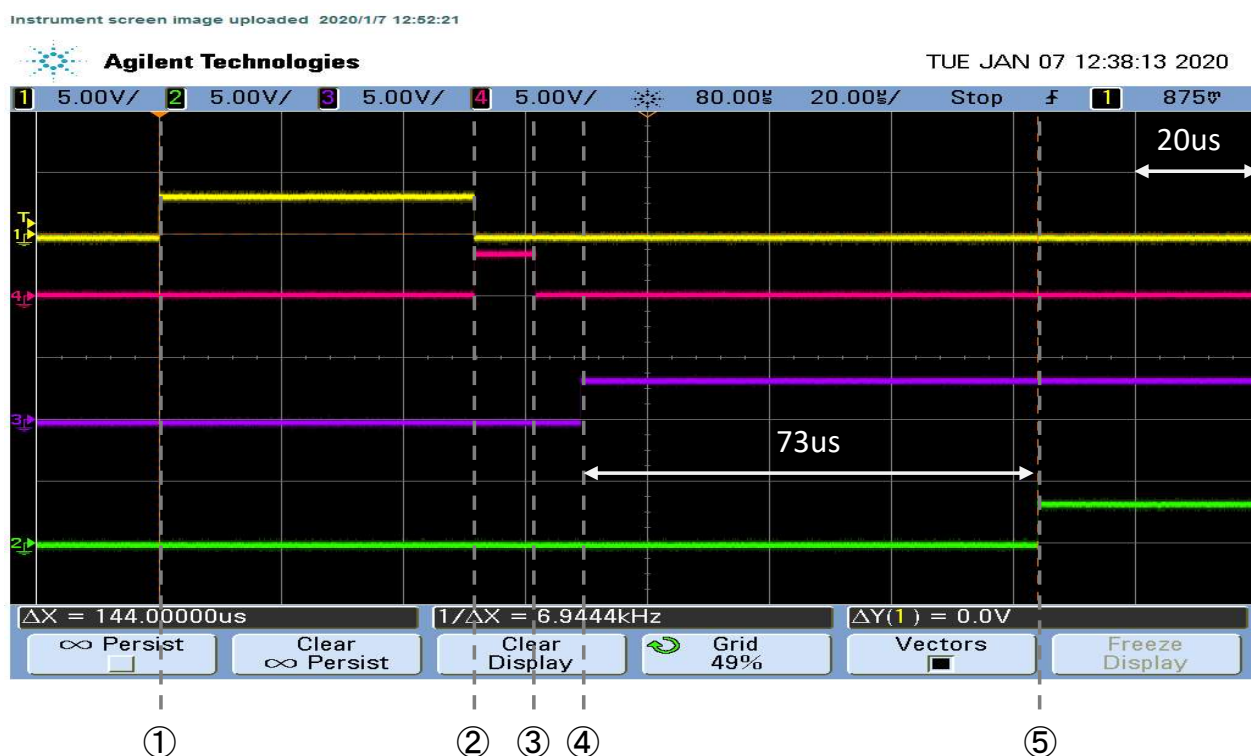


図 1.2.3-9 ソフトウェアベース EtherCAT マスタ使用時の光スイッチ制御タイミングチャート例

① $t=0$ マイクロ秒: マスタが要求を受信、② $t=52$ マイクロ秒: マスタが共有メモリへの書き込みを開始、③ $t=61$ マイクロ秒: 書き込みの完了、④ $t=71$ マイクロ秒: PDO 周期の開始、⑤ $t=144$ マイクロ秒: 光スイッチの設定完了。

＜MEOW (Multimaster EtherCAT-based control plane for Optical netWorks)の開発＞

前節での検証を踏まえて、SoC FPGA を用いた光電ハイブリッドスイッチネットワーク向けコントロールプレーンである MEOW (Multimaster EtherCAT-based control plane for Optical netWorks)を提案する。MEOWは図 1.2.3-10 に示すように、ネットワークコントローラ、デバイスコントローラ、デバイスの 3 種類のコンポーネントから構成される。MEOW ネットワークコントローラはシステムに一つ存在し、資源管理、API サーバとして動作する。ネットワークコントローラは Northbound API として、アプリケーション・サーバ向けに REST API を提供し、Southbound API として、スイッチ等ネットワーク機器向けのインタフェースを提供する。Southbound API は電気コアスイッチと光コアスイッチごとにインタフェースが異なり、前者は OpenFlow、後者は EtherCAT を想定した独自 API を実装する。MEOW デバイスコントローラは、図 1.2.3-11 に示すように複数の EtherCAT マスタを管理する。EtherCAT スレーブ機能を有する光コアスイッチを MEOW デバイスと呼ぶ。本研究開発項目にて、動作検証のため開発した MEOW デバイスコントローラと 4x4 光ファイバスイッチとして機能する MEOW デバイスの写真を図 1.2.3-12 に示す。

MEOW を ONOS (Open Network Operating System) [8]上に実装した。ONOS はソフトウェア定義ネットワーク (SDN) コントローラプラットフォームのオープンソースソフトウェア実装であり、拡張可能性が高いモジュール構成を採用している。また、キャリアグレードの採用実績もあるなど実用性も高い。

さらにデバイスコントローラを Xilinx ZCU106 評価ボード上に実装した。ZCU106 評価ボードには Zynq UltraScale+マイクロプロセッサ SoC、DDR4 メモリ、FMC 拡張ポート、高速シリアルトランシーバ等が搭載されている。Zynq は Arm Cortex マイクロプロセッサとプログラマブルロジック (PL)が単一チップ上に実装されている。マルチマスタ EtherCAT 機能は PL 領域に実装した。当領域は 100 MHz で駆動し、単一チップ上に 6 つのマスタを実装している。現在の実装では、マスタ間の PDO 同期の実装していないが、原理的には同期は可能である。Petalinux 2018.3 が Arm Cortex マイクロプロセッサ上で動作しており、MEOW デバイスコントローラはリアルタイム性を保証するためにユーザ空間プロセスとカーネル空間リアルタイムタスクの混成として実装している。ネットワークコントローラからの要求の受付処理はユーザ空間プロセスで実現し、PDO 周期での送受信処理はカーネル空間タスクが担っている。

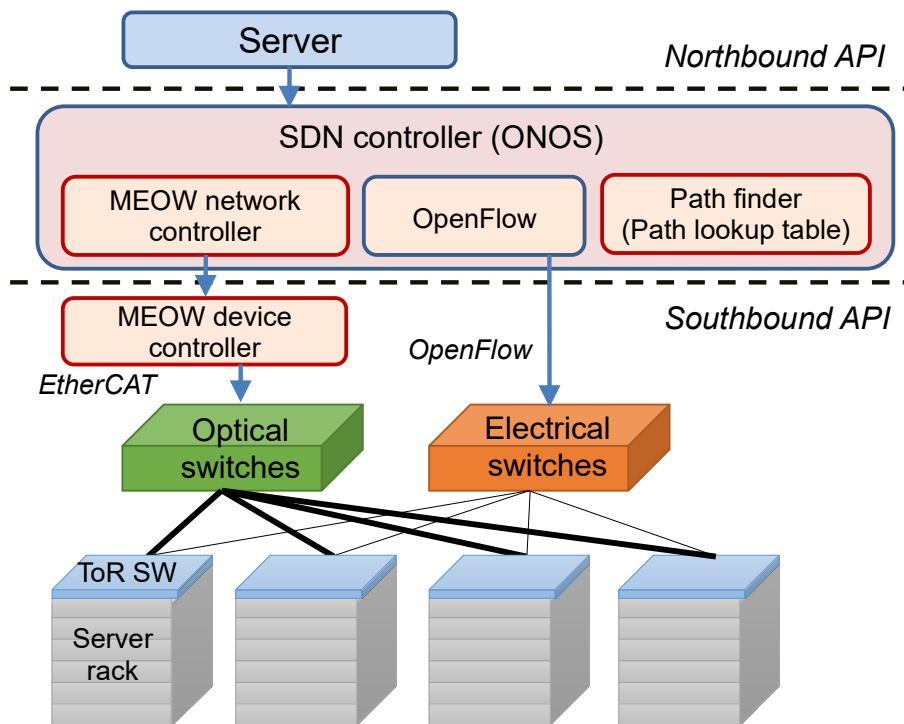


図 1.2.3-10 MEOW の概要

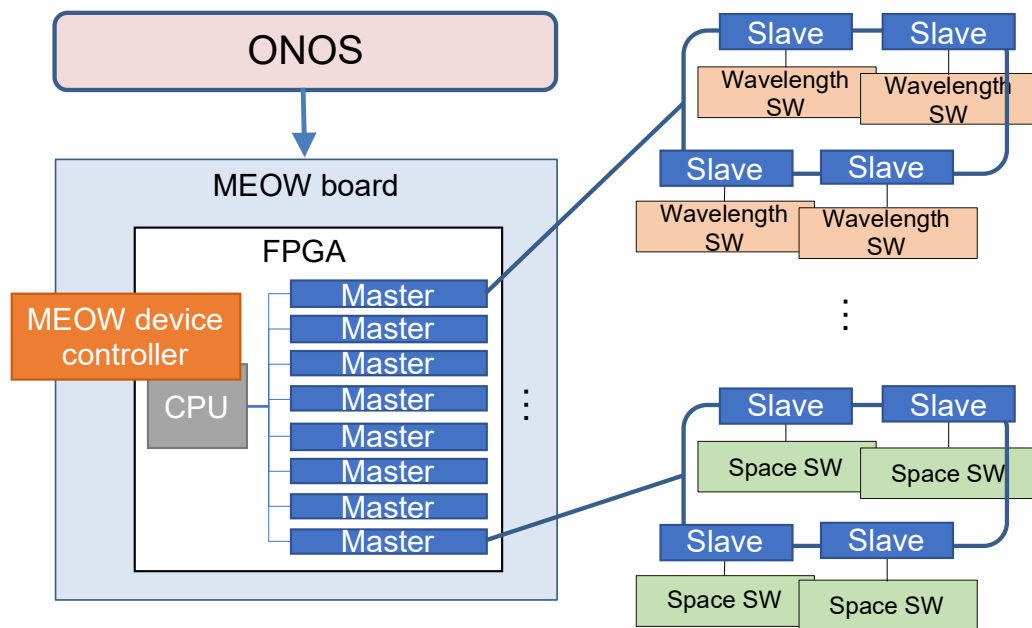


図 1.2.3-11 MEOW デバイスコントローラの概要

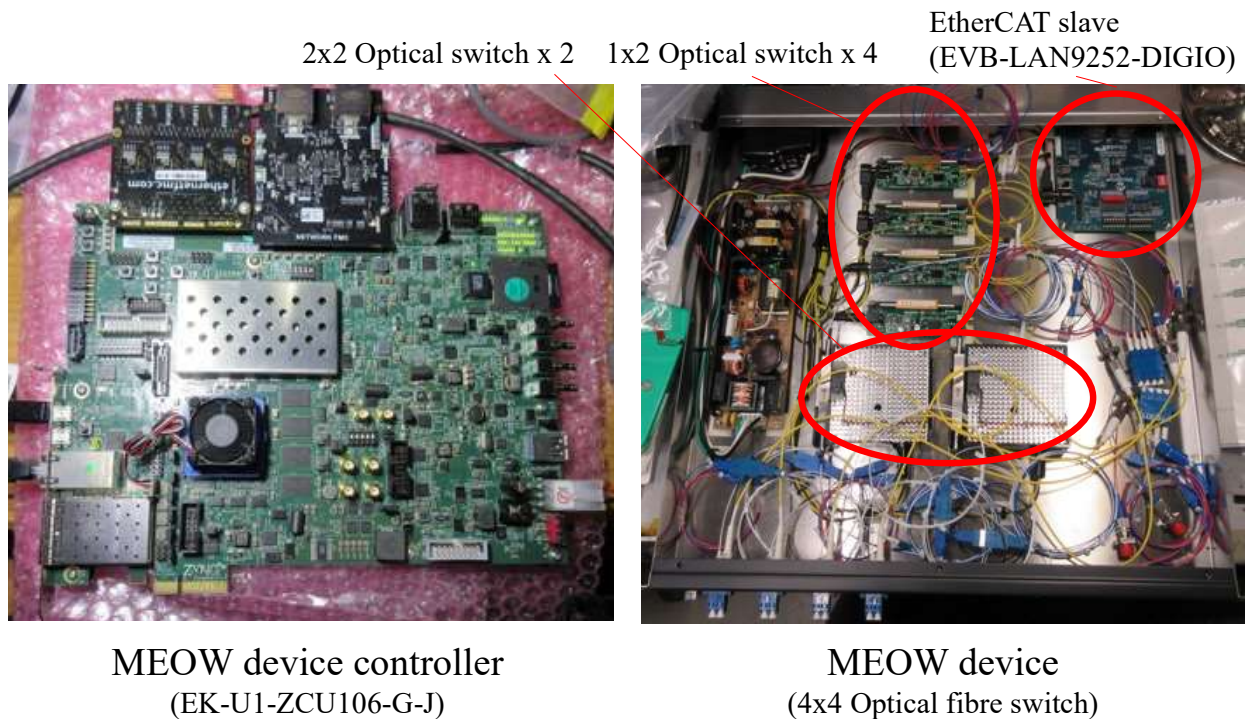


図 1.2.3-12 MEOW デバイスコントローラと MEOW デバイスの実装例

＜光部品表記法の開発＞

本システムでは、光ネットワークの構成管理や経路探索を体系的に実現するために、多様な光部品の機能を統一的かつ自動処理可能な形式で記述する FBD (Functional Block based Disaggregation) モデル[9]を採用する。FBD モデルを用いることで、光ネットワークを機能ブロック (AWG や FXC などのコンポーネント) 単位でモデル化し、ネットワークコントローラやアプリケーションからの要求光パスに対し各部品の必要設定を自動計算できる。具体的には、次のように光ネットワークの設計から経路計算までを行う。なお、FBD モデルを用いたトポロジ情報生成ツールおよびサンプルライブラリについて、CPPC ホームページ[10]で公開している。

- (1) オープンソース CAD (KiCAD) を用いた GUI 操作 (シンボルライブラリを使用) により、任意の光コンポーネントのライブラリを生成する。コンポーネント属性として、光切替機能を GNU MathProg モデリング言語を用いた機械可読な整数線形計画問題として記述する。これにより任意の二点間の経路を自動で計算することが可能となる。
- (2) オープンソース CAD を用いた GUI 操作 (回路図エディタを使用) により、(1) で生成した光コンポーネントを任意に組み合わせた光ネットワークを設計する。開発したプラグインプログラムにより、設計した光ネットワークのトポロジ詳細情報を XML 形式で出力する。
- (3) (1) で生成したトポロジ情報を自動解析するプログラムにより、全トランシーバペア間の経路計算を行うことで、ネットワークコントローラが利用するルックアップテーブルを生成する。

本プロジェクトで開発する光電ハイブリッドスイッチは論理的に対称な構成であるため、全要素を一度にインストールし、物理的にも対称な構成で使用する場合には、任意の経路を計算可能な FBD

モデルのフレームワークを使用せずに、簡易なアルゴリズム（対称性を前提として for ループで制御要素番号を回すなど）により、経路設定情報を生成することも可能である。ただし、そのような簡易なアルゴリズムを利用すると、段階的なインストールに対応できない。さらに、図 1.2.3-13 に示すような論理的に同一でも物理的に異なる接続を考慮することができない。図 1.2.3-13 に示す(a)と(b)は論理的には同様の機能を提供するが、途中の光スイッチの接続が異なる。そのため、C1_P1 から C6_P4 までの光パスを同様に設定しようとしても、(a)と(b)とでは設定すべき光スイッチポート接続が異なる。このような非対称な接続は、例えば、光デバイスのポート毎の物理特性に応じて最適な組み合わせで光ファイバコードを接続する場合に必要となる。FBD モデルを用いることで、これらの非対称なインストール状況に一律に対応可能となる。

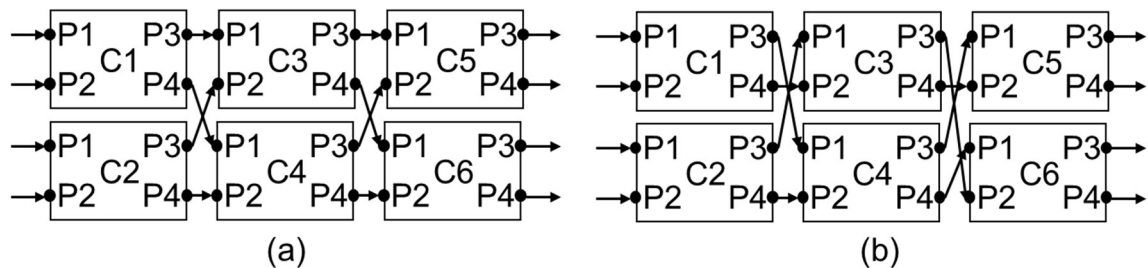
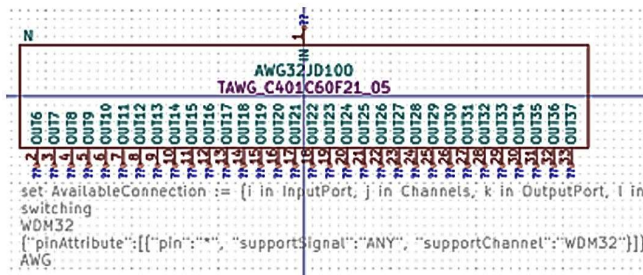
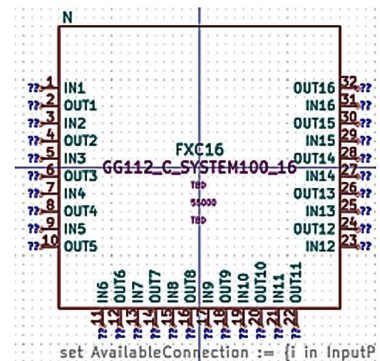


図 1.2.3-13 光スイッチの論理接続と物理接続の関係



AWGの例



FXCの例

図 1.2.3-14 KiCAD で記述した光コンポーネント例

生成した JSON フォーマットのルックアップテーブルの一例（抜粋）を図 1.2.3-15 に示す。ルックアップテーブルでは、始点と終点のトランシーバポートペア（図中“src”と“dst”）に対し、設定すべき光スイッチと波長の設定内容がリストで記載される。具体的には、“set-switch”において、光スイッチ機器を示す“id”と、各スイッチ機器で設定すべき入出力ポートペアのリストが記載される。また、“set-lambda”において、設定すべき波長制御機器（波長可変光送信器/受信機、チューナブル波長フィルタなど）を示す“id”、および、設定波長を示す“lambda”、さらに、一つの機器で複数のポートを持つ場合に対応するためポートを指定する項目（図では“inport”）を含むリストが記載される。なお、図の例では波長可変受信機を想定した光電ハイブリッドスイッチアーキテクチャの例が記載されているが、開発した光部品表記法は、波長可変光送信器、あるいは、チューナブル波長フィルタを用いたアーキテクチャへも問題なく適用可能である。通常、データセンタ内で使用される Ethernet は双方向

通信であり、本プロジェクトで使用を想定する光トランシーバは2芯双方向通信が基本である。送信端からの光パスリクエストに対し、ペアとなるパスを自動で設定するために、始点終点のトランシーバポートペアに対し、ペアとなるパス設定が“pair-config-set”として記載されている。

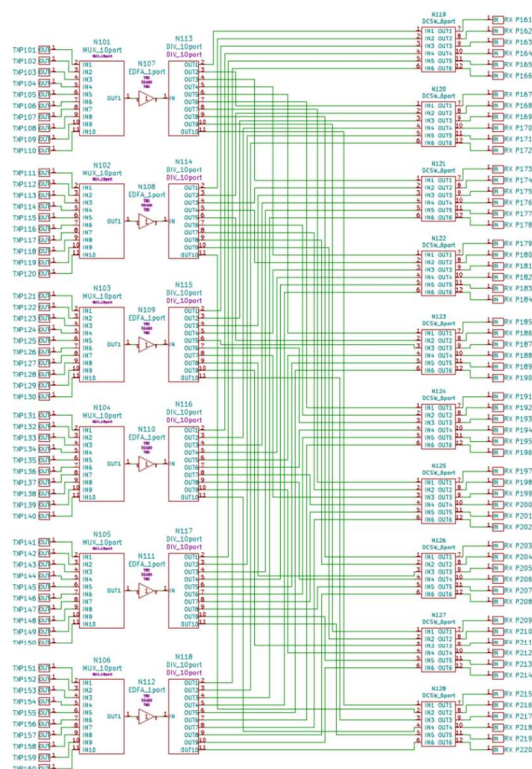
ルックアップテーブルでは、光パスの設定のための情報に加え、EtherCAT のコントロールプレーンネットワークの構成情報も記載される。これにより、光電ハイブリッド光スイッチ制御のための必要な情報がすべて一元管理できることとなる。“master-info”に EtherCAT マスタの情報がリストで記載される。リスト表記の採用により、マルチマスタの情報を不足なく記載することが可能である。“control-info”ではスレーブ機器がリストで記載され、マスタ機器との対応を取るための情報も含まれる。

```
21 {
22   "src": "P101_1",
23   "dst": "P106_1",
24   "set-switch": [
25     {
26       "id": "N104",
27       "inport": "1",
28       "outport": "17"
29     }
30   ],
31   "set-lambda": [
32     {
33       "id": "P106",
34       "inport": "1",
35       "lambda": "WDM96_1"
36     }
37   ],
38   "pair-config-set": {
39     "src": "P102_1",
40     "dst": "P105_1",
41     "set-switch": [
42       {
43         "id": "N103",
44         "inport": "1",
45         "outport": "17"
46       }
47     ],
48     "set-lambda": [
49       {
50         "id": "P105",
51         "inport": "1",
52         "lambda": "WDM96_2"
53       }
54     ]
55   }
56 }

508 "master-info": [
509   {
510     "MasterID": 0,
511     "IP": "192.168.0.10",
512     "PORT": 54890
513   },
514   {
515     "MasterID": 1,
516     "IP": "192.168.60.0.11",
517     "PORT": 54891
518   }
519 ],
520 "control-info": [
521   {
522     "N103": {
523       "MasterID": 0,
524       "SubMasterID": 0,
525       "SlaveID": 0
526     }
527   },
528   {
529     "N104": {
530       "MasterID": 0,
531       "SubMasterID": 0,
532       "SlaveID": 1
533     }
534   },
535   {
536     "P105": {
537       "MasterID": 0,
538       "SubMasterID": 0,
539       "SlaveID": 4
540     }
541   }
542 ]
```

図 1.2.3-15 自動生成した JSON フォーマットのルックアップテーブルの一例（抜粋）。

左：スイッチ設定記述部。右：EtherCAT コントロールプレーン情報記述部。



例) 60x60スイッチ

```

1  "config-set": [
2
3  {
4    "src": "P101_1",
5    "dst": "P131_1",
6    "lambda": "WDM96_1",
7    "set-switch": [
8      {
9        "id": "N110",
10       "input": "1",
11       "output": "4"
12     }
13   ],
14 },
15 {
16   "src": "P101_1",
17   "dst": "P132_1",
18   "lambda": "WDM96_1",
19   "set-switch": [
20     {
21       "id": "N110",
22       "input": "1",
23       "output": "5"
24     }
25   ],
26 },
27 {
28   "src": "P101_1",
29   "dst": "P133_1",
30   "lambda": "WDM96_1",
31   "set-switch": [
32     {
33       "id": "N110",
34       "input": "1",
35       "output": "6"
36     }
37   ],
38 },
39 ]

```

JSONフォーマットの
ルックアップテーブル

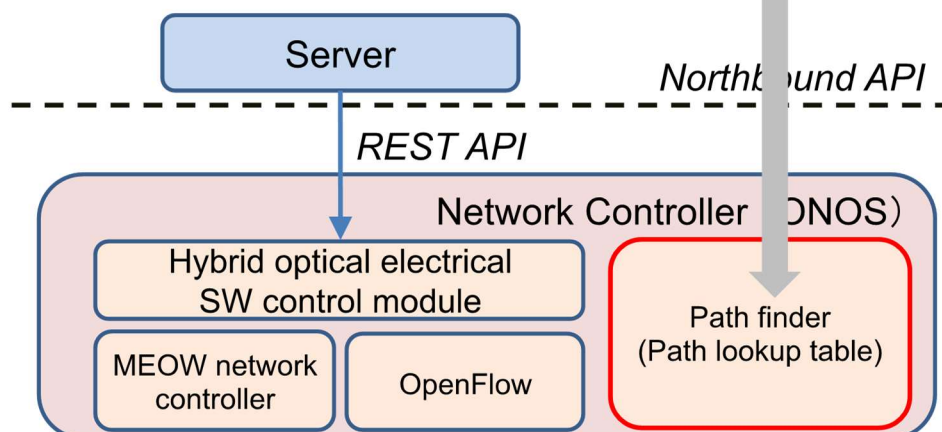


図 1.2.3-16 光部品表記法を用いた経路制御の流れ

＜MPI 通信ライブラリ支援機能の開発＞

前述した分散深層学習をはじめとして高性能計算アプリケーションなど、大規模な並列処理プログラムでは MPI (Message Passing Interface) 通信ライブラリがデファクトとして広く使われている。サービス駆動方式において、MPI を用いて記述されたプログラムの実行に合わせて、ONOS を経由して光パスを確立するための仕組みを実装した。MPI ライブラリとしてトラフィック振り分けアルゴリズムを実装することで、アプリケーション自体を変更することなく、光電ハイブリッドスイッチネットワークを活用できる。

実装の概要を図 1.2.3-17 に示す。実行の流れとしては以下の通りである。

- (1) 通信に参加する計算サーバのリスト (ホストリスト) と光パスのトポロジファイルを入力として、`mpirun` ラッパースクリプトを実行すると、`request.json` が生成される。なお、

mpirun とは MPI プログラムのランチャープログラムである。

- (2) mpirun ラッパースクリプトは ONOS に request.json を POST して光パスを確立する。
- (3) mpirun ラッパースクリプトは mpirun を実行し、MPI プログラムを起動する。

電気ネットワークと光ネットワークへの経路の切替は ToR スイッチ（OpenFlow スイッチ）で実現する。経路の切替は、パケットに付与されたヒント情報（タグ）に基づいて行うが、ここでは ToS (Type of Service) 値を用いることとする。振り分けの OpenFlow ルールはあらかじめ ToR スイッチに登録しておく。MEOW ネットワークコントローラは、経路と ToS 値の対応を管理しており、光パス確立の要求に合わせて、計算サーバに払い出されて適切に設定される。ToS 値の変更には Linux OS の iptables コマンドを利用する。

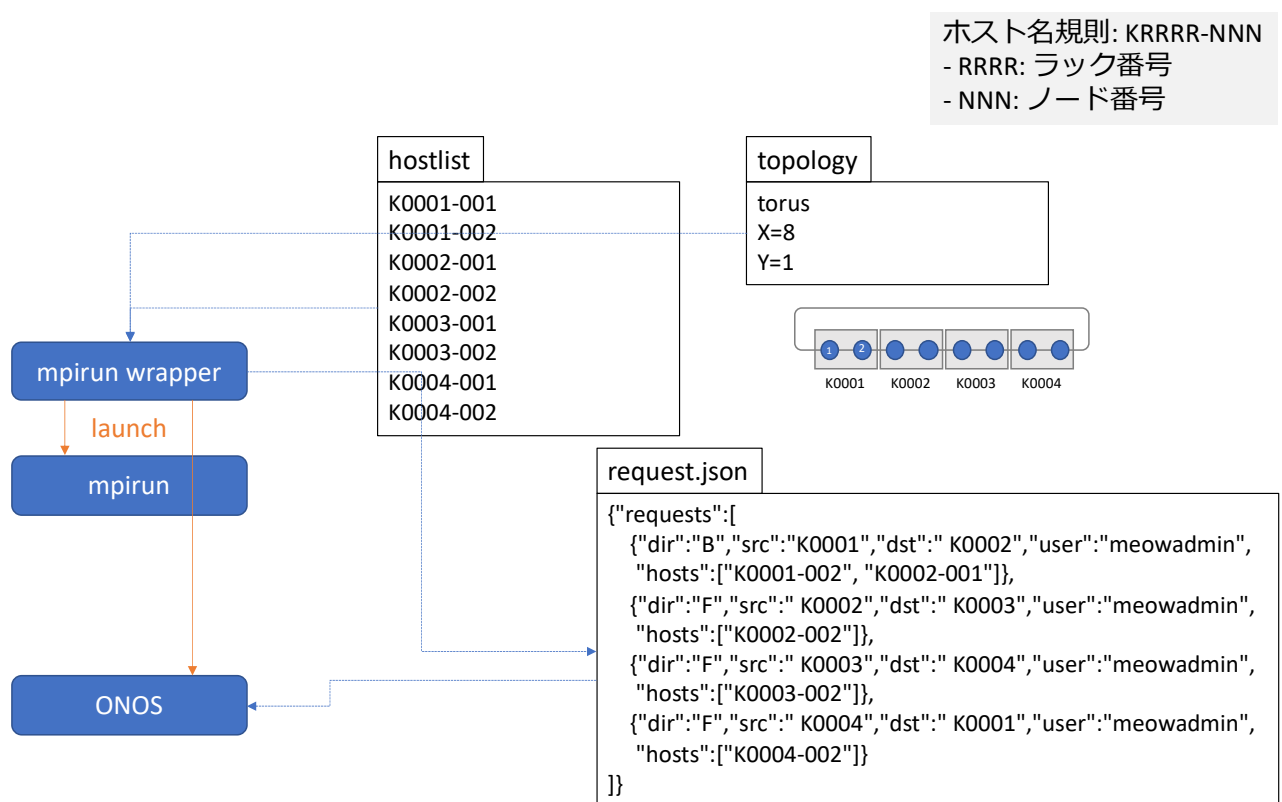


図 1.2.3-17 MEOW と MPI ランタイムとの連携動作

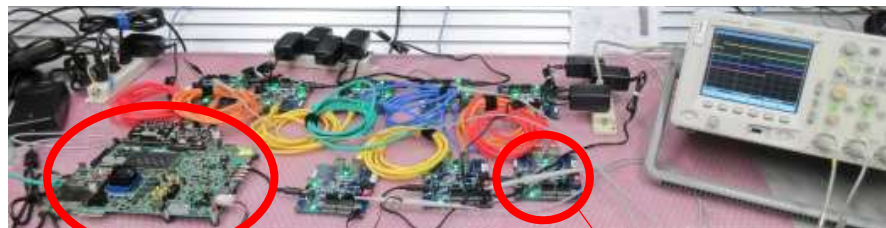
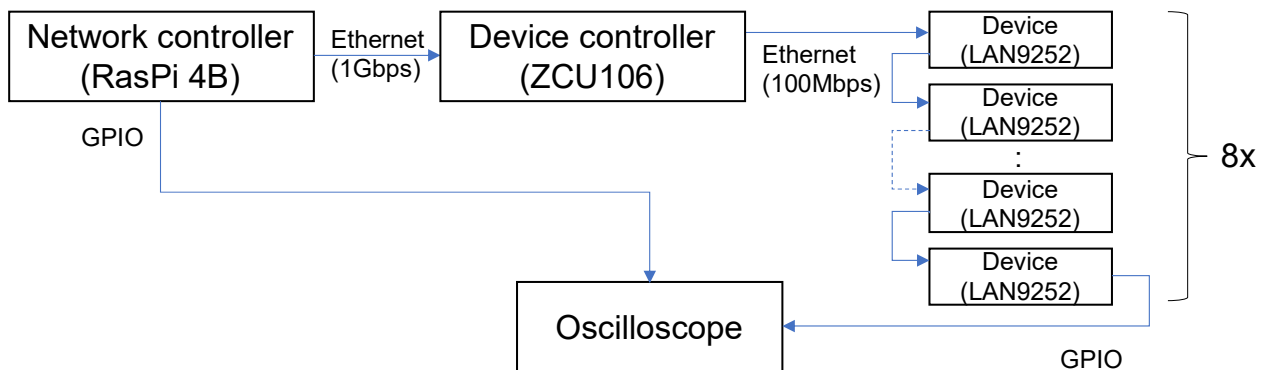
<システム検証実験（シングルマスタ）>

MEOW の実現可能性を検証するために、図 1.2.3-18 に示すように、1 つのデバイスコントローラと 8 つのデバイスから構成される小規模な実験環境を用意した。MEOW デバイスとしては、<EtherCAT を用いた光スイッチ制御>にて示した予備実験と同じく EVB-LAN9252-DIGIO 評価ボードを用いた。MEOW デバイスコントローラと 8 つのデバイスは、Opsero OP031-1V8 EtherFMC カードを経由してカスケード接続した。

実験では、ネットワークコントローラによるパス確立リクエスト送出から、すべてのデバイスがリクエストの処理を完了するまでの経過時間を測定した。接続末尾の EVB-LAN9252-DIGIO 評価ボードのデジタル I/O ピンはオシロスコープに入力されており、この信号のアサーションを観測することで操作完了とみなす。なお、本実験では EtherCAT スレーブに光スイッチが接続されておらず、光スイ

ッチ自体の切替時間は結果に含まれない。また、予備実験ではネットワークコントローラと EtherCAT マスタ間の通信時間を含まなかったが、本実験では含むことに注意されたい。

ネットワークコントローラとして、簡単化と測定の容易さのため、PC サーバ上の ONOS の代わりに、Raspberry Pi 4 Model B（以下、RasPi と呼ぶ）上に実装した簡易コントローラを用いた。コントローラからリクエスト送出するタイミングで、RasPi の GPIO ピンの出力をオンにし、オシロスコープでこれをトリガとすることでリクエストの処理経過時間を測定した。なお、RasPi の CPU クロックは 1.5 GHz に固定している。また、RasPi と MEOW ボードは NETGEAR GS324（仕様上のポート間遅延の 4.7 マイクロ秒）を介して接続した。



MEOW device controller
EK-U1-ZCU106-G-J



MEOW device
EVB-LAN9252-DIGIO



図 1.2.3-18 シングルマスタ実験構成

1000 リクエストの処理を観測したときのタイミングチャートを図 1.2.3-19 に示す。前述したように、リクエスト処理完了時間はデバイスがリクエストを受信したタイミングによってばらつく。PDO 周期の直前でリクエストを受信した場合は、待ち時間なくテレグラムを送出できる。一方、PDO 周期の直後にリクエストを受信した場合は、次の PDO 周期が来るまで、つまり今回の設定では最悪 30 マイクロ秒、テレグラムの送出手間を待つ必要がある。処理時間の最小値は 70.0 マイクロ秒 (③) であり、最大値は 103.4 マイクロ秒 (③+⑥) であった。最悪ケースでは目標値をやや超過するものの概ね許容範囲内であると考えられる。

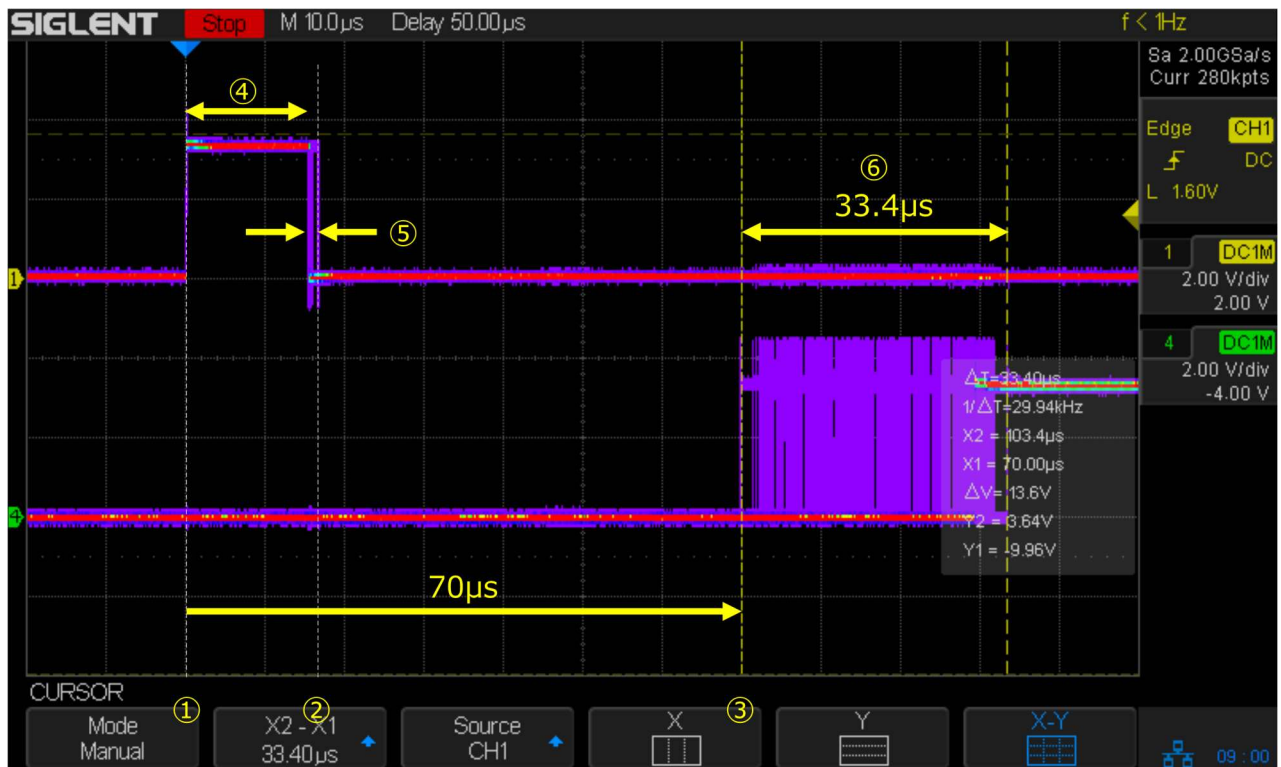


図 1.2.3-19 タイミングチャート（横軸の目盛線は 10 マイクロ秒）。

- ① 簡易ネットワークコントローラがリクエストを開始、② リクエストの送信が完了、
- ③ すべてのスレーブ処理が完了、④ 送信処理に要した時間、⑤ コンテキストスイッチに
関係するジッタ時間、⑥ POD サイクル待ちに起因するジッタ時間。

< システム検証実験（マルチマスタ） >

前節に続き、さらに今後の展開を見据えて、複数のマスタを並列で動作させた場合の動作確認を行った。図 1.2.3-20 に示すように、8 つの MEOW デバイスを 2 つずつネットワークセグメントに配置することで、4 つのマスタ機能を並列で動作させた。

結果を図 1.2.3-21 に示す。シングルマスタでは PDO サイクルが 30 マイクロ秒で動作したが、4 マスタでは安定動作しなかったため、PDO サイクルを 80 マイクロ秒に設定した。この影響もあり処理時間の最小値は 100 マイクロ秒になるものの、最大値は 187 マイクロ秒となった。問題の分析と解析は今後の課題とする。

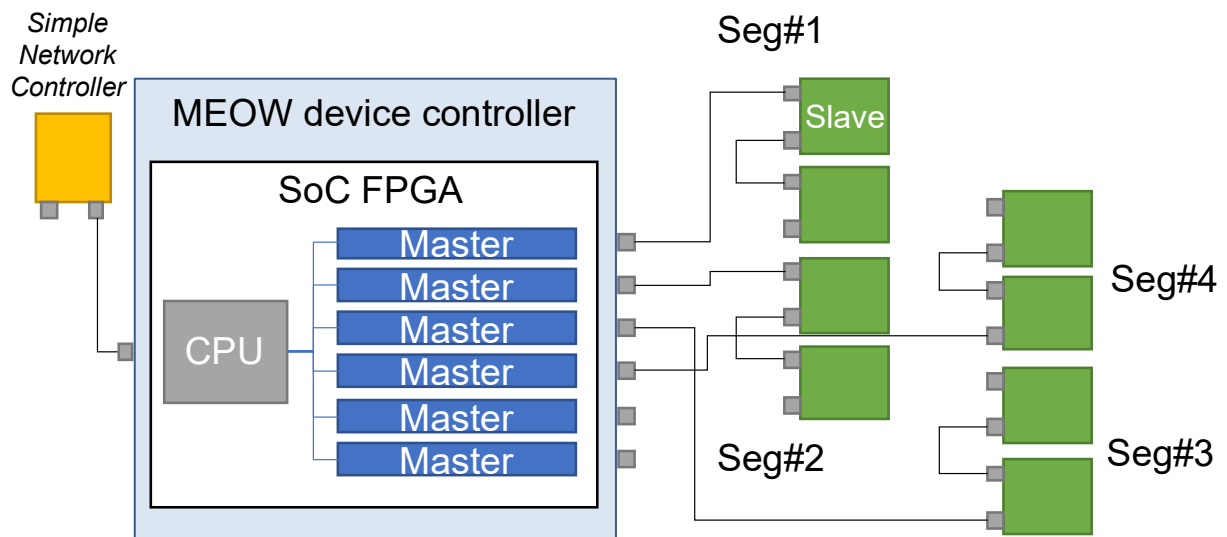


図 1.2.3-20 マルチマスタ実験構成

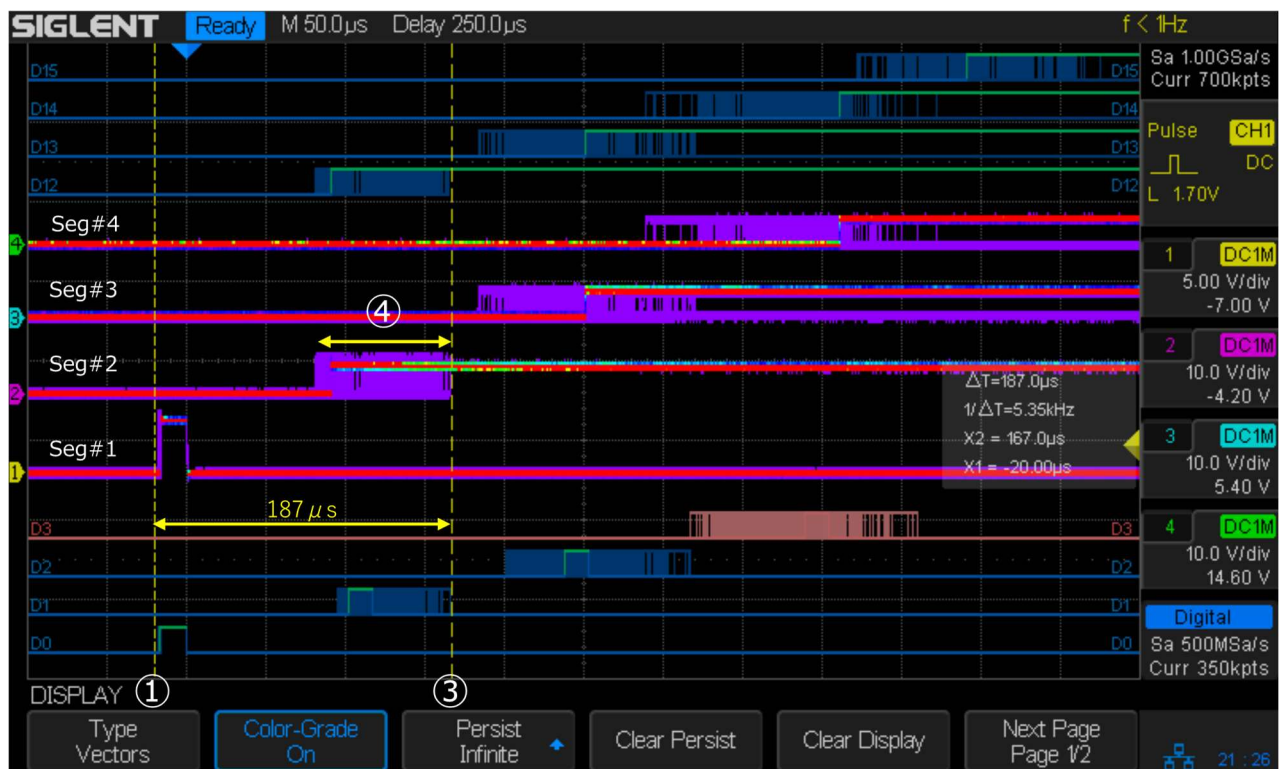


図 1.2.3-21 4マスタでのタイミングチャート（PDO サイクル 80 マイクロ秒）

- ① 簡易ネットワークコントローラがリクエストを開始、② マスタがスレーブにリクエストの送信を開始、③ すべてのスレーブ処理が完了、④ POD サイクル待ちに起因するジッタ時間。

＜MPI プログラムを用いたシステム動作検証＞

図 1.2.3-22 に示すような、4 ラックの光電ハイブリッドスイッチネットワークを模した環境を構築した。各ラックには計算サーバが 1 台ずつ存在する。光スイッチは電気スイッチを用いて模擬している。本実験では MEOw コントロールプレーンを用いて、期待する動作を確認できた。

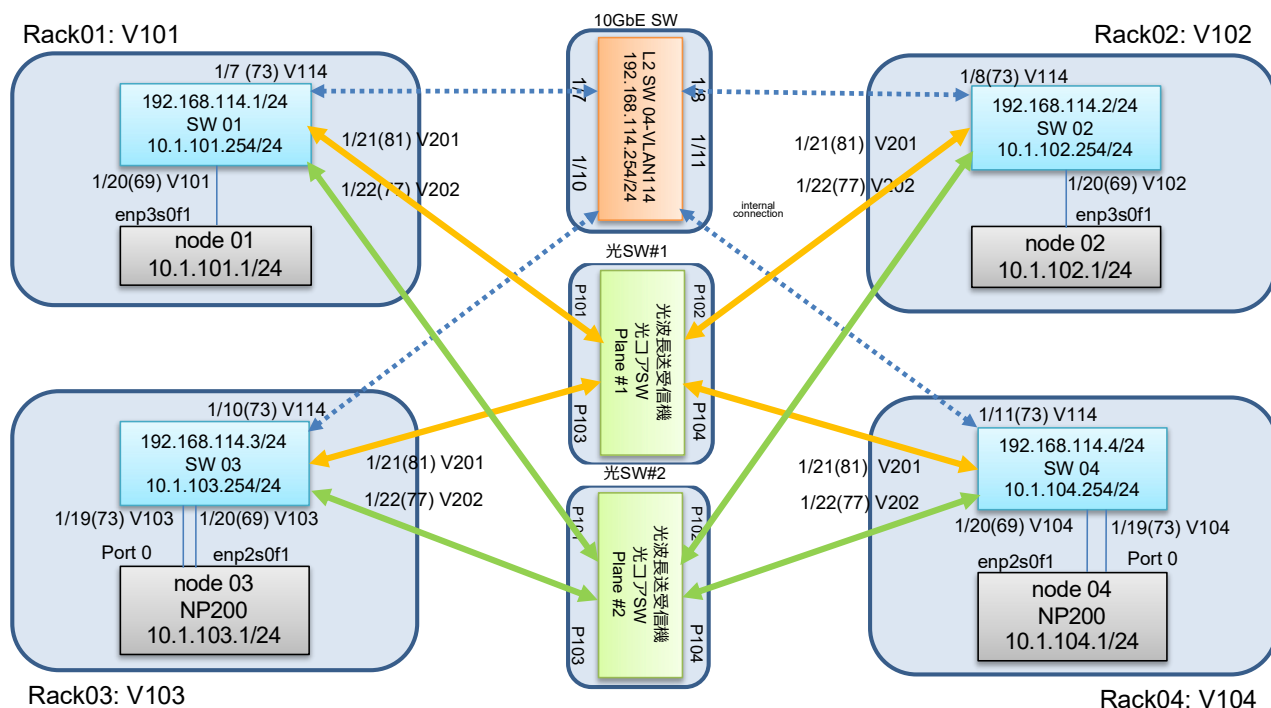


図 1.2.3-22 MPI AllReduce の検証環境 (1 ノード×4 ラックの実験環境)

さらにスイッチ切替時の挙動をマイクロに確認するため、スイッチでのポートミラーリングと FPGA スマート NIC (Napatech NT200) によるパケットキャプチャを用いて、パケット間隔の変化を観測した。iperf を用いて 4Gbps の UDP トラフィックを生成し、MEOW ネットワークコントローラ経由で ToS 値を切り替えた。結果を図 1.2.3-24 に示す。横軸がパケットに付与したシーケンス番号であり、縦軸がパケット間の間隔である。ToS 値が 0 のときは電気スイッチ、4 のときは (模擬) 光スイッチを経由している。経路切替時にパケット損失は観測されなかったが、1 ミリ秒強の受信遅延が発生していることがわかる。この原因は Linux OS の iptables コマンド操作による ToS 値設定に関係していると推測されるが、詳細な調査は今後の課題である。

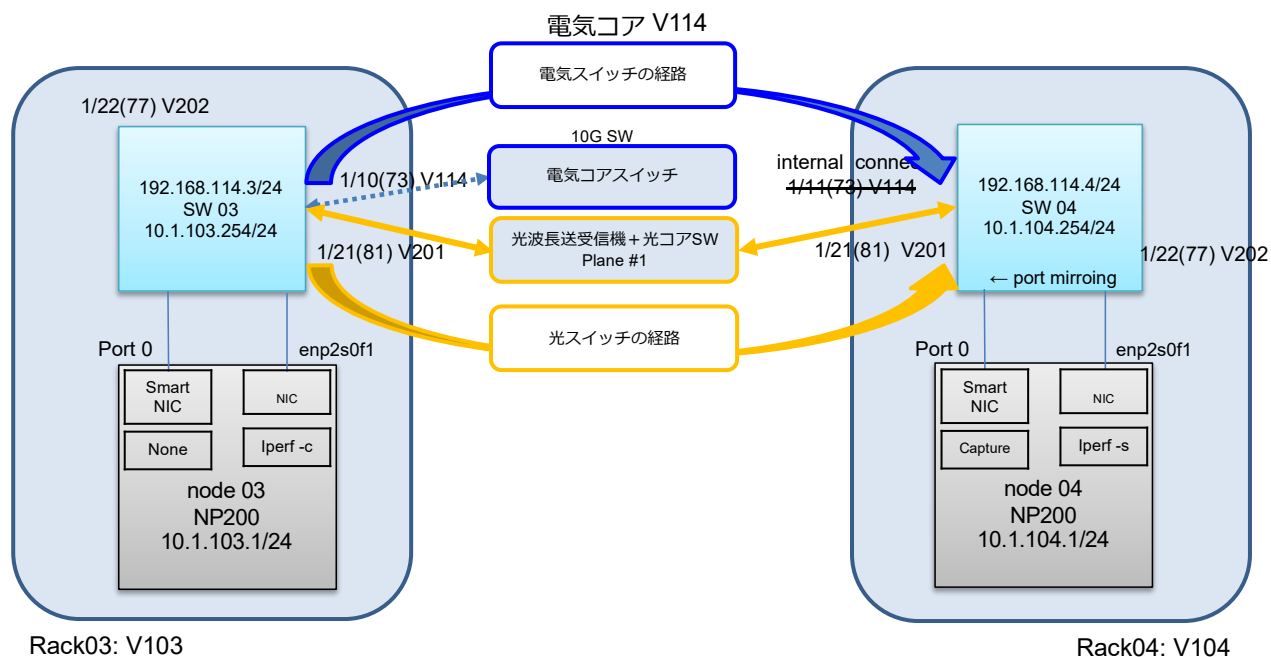


図 1.2.3-23 パケット測定実験環境

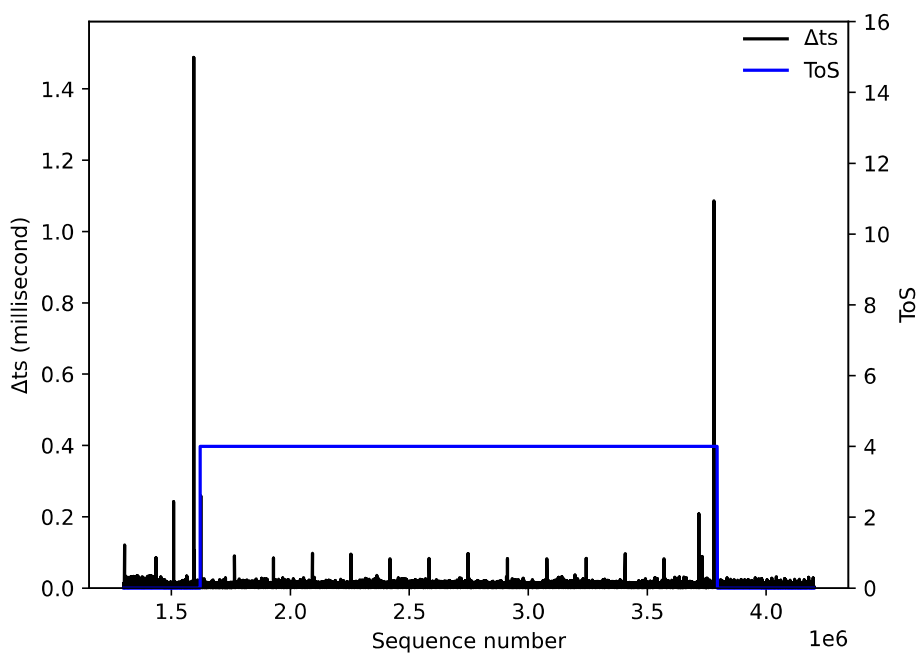


図 1.2.3-24 パケット間隔の観測

さらに ONOS 内部でのリクエスト処理時間の内訳を表 1.2.3-2 に示す。ONOS は Java で実装されており、メモリ管理にガベージコレクション (GC) を採用しているため、初期化時のクラスロードや GC の影響による性能劣化を考慮する必要がある。今回の実験では、初回の実行時間を除いた 10 回試行した場合の平均値を測定した。MEOW デバイスマネージャとの通信と計算サーバの ToS 値の設定は同時並行に実行できるため、リクエスト処理時間はこれらの総和になるわけではないが、数百マイクロ秒かかっており、さらなる高速化の必要がある。

表 1.2.3-2 ONOS リクエスト処理時間の内訳

処理内容	実行時間 (マイクロ秒)
REST-IF から要求された JSON 形式のリクエストから、光パスの設定に必要な情報（波長、ポート番号、パスの向き等）の取出し	133.2
光スイッチのポートの排他制御	13.1
複数のパスを一括りにして扱うグループの割当	4.7
リクエストメッセージの組立て	33.7
MEOW デバイスマネージャへのメッセージの送受信	164.8
ホストリストに登録されたサーバに対する ToS 値の設定	308.8

<まとめ>

本研究開発項目では、光電ハイブリッドスイッチネットワークの特性に適合したトラフィック振り分けアルゴリズムを提案し、分散深層学習での効果を確認した。さらにトラフィック振り分けアルゴリズムの効果を発揮するためにも高速なコントロールプレーン技術が必要である。そこで、1000 ラック規模まで拡張可能であり、100 マイクロ秒で光スイッチ制御が可能な高速コントロールプレーン技術 MEOW を提案し、動作検証を完了した。具体的には、リアルタイム Ethernet 規格である EtherCAT をベースに、マスタ機能のハードウェア化および多重化を実現し、オープンソースネットワークコントローラである ONOS を拡張することでコントロールプレーンとしてシステム化した。FPGA と汎用ハードウェアを用いて提案システム試作し、約 70 マイクロ秒から約 100 マイクロ秒で光スイッチ制御が実現可能なことを確認した。このように必要要件を、汎用で安価なコモディティハードウェアとオープンソースソフトウェアを用いて実現できることを明らかにし、光電ハイブリッドスイッチシステムの実用化に向けたソフトウェア制御技術確立できた。我々の知る限り、データセンタ規模のネットワーク制御に EtherCAT を用いることは前例がない。

さらに研究開発項目⑤「光電ハイブリッドスイッチシステム実証」と連携して MEOW の技術をシステム実証にて検証した。また、開発したネットワークコントローラを参照実装として検証できるように、オープンソースソフトウェアとして github 上 (<https://github.com/ryousei/meow-rest>) に公開した。