修 士 論 文

機能指向型ネットワークデザインの研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

千装 俊幸

2009 年 3 月

修 士 論 文

機能指向型ネットワークデザインの研究

指導教官　篠田陽一 教授

審査委員主査　篠田陽一 教授
審査委員　丹康雄 教授
審査委員　敷田幹文 准教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

0710047 千装 俊幸

提出年月: 2009 年 2 月

概 要

本報告書は、既設スイッチのリプレースによらない、新しい機能のネットワークへの「機能追加」、装置の大量設置によらないネットワーク全体への「機能提供」、および装置設置場所に拘束されない柔軟な「機能適用」を実現する手法の提案である。新しい機能のネットワークへの導入と機能追加に伴うコスト軽減を両立し、ネットワークの発展に寄与することを目的とする。

目 次

図 目 次

表 目 次

第1章　はじめに

本論文は、既設スイッチのリプレースによらない、新しい機能のネットワークへの「機能追加」、装置の大量設置によらないネットワーク全体への「機能提供」、および装置設置場所に拘束されない柔軟な「機能適用」を実現する手法の提案である

今日のネットワークには、多様化し続ける管理上の要求や、高度化するセキュリティ問題への対応が求められている。そして近年、これらの要求、問題に対応する機能として、種々のプロトコルや機構が提案、開発されている。

これらプロトコルや機構によって実現される「機能」のネットワークへの導入は、既設ネットワーク機器を廃棄し、機能を実装した新しいネットワーク機器を既設ネットワーク機器に置き換えるリプレースや、種々の機能を実現する装置 (MiddleBox 等) の新規導入により行われる。これら機能の装置単位での導入は、機能追加のたびにネットワークに継続的なコストを強いる。また装置の設置場所により機能を提供できる範囲が限定されるため、ネットワーク全体に機能を提供するためには装置を多数設置しなければらない。

本論文では、既設ネットワーク機器のリプレースによらない機能追加手法、装置の大量設置によらないネットワーク全体への機能提供、および装置の設置場所に拘束されない柔軟な機能適用を実現する手法を提案する。新しい機能のネットワークへの柔軟な導入と、機能追加に伴うコスト軽減を両立し、ネットワークの発展に寄与することを目的とする。

本論文では、まず現状のネットワーク (特にエンタープライズネットワーク) を取り巻く環境について俯瞰し、そこに表出している「機能追加の問題」を取り上げる。そして、この問題に対するいくつかの先行研究や他のアプローチについて紹介し、それらでは解決しきれていない点や、不十分な点について批判的検証を行う。その後、本論文で提案する手法について述べ、それにより先行研究や他の手法において解決できていなかった点にどのように取り組むかについて述べる。また提案手法について評価を行い、有効性を検証する。

第2章　エンタープライズネットワークの現状と機能追加の問題

本章では、今日のエンタープライズネットワークを取り巻く環境と、その変化について俯瞰する。そしてその変化によって生じてきたネットワークシステムの高機能化と、その機能追加に伴う問題について触れる。

2.1　速度・性能から機能への変化

エンタープライズネットワークとは、企業等の組織内に構築される Local Area Networkq(LAN) のことである。従来、ネットワークは高帯域化やスループットの向上といった通信速度の性能面に焦点があてられてきた。Ethernet 技術の発展・普及により、今日では Gigabit Ethernet の環境が安価に構築できるまでになってきている。

しかし近年では単なる速度といった「性能」面から、ネットワークがどのような機能を提供できるかという「機能」へ焦点が移ってきている。この変化は、エンタープライズネットワークを利用する利用者である労働者の労働形態の多様化や、企業の生産活動のインフラストラクチャーとしてエンタープライズネットワークの役割が増加してきたことなどが起因となっている。

労働形態の多様化はネットワーク環境の利用者の多様化を意味し、使いやすさに加え、企業はネットワークの運用・管理に情報漏洩対策等を考慮する必要が出てきた。従来の正社員に加え、派遣社員や協力会社の常駐者等、ネットワークにアクセスする主体が多様化し、アクセスできる情報資源に対する権限や、アクセスログの管理が必要となってきた。

またセキュリティ対策においても、従来のような、ウィルス対策ソフトの配布やファイアウォールの設置といった十把一絡な対策だけでは不十分になり、より高度な攻撃手法や攻撃手法の多様化・個人や特定の組織をねらったスピア化への対応が必要となってきた [1][2]。

これらの情報漏洩対策、セキュリティ対策、内部統制強化等の普及は、企業の自由意志や自発的防衛意識による内部からの動機だけでなく、日本版 SOX 法等の社会的な外部圧力もその要因となっている。

また、エンタープライズネットワークは単なる業務効率化のツールとしての役割から、より重要な企業活動の生産手段・販売手段の源泉としての役割を担うようになってきたた。そのため、その使用手段・用途の多様化とともに、安定した稼働や大規模化への対応

も求められるようになった。

これらエンタープライズネットワークに求められる要求事項の多様化は、現在のエンタープライズネットワークに求めらる「機能」の多様化を意味する。

2.2 対応規格・プロトコルの登場と製品化

このようなネットワークの機能の多様化に対応するために、様々な規格やプロトコルが提案・開発され、機能を実装した製品が登場している [3]。ネットワークの多機能化についていくつか例を以下に挙げる。

- ネットワークシステムの不正利用や情報漏洩問題への対策としての各種認証機能やアクセス管理機能
- 特定の組織や個人を標的とするスピア型攻撃に対処に特化した、攻撃・ウィルス対策機能
- 認証技術やウィルス対策技術を盛り込んだ検疫ネットワーク機能
- 送受信データのヘッダ情報だけでなく、送受信データのペイロード部を調査する URL フィルタリングや Deep Packet Inspection(DPI) 等の各種フィルタリング機能
- 通信の機密性・完全性・可用性を実現するための暗号化機能や PKI、VPN 機能

各々の問題や脅威に対して上記に挙げたような機能を実装したハードウェア機器やソフトウェア、複合的なセキュリティソリューションが開発されている。

2.3 ネットワーク・ネットワーク機器への機能追加

ネットワークに対する様々な要求への対応として、種々の機能がネットワークに追加される。これらの機能をネットワークに追加する方法は、ある種の機能については既存のネットワーク機器を、新しい機能を実装した機器にリプレース (置き換え) によって実施される。

ネットワーク機器のファームウェアのアップデートやハードウェアモジュールの追加による機能追加が可能な場合もある。しかしながら、ファームウェアのアップデートには、ネットワーク機器に搭載している物理メモリの容量の限界や、度重なるファームウェアの改版によるファームウエアの複雑化などの問題により、限界がある。またハードウェアの追加による方法も、ネットワーク機器の空きスロット数も有限であり、ファームウェアアップデートと同種の問題による限界がある。

結果、ある程度の軽微な変更以上の変更を伴う機能追加にはネットワーク機器のリプレースによる機能追加が行われることになる。次節においてネットワークに認証機能を追加する際に、既存機器のリプレースが行われる例を示す。

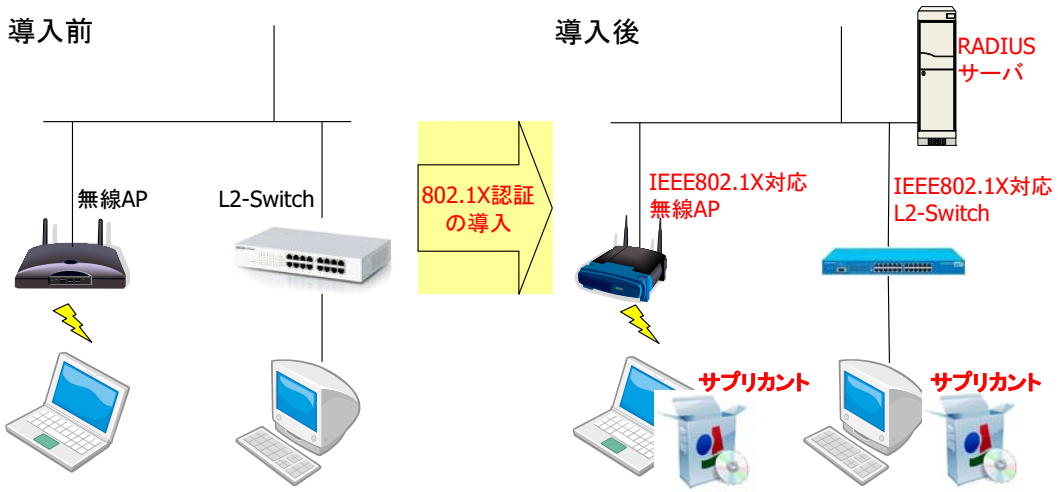


図 2.1: IEEE802.1X 機能導入のための既設機器のリプレース

2.4 既存機器のリプレースによる機能追加の例 -IEEE802.1X の導入-

端末・ユーザを認証しネットワークへの不正な接続を禁止する認証技術として代表的なものに、IEEE802.1X[4] 認証機能がある。IEEE802.1X 認証機能は、RADIUS サーバ、IEEE802.1X 認証対応スイッチ (無線ネットワークの場合は、IEEE802.1X 対応無線アクセスポイント)、およびクライアント端末にインストールするサプリカントから構成される。IEEE802.1X 機能をネットワークに導入するためのリプレース (機器の交換) と追加を図 2.1 に示す。

この IEEE802.1X 機能を導入するには、既設のスイッチや無線アクセスポイントを廃棄して IEEE802.1X 機能対応のスイッチや無線アクセスポイントにリプレースする必要がある。

既設機器のフォワーディング性能 (スイッチング性能やスループット) が十分であったとしても、機能追加のためにリプレースが必要となる。今後もしこのネットワークに検疫機能を追加しようとすると、さらに検疫機能対応の IEEE802.1X 機能導入のために導入したネットワーク機器はさらに検疫機能対応のネットワーク機器にリプレースする必要が生じることになる。

2.5 機能追加に伴うネットワークのコスト増

このようなリプレースによる機能追加は、機能追加の度に、「既設機器の廃棄」と「新規導入機器の購入」というコスト増を生じさせる。機能を追加したいがために、「データを転送する」というフォワーディングプレーンの性能は十分であるにも関わらず、フォワーディングプレーン性能には差異がない新しい機器にリプレースするというのは無駄が多

い。また、費用面のコストだけでなく、リプレース作業はネットワークシステムのダウン時間や機器の再設定作業という機会費用損失や人的コストも生じるさせることになる。

これまでにも、「機能追加のたびに既設機器のリプレースが強いられ、コスト増を招く」という問題を回避する、いくつかのアプローチや先行研究があり、検討・実施されてきた。

- 高スループット、低レイテンシーかつ、様々な機能を内包したHigh-end-Switch[5]をネットワークに導入し、機能追加に備える方法
- 単一機能を担う単機能アプライアンス機器の設置により機能を追加していく方法
- レイヤ2とレイヤ3の間に機能制御を担当する2.5層を設け、MiddleBoxとの組合せにより機能を追加していく手法

次章ではこれら先行研究やアプローチについての分析・検討を行う。

第3章　関連研究および既存手法

2章では、ネットワークへの機能追加と機能追加に伴う既設機器のリプレースの問題について述べた。近年ネットワークに望まれる事項が多様化し、それに対応する技術・プロトコルが開発され、機能を実装した製品が登場していること、そして既設機器を機能を実装した製品にリプレースすることで機能追加が行われており、そのコストが問題になっていることについて触れた。

本章では、「機能追加のたびに既設機器のリプレースが強いられ、コスト増を招く」という問題に対するこれまでの先行研究やアプローチについての分析・検討を行う。「High-end-Switchの導入による手法」、「単機能アプライアンスの導入による手法」、「ネットワーク機器制御層構築による手法」の順に、それぞれの手法について紹介し、内容について検討を行う。

3.1　High-end-Switchの導入による手法

いわゆるHigh-end-Switchといわれるネットワーク機器をネットワークに導入することにより、追加的な機器のリプレースを回避する方法である。

High-end-Switchとは、レイヤ2/レイヤ3スイッチとしての高速なフレーム、パケット処理性能に加え、各種ファイアウォールやロードバランサーなどの様々な機能を内包したネットワーク機器である。ネットワーク機器の高機能化は現在のトレンドでもある。ネットワーク機器に想定される機能を多く搭載しており、用途によって搭載機能の中から選択して使用することで様々なネットワーク形態やネットワークの改編にも対応することができる。

しかしながらHigh-end-Switchと呼ばれるネットワーク機器は非常に高価であり、価格が数百万円するHigh-end-Switchをコア層からディストリビューション層、アクセス層まで全てに配置するのは費用面で大変高い壁となる。

また、これから登場するであろう機能が必ずしもHigh-end-Switchに実装されるとは限らず、実装されたとしても当該機器にもファームウェアアップデートやハードウェアモジュールの追加で機能追加可能となる保証もない。結果不足する機能が生じた場合、リプレースが行われる事態が生じる。

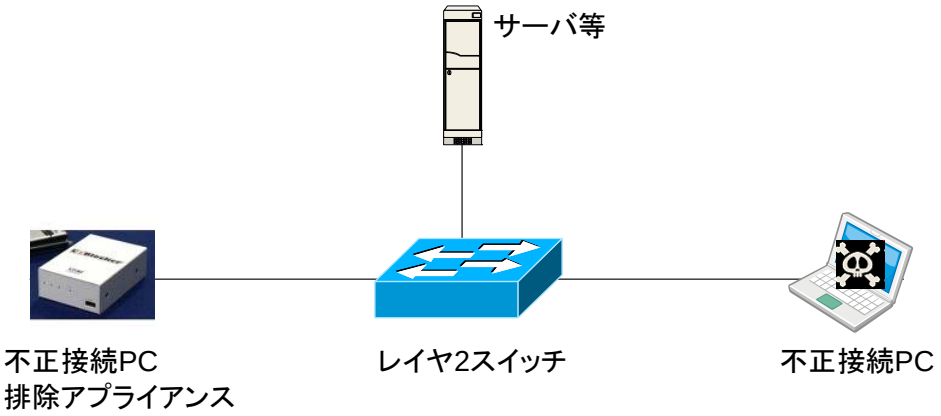


図 3.1: 不正接続 PC 排除システムの設置

3.2 単機能アプライアンス機器の導入による手法

ネットワークに求められる諸問題に対して、それを専門的に解決する単機能機器である単機能アプライアンスと呼ばれる製品が多く提案され、また採用されている。ある問題に対する対処としてある単機能アプライアンス製品を設置し対応にあてるというかたちで、導入・設置し機能を追加することで、ネットワーク機器自体のリプレースを回避し機能追加を計る方法である。本システムが受け入れられやすい理由として、アプライアンス製品ならではの設置のしやすさと、設置目的のわかりやすさがあげられる。

単機能アプライアンス機器の例として、不正接続 PC 排除アプライアンス [6][7] の例を紹介する。

端末･ユーザを認証しネットワークへの不正な接続を禁止には 2.4 節にて前述した IEEE802.1X[4] 認証機能があるが、IEEE802.1X の導入コストに比べ、不正接続 PC 排除アプライアンスの導入は安価であり、設置も当該アプライアンス機器を LAN に接続する 3.1 だけでよいので、これを採用する組織が拡がってきている。

不正接続 PC 排除アプライアンスは不正接続 PC の排除に ARP ポイソニング (Poisoning) いう手法を用いている [8]。この ARP ポイゾニングはスニフィングの手法として知られている。不正 PC 排除システムはネットワークを流れる ARP リクエストパケットを監視し、あらかじめ自身に登録されている正規の PC が持つ MAC アドレス以外からの ARP リクエストパケットを観測するとネットワーク上に存在しない MAC アドレスを埋め込んだ偽の ARP リプライをリクエスト元に送信する。これにより、不正接続 PC の ARP テーブルを汚染し、通信不能状態にすることで不正接続 PC を排除する。不正接続 PC 排除アプライアンスによる ARP ポイゾニングを使用した不正接続 PC 排除の流れを図 3.2 に示す。

しかしながら不正接続 PC 排除アプライアンスが用いている ARP Poisoning による不正接続 PC の排除手法にはいくつかの問題点がある。

- ARP による解法を図っているため、セグメントを越えて端末を管理することができない。

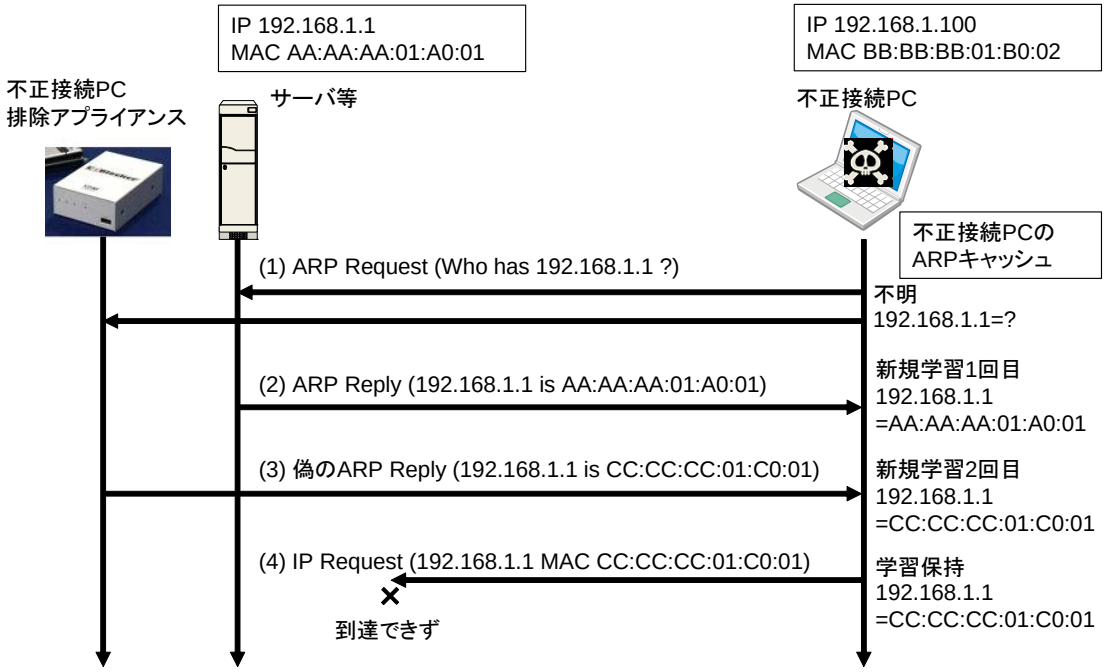


図 3.2: 不正接続 PC 排除システムによる不正接続 PC 排除の様子

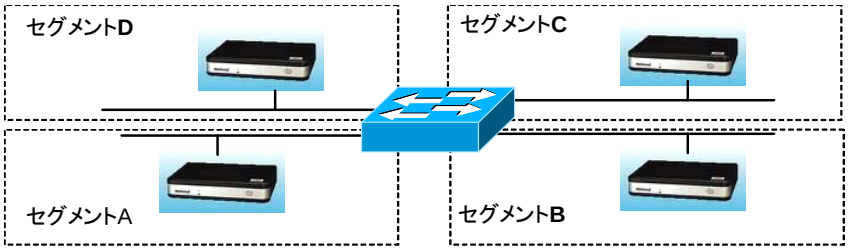


図 3.3: 単機能アプライアンスの追加的設置

- ネットワーク内に複数のセグメントが存在する場合は、セグメントごとにアプライアンスを設置する必要がある (図 3.3)。
- 不正接続端末が「ping 255.255.255.255」などのネットワーク内の MAC アドレスを一網打尽に取得するような攻撃をとった場合、偽 ARP リプライの生成ができない。これは偽 ARP 生成の際にどの IP アドレスの偽 ARP を生成してよいか不正接続 PC 排除アプライアンスが判断できないためである。

このような単機能アプライアンス製品の導入・設置は、別の問題が現れるたびにまた別の単機能アプライアンス製品を導入・設置していくというかたちで、追加的に設置されていくアプライアンス機器が増加していく。

また前述したように、製品の機能としてカバーできるネットワークの対象範囲が限られるものもあり、その場合はロケーションごとにまた新たな設置台数が必要となる。これら設置される機器の増加は管理コストの増加につながり、また安易な機器設置は管理のノン

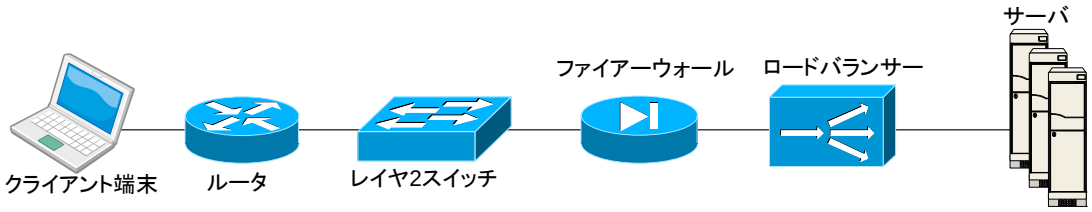


図 3.4: MiddleBox のインライン配置

マネンジメント化を招く。結果としてネットワークの複雑化、運用コストの増加など好ましくない状況を到来させてしまう。

3.3 ネットワーク機器制御層構築による手法

ネットワーク機器制御層とは、ネットワーク機器が従来行っている MAC アドレスや IP アドレスといったヘッダ情報によるスイッチング/ルーティング以外に、ポリシー (Policy) やルール (Rule) といったユーザ定義に従いフレームやパケットの転送動作を行う層を意味する語である。本手法はレイヤ 2 とレイヤ 3 の間に 2.5 層として NW 機器制御層を導入し、フレームの転送を行う手法である。NW 機器制御層を有するスイッチがフレームを受信すると、フレームのヘッダ情報に基づき、どのようなポリシーに基づき受信したフレームを処理すべきか判断し、ルールに従い MiddleBox に転送を行うことで追加的な機能を実現する。「MiddleBox」とは伝送ポリシーを強制的に適用するためのインターネット装置のことである。例えば、ファイアウォールやロードバランサー、Web キャッシュ装置などがそれにあたる。

MiddleBox をネットワーク上の伝送路に単にインライン配置する場合、そこを通過するトラフィックは全て配置した MiddleBox を経由することになる。図 3.4 はファイアウォールやロードバランサーをクライアント端末とサーバ間の通信経路上にインラインに配置した例である。このようにインラインに MiddleBox を配置した場合、本来フィルタリングや負荷分散の必要が無いトラフィックについても MiddleBox を経由するため、フィルタリングや負荷分散が行われてしまう。これを避けるためにファイアウォールに例外を定義したり、サーバの保守時に一時的に機能を停止したりといった、不自由な運用を余儀なくされる。

このような MiddleBox のインライン配置による、柔軟さを欠くネットワーク構造からの開放を計ったのが「ネットワーク機器制御層構築による手法」である。本手法は、レイヤ 2.5 層ともいうべき NW 機器制御層を設け、機能を提供する MiddleBox を、インラインではなく図 3.5 のようにオフラインに配置する。受信したトラフィックを MiddleBox へ転送すべきかどうか Flow 制御を行うことで、「MiddleBox を経由させるトラフィック」と、「MiddleBox を経由させないトラフィック」とで経路を使い分けることを可能としている。

ネットワーク機器制御層構築による手法にはいくつかの先行研究がある。例として、

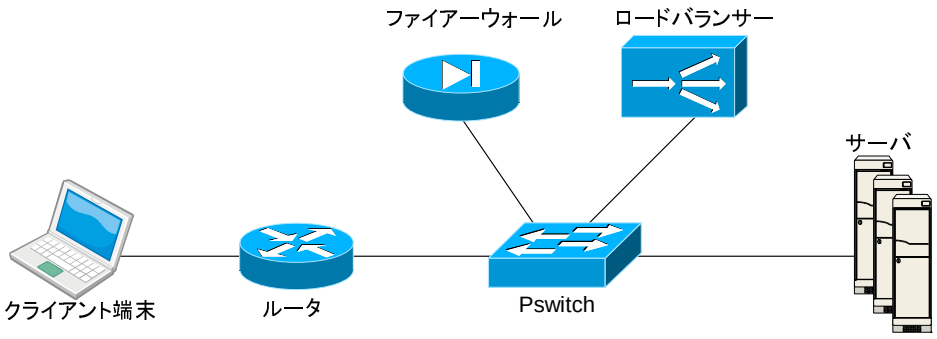


図 3.5: MiddleBox のオフライン配置

Match	Next Hop
R*,	F
Iface F*,	L
L*,	FinalDest

表 3.1: Pswitch に設定されている Rule

Pswitch&Player[9]、SelNet[10]、Ethane[11] などがある。本節では、
Pswitch&Player について詳しく取り上げる。SelNet は Flow 毎に転送すべき次ホップを変更する機構を導入するアプローチであり、これにはネットワーク機器以外にもクライアントホストなどの端末にも当該機構を導入する必要がある。また、Ethane はセンタ装置による Flow の集中管理を実施するアプローチであり、主にセキュリティ用途に向くものである。よって、本論文は、クライアント端末に改修を施したり、機能をセキュリティに限った議論ではないため、この二つの先行研究ではなく、Pswitch&Player について取り上げることにした。

Pswitch&Player は、ネットワークトポロジー上の「Reachability(到達性)」と、転送を決定する「ポリシー」を分離するアプローチである。Pswtich は受信したトラフィックをポリシーに基づき分類し、ルールに従って適切な MiddleBox に転送する。

従来のような、MiddleBox インライン配置による機能適用では、機能適用が不要なトラフィックも全て MiddleBox を経由してしまう。

Pswitch&Player のアプローチは、MiddleBox をオフパス (off-path:脇道) に設置し、当該トラフィックが任意の MiddleBox を経由するように (または MiddleBox を経由せずにエンドホストに到着するように)、Pswitch が制御を行う。

Pswitch によるトラフィックの制御について説明する。簡単な例として図 3.6 のようなネットワーク構成を用いて説明する。図 3.6 中の Pswitch に設定されているルールを表 3.1 に示す。

表 3.1 のテーブルは Pswitch に設定された Rule を表している。ルールの意味を以下に示す。

- 一行目。R(ルータ) から受信した全てのトラフィックは F(ファイアウォール:F/W) に送信する。
- 二行目。IfaceF(Pswitch と F/W:ファイアウォールが接続されているインタフェース、物理ポート) から受信した全てのトラフィックは L(L/B:ロードバランサー) へ送信する。
- 三行目。L(L/B) から送信されてくるトラフィックは全て FinalDest(最終到着先) へ送信する。

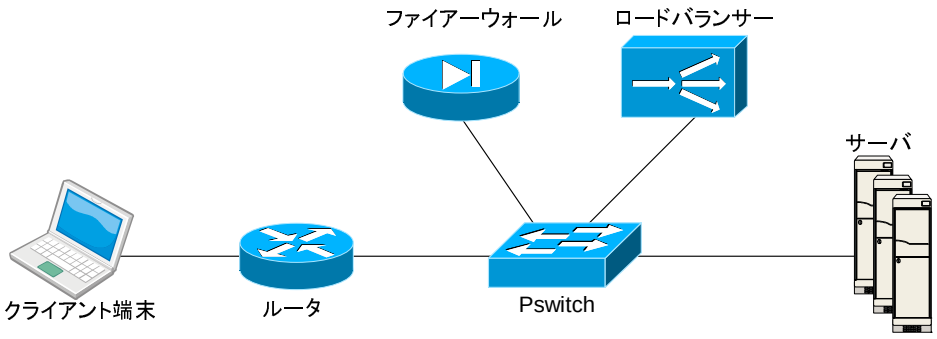


図 3.6: MiddleBox のオフライン配置と Pswitch による制御

このルールに基づくトラフィックの処理の流れは、以下の用になる。まず Pswitch が受信したフレームの送信元 MAC アドレスがルータのものであった場合、宛先 MAC アドレスを Pswitch の MAC アドレスに上書きし、ファイアウォールと接続されたポートに転送される。ファイアウォールにてフィルタリングされたトラフィックが、ファイアウォールと接続されたポートから Pswitch に再び転送されてくると、今度しロードバランサーに転送する。そして当該トラフィックを FinalDest として設定されているサーバ群に転送する。このように Pswitch が中心となって MiddleBox との間でフレームのリダイレクトを繰り返すことで、フィルタリング、ロードバランシング処理を施すことを可能としている。例では接続ポートを単位とした制御を紹介したが、より細かい粒度として、送信元/宛先 MAC アドレス、送信元/宛先 IP アドレス、ポート番号などをキーとしフローの分類と制御も可能としている。

Pswitch&Player では、「ポリシー」をネットワーク管理者が記述し、システムがルールに変換して Pswitch にストアすることでこの制御を実現している。
Pswitch&Player は論文 [9] の中で実際に Pswitch の試作と評価を行っている。Pswitch の試作には、Linux マシンにソフトウェアスイッチである Click[19] をスイッチコアとして使用し、Pswitch のエンジン部には NetFPGA[12] を用いている。

Pswitch&Player のねらいは、物理・論理トポロジーからトラフィックの経路を開放した柔軟なフロー制御を実現するところにある。そのための Pswitch と MiddleBox 間の頻繁なリダイレクトの多用は、データセンターのような高スループット、低レイテンシーな環境であれば問題にならないだろうとしている。

Pswitch&Player の手法は、スイッチ自体に機能を持たせるのではなく、MiddleBox に機能を持たせ、経由させる MiddleBox を変更することで機能のオン/オフや組合せを柔軟に実現している。この手法に対して、本論文では、「機能追加のたびに既設機器のリプレースが強いられ、コスト増を招く」という問題について、Pswitch&Player の手法による対応は以下の 2 点についてまだ不十分であると考える。

- 結局は既設のスイッチを Pswitch に置き換えることで実現しており、リプレースによる手法である。もしリプレースが一度で済むのならば、「追加的なコスト増」とはならないが、Pswitch や Player の機構に変更があった場合、やはり Pswitch のリプレースが行われてしまう。試作では Linux マシンおよびソフトウェアスイッチと NetFPGA によって Pswitch は試作されていたが、論文 [9] の結論としては、pswitch がハードウェアで作成されることを将来的な展望としていた。ハードウェア化されるとさらに機構の変更による刷新はリプレースに頼らざるを得なくなるだろう。

- Pswitch&Player はトラフィックの Flow 毎に、MiddleBox を経由するものと、MiddleBox を経由しないものとに分類することで、MiddleBox のインライン配置による欠点を回避していた。しかしながら、全てのトラフィックは Pswitch を通過する。すなわち、全てのトラフィックについて、Pswitch が Flow を分類し、ルールを適用するという処理を行うことになる。MiddleBox を通過する必要のないトラフィック、

Rule にマッチしない全てのトラフィックも Pswitch にてルール適用有無の判断処理を行わなければならず、その分だけ遅延が発生することは回避できない。

次節では、これまで分析・検討してきた「High-end-Switch の導入による手法」、「単機能アプライアンスの導入による手法」、「ネットワーク機器制御層構築による手法」という手法が、本論文で問題としている「単機能アプライアンスの導入による手法機能追加のたびに既設機器のリプレースが強いられ、コスト増を招く」という問題に対する解法として、どういった点でまだ問題があり、不十分であるのかについてまとめる。

3.4 既存アプローチ・先行研究の問題点

「High-end-Switch の導入による手法」、「単機能アプライアンスの導入による手法」、「ネットワーク機器制御層構築による手法」は各々が独自の方法で、「機能追加のたびに既設機器のリプレースが強いられ、コスト増を招く」という問題への対応方法として取りうる手段ではある。しかしながらまだ不十分な点や残課題がある。

「High-end-Switch」は導入時点での最新鋭のネットワーク機器であり、当面は機能の不足が問題になることはないかもしれない。しかし、その後新しい機能が開発された場合、それを追加できるかは保証されない。結果として機能追加にリプレースが不可避になる可能性がある。

「単機能アプライアンスの導入による手法」は、欲しい機能を追加的に導入できるということで、機能追加の容易さにおいて利点がある。しかしながら、全ての機能が単機能アプライアンスとして開発されているわけではなく、また仕組上、単機能アプライアンスでは実現できない機能もある。また容易に設置できる半面、統合的に運用されるわけではないので、分散配置、もしくは無秩序な配置といった管理上このましくない配置方法にならざるを得ない。インライン上への配置が必要な場合は、ネットワーク上の出口となるゲートウェイ (GW) 付近に設置して、設置台数を抑える代わりに柔軟な適用を犠牲にするか、柔軟に適用するために、分散して配置するかというような難しい問題に直面する。

「ネットワーク機器制御層構築による手法」は、既設のスイッチの置き換えや端末への改修を伴う。置き換えが一度で済むならば、「追加的なリプレース」にはならないので有効な手段となりうる。しかしネットワーク制御層の改版時には結局総入れ換えになってしまう。

また、トラフィックは全てフローの分類、およびルールの適用有無の判断処理を施されるので、その処理遅延が不可避である。

次章では「機能追加のたびに既設機器のリプレースが強いられ、コスト増を招く」という問題に対して、上記の手法の不十分な点を踏まえて、既設のスイッチをリプレースすることなく機能を追加し続け、継続して使用することを可能とする手法の提案を行う。新しい機能のネットワークへの柔軟な導入と、機能追加に伴うコスト軽減を両立する提案の基本アイデアについて述べる。

第4章　基本アイデア

3 章では、2 章にて述べた「機能追加のたびに既設機器のリプレースが強いられ、コスト増を招く」という問題に対する、既存の手法や先行研究について検討を行った。

ネットワークに機能追加が行われるのは、ネットワーク構築時には不要であったり、または検討されていなかった機能、そもそもネットワーク構築時には開発されていなかった機能が、ネットワーク構築後の運用を重ねるにあたって、必要となったがゆえに生じる。つまり、構築時に、機能追加を想定しておくことは難しい。ある程度の予測をもとに、帯域や物理 Port の数に余裕を持ったネットワークを構築したり、シャーシ型のネットワーク機器に機能追加用のモジュールスロットを確保しておくことは可能であるが、追加的、継続的な機能追加や想定外の機能の追加に対応することは非常に難しい。このことから、例え構築当時の最新機種である High-end-Switch であっても、追加的な機能追加に対処するのは困難である。

図 4.1 はファイアウォール機能搭載のレイヤ 2 スイッチをレイヤ 2 スイッチとして採用し、ネットワークに設置した例であるが、収容する端末の増加に伴い設定するフィルタテーブルが増加したり、Intrusion Detection sSystemq(IDS) 等新しいファイアウォール機能が必要になったときにはリプレースせざるを得なくなる。

必要になったときに、必要に応じて機能を追加できるという意味で、単機能アプライアンスの設置による機能追加は理想的ではある。2 章で紹介した不正接続 PC 排除システムや IP Adress Managementq(IPAM) など、アイデアに富んだ非常にユニークな製品が多いのも特徴である。ただし、ブロードキャストされるフレームを検知による動作や、スニフィング技術を用いる手法によって、ネットワークに求められる全ての機能を単機能アプ

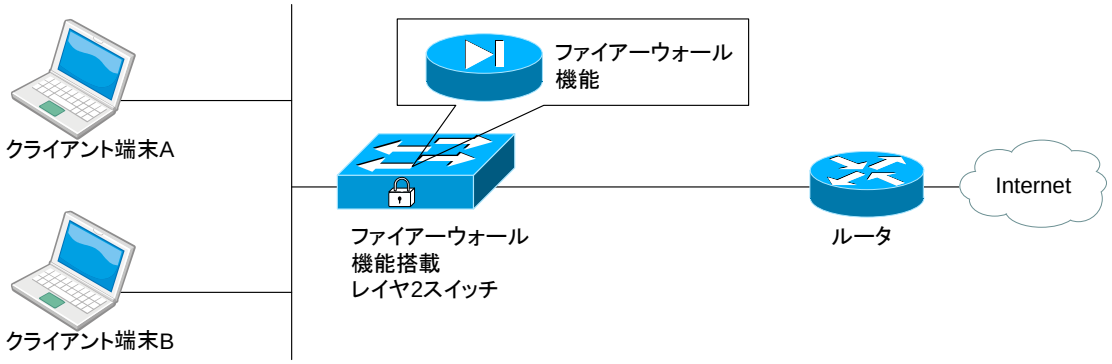


図 4.1: 高機能スイッチの導入

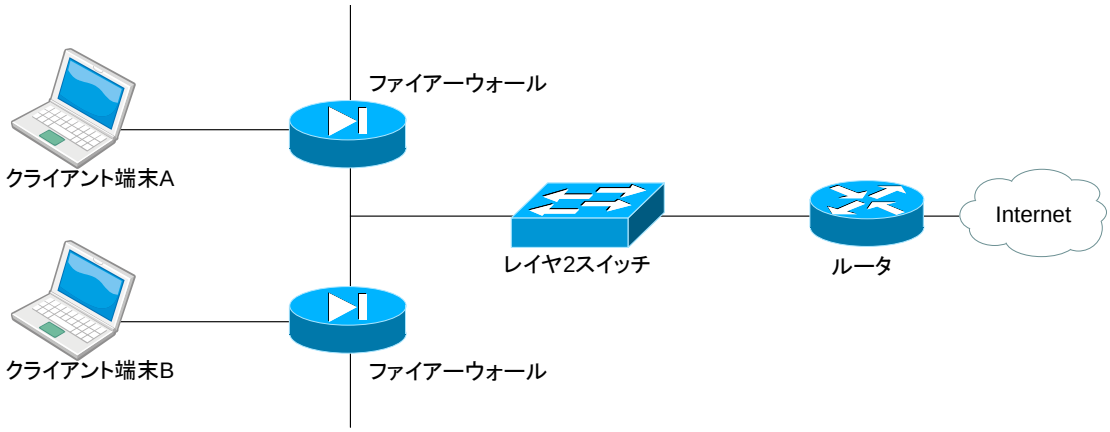


図 4.2: MiddleBox の分散配置

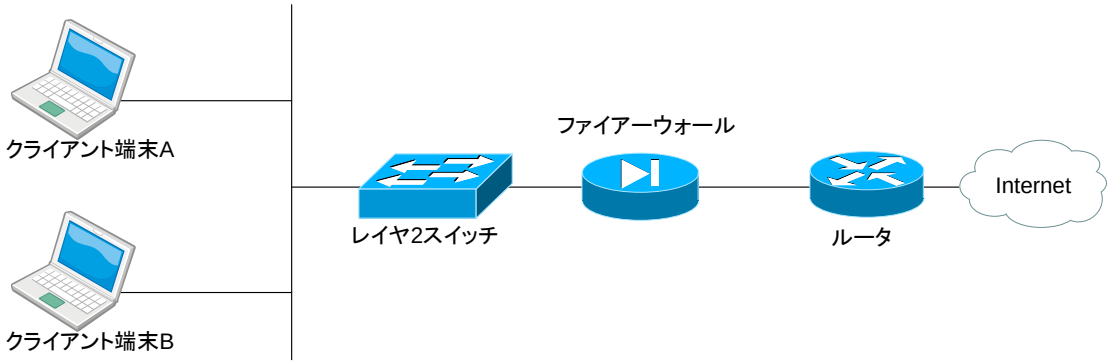


図 4.3: MiddleBox の GW 配置

ラインアスで提供できるかと言えば、それは難しい。また単機能アプライアンスのような諸問題を単機能機器で解決するソリューション型の装置だけでなく、MiddleBox を組み合わせて用いる場合でも、カバーできる機能の範囲には限界がある。

例えば、QoS 機能やフィルタリング機能など、端末から送信されたフレームを直接処理する必要がある機能の実現は、スイッチなどのネットワーク機器にて実現することが求められる。フレームのヘッダ情報を書き換えたり、フレームを解釈して動作を変更したりする必要がある機能を、スイッチなどのネットワーク機器外の装置で実現するとなると、図 4.2 のように端末とスイッチの間毎に当該装置を設置しするか、また図 4.3 のようにネットワークの入出力地点となるゲートウェイ (以下、「GW」) 付近に設置して一括処理するかの選択になる。前者は分散配置した装置間での連携動作や、端末数毎の設置台数が必要となり、後者は装置のインライン配置による柔軟性を欠くネットワーク構成となり、いずれも他の問題を生じさせることとなる。

継続的な機能追加を実現し、インライン配置による柔軟性の阻害からネットワーク構成を開放するという Pswitch&Player のアプローチは面白い試みではある。しかしながら、結局は「Pawitch」という新たなスイッチを導入する必要があり、Pswitch の機能が変更さ

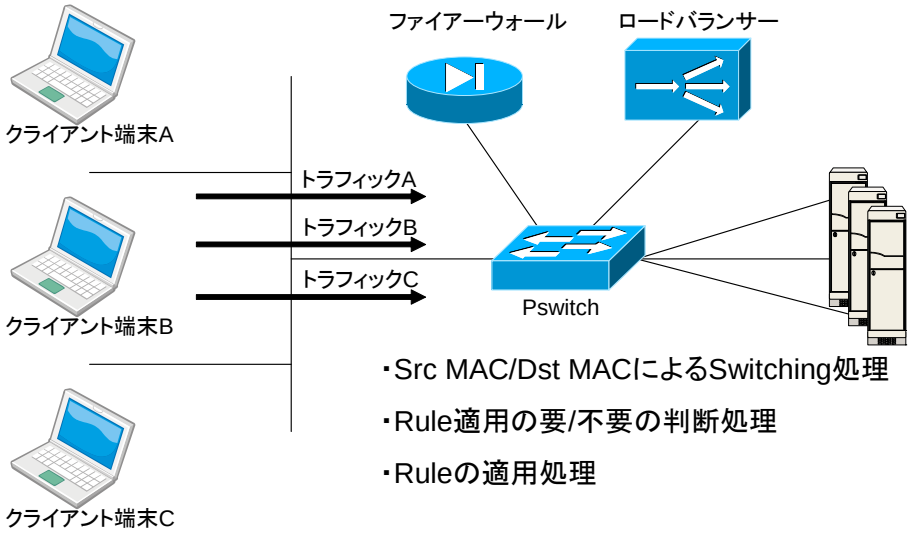


図 4.4: Pswitch のインライン配置と処理負荷の集中

れる度にリプレースが発生する。また、一見 MiddleBox のインライン配置によるネットワークの硬直的構成から、ネットワークを解法するアプローチに見えるが、結局は Pswitch をインライン配置し、インライン配置した Pswitch により Flow を制御しているに過ぎない。つまり MiddleBox の使い分けを行いたい場合は、一度 Pswitch にトラフィックを集約する必要があり、Pswitch を分散配置するか、一台の Pswitch にネットワークを集約するかという配置の問題にもなる。また、図 4.4 のように Pswitch は全てのトラフィックを分析し、Flow 毎の処理・制御を行うため、全てのトラフィック処理にレイヤ 2 のスイッチング処理に加え、Rule の適用を処理するレイヤ 2.5 処理を行う必要が生じ、処理負荷が高くなる。これは実際に処理性能として通常のハードウェアスイッチやソフトウェアスイッチより処理性能が落ちることが論文 [9] でも述べられている。評価の章で後述するが、Pswitch+ファイアウォールという単純な構成で性能を測定したところ、スループットは 350Mbps、レイテンシーは 0.6ms と、ほぼワイヤレートで動作するハードウェアスイッチに比べ大幅な処理性能低下が見られる。あくまでもソフトウェアスイッチと NetFPGA による試作機による測定であるため、Pswitch のハードウェア化などが計られれば処理速度の向上は見込めると思われる。しかし上記で述べたような根本的な問題、つまり pswitch の処理負荷 (すべてのトラフィックについてレイヤ 2+2.5 の処理) が変わることはない。

先に述べた既存手法や先行研究に残存する不十分な点を踏まえ、本論文では、「機能追加のたびに既設機器のリプレースが強いられ、コスト増を招く」という問題を解決するあたらしい手法の提案を行う。手法の提案にあたり本論文で採用する機能追加の基本アイデアを本章で述べる。提案の要旨を以下にまとめる。

- 既存 (あるいは既設の) スイッチを使用し、有効的に活用する
- スイッチ自体への機能追加は行わない

- 継続的な機能追加を実現する
- 機能の使いわけの自由度を提供する

次の節以降では、具体的な提案の中身やアイデア・手法について述べる。

4.1 VLANによるbypass経路の構築

本論分の提案では、High-end-Switch のような高価格・高機能なスイッチや、Pswitch のような新規に開発するスイッチではなく、一般的なレイヤ 2 スイッチを使用する。しかしレイヤ 2 スイッチならばどんなものでも良いというわけではない。以下、本提案に組み込むことを想定するレイヤ 2 スイッチの機能要件についてまとめる。

- ポート VLAN、IEEE802.1q[13] タグ VLAN に対応していること
- telnet、ssh、NETCONF(後述) 等のネットワークによる外部からの制御に対応していること

前節でも述べたが、QoS やフィルタリング等のフレームを端末から送信されたフレームを直接処理する必要があるため、スイッチ等のネットワーク機器で機能を実現する方が、処理の主体、そして配置の点からしても望ましい。しかしながら、スイッチに機能を実装するためにはスイッチのハードウェア・ソフトウェア等の内部処理機構に改造を施すことが必要になる。結局はリプレースによる機能追加になってしまう。

そこで、本論文ではスイッチ自体の内部処理機構には手を加えず、端末から送信されたフレームを処理するため、VLAN を使用したバイパス経路の形成と、バイパス経路上に配置する FunctionBox による処理機構を用いる。

スイッチが VLAN で分割されていない状態 (全ての物理ポートがデフォルト VLAN に所属している状態) において、端末 A と端末 B が通信を行うと、図 4.5 のように端末 A から送信されたフレームはスイッチの内部バスを経由して端末 B に受信される。

ここで図 4.6 のようにスイッチを VLAN により分割し、送信端末 A と受信端末 B を別々の VLAN に所属させた場合、それぞれの所属する VLAN が異なるため、図 4.5 スイッチの内部バスを経由した通信ができなくなる。そこで、図図 4.5 のように送信端末 A が所属する VLAN と受信端末 B が所属する VLAN の bypass 経路となるケーブルを接続する。このバイパス経路を構築することで、送信端末 A と受信端末 B はバイパス経路を経由した通信が可能となる。

VLAN によりスイッチを論理的に分割し、フレームにバイパス経路を経由させることで、スイッチの内部バスを流れる通信をスイッチの外に出すことができる。そしてこの bypass 経路上で何か処理を行う装置を設置すれば、スイッチの内部処理機構に改造を施すことなく、フレームの処理・解釈機構を加えることを可能になる。次節では実際にこのフレームの処理・解釈を行う機構として動作させる「FunctionBox」について述べる。

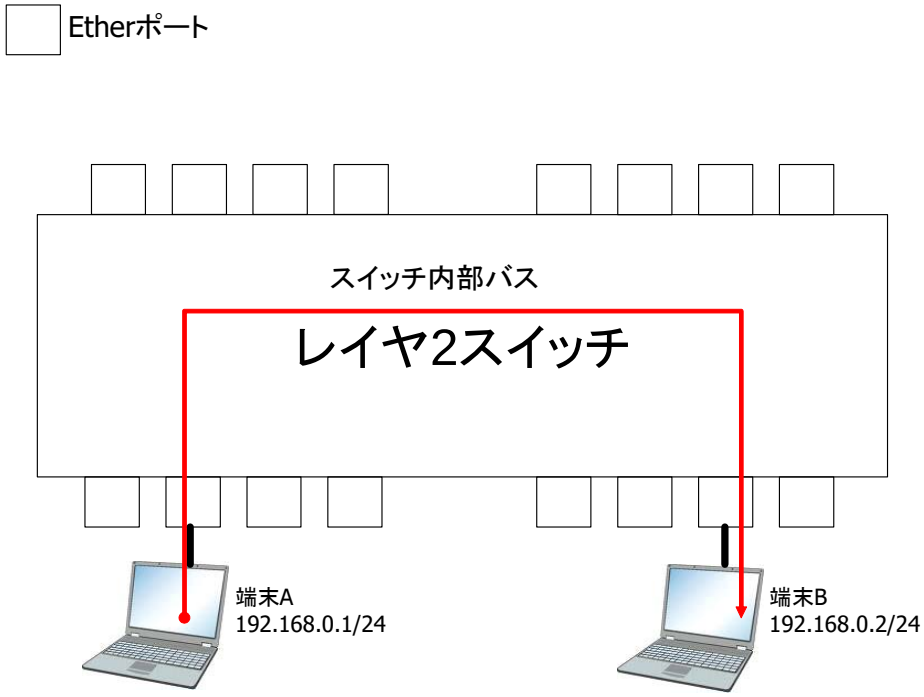


図 4.5: スイッチの内部バスを経由する通信

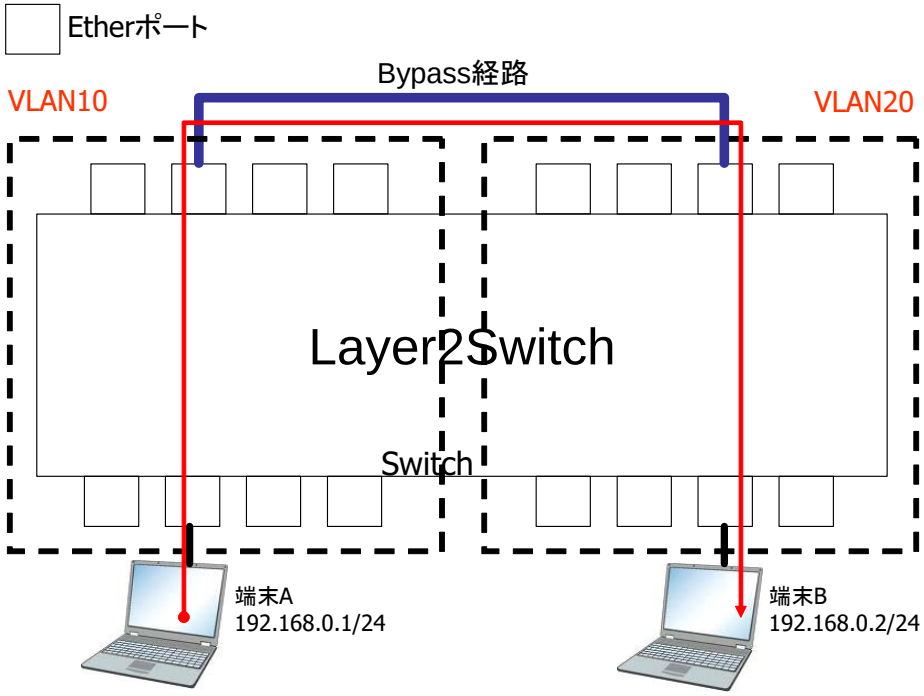
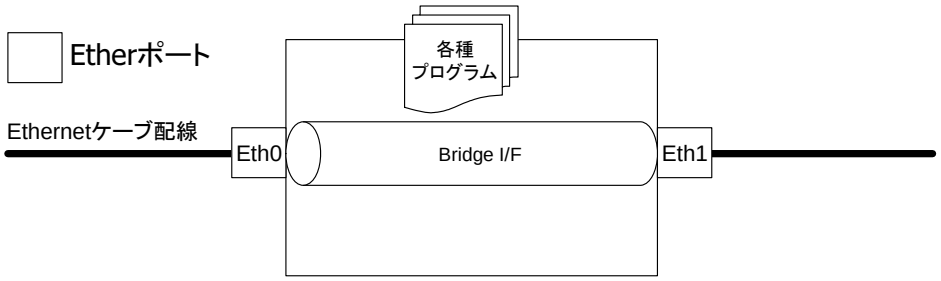


図 4.6: VLAN 間を接続する bypass 経路を経由する通信



・Linux等のOSを搭載したPC
・Ethernet I/Fを2つ以上搭載
・レイヤ2にインライン配置する際は、2つのI/FをBridge化して設置
・レイヤ3機器として配置する際は、ルータとして設置
・インタフェースで送受信するデータに対してフィルタリング、監視、暗号化
などの処理を行うプログラムを動作させる

図 4.7: FunctionBox の構成

4.2 FunctionBoxによるフレーム処理

bypass経路上に設置する装置について述べる。本論分ではこの装置のことを「FunctionBox」と呼ぶ。FUnctionBox の構成を図 4.7 に示す。FunctionBox は Linux 等の OS を搭載した PC であり、特別なハードウェア等は必要としない。このように PC を「機能を提供する箱」として用いる。

2 つの Ethernet インタフェースを装備した PC を FunctionBox としてバイパス経路上に配置し、FunctionBox に任意のプログラムをインストールし動作させることで、任意の機能を実現することができる。これにより、スイッチの内部処理機構に手を加えずに、機能を追加することを可能にする。

フレームを直接処理する L2 フィルタリングを行いたい場合は、2 つのインタフェースを bridge 接続することで、インラインに配置することができる。NAT などの L3 以上の処理であれば、2 つのインタフェースに別々のセグメントのアドレスを割当、ルーティングさせることでインラインに配置する。FunctionBox の装置構成と、バイパス経路への配置を図 4.8 に示す。

FunctionBox で、受信したフレームの MAC アドレスを読み取り、フィルタするプログラムを動作させれば、レイヤ 2 フィルタ機能をスイッチに加えることができる。

- スイッチにレイヤ 2 フィルタ機能が実装されていない場合
- レイヤ 2 フィルタ機能は実装されているが、フィルタ設定数の上限を超えてしまっており、さらにフィルタ条件を設定できない場合
- レイヤ 2 フィルタ機能は実装されているが、物理ポート毎にフィルタの ON/OFF を切替えられない場合

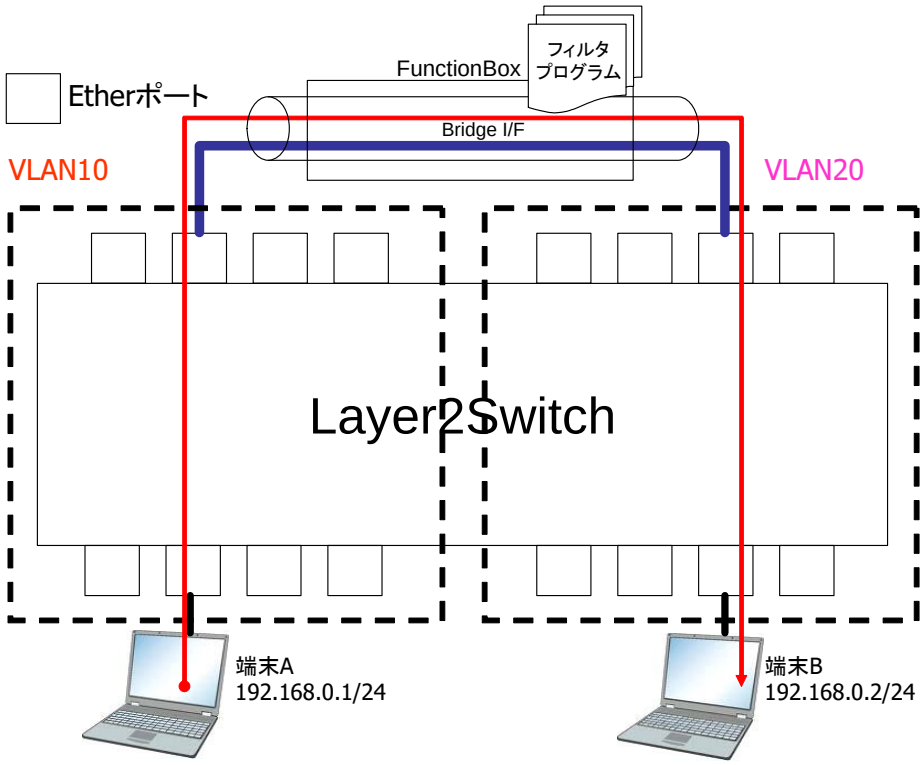


図 4.8: 既設レイヤ 2 スイッチへのフィルタリング機能の追加

このような場合、より高機能なフィルタ機能やフィルタ設定数を実装したスイッチ にリプレースすることが検討されるが、本機構を用いれば既存機器をリプレースすることなくフィルタ機能を追加・拡張することができる。

機能追加のためにスイッチ自体の内部処理機構に手を加えたい場合、方法は、開発元のベンダに依頼するか、専用機器を開発するしかない。しかしここで示した一般的な PC を用いた、FunctionBox による機能追加であれば、自由にプログラムを作成・改修することができ、ネットワーク機器に実装するのに比べ、メモリや電源等の制約も少ない、より自由な機能開発・実現が可能となる。

4.3 スイッチ多段構成下におけるバイパス経路の構築

4.2 節ではバイパス経路に設置し、機能を実現するプログラムを動作させる FunctionBox について述べた。しかし 1 つのスイッチに 1 つの FunctionBox を設置するのでは、単機能アプライアンスや MiddleBox を分散配置するのとあまり効果の面でかわらなくなる。しかしながら、Tag based VLAN の機構を使用し論理ネットワークを構築することで一つの FunctionBox を用いて複数のスイッチに対して機能を提供することを可能とし、ネットワーク全体に機能を提供・追加することができる。

Tag-based VLAN を使用して論理ネットワークを構築し、送信端末と FunctionBox、受

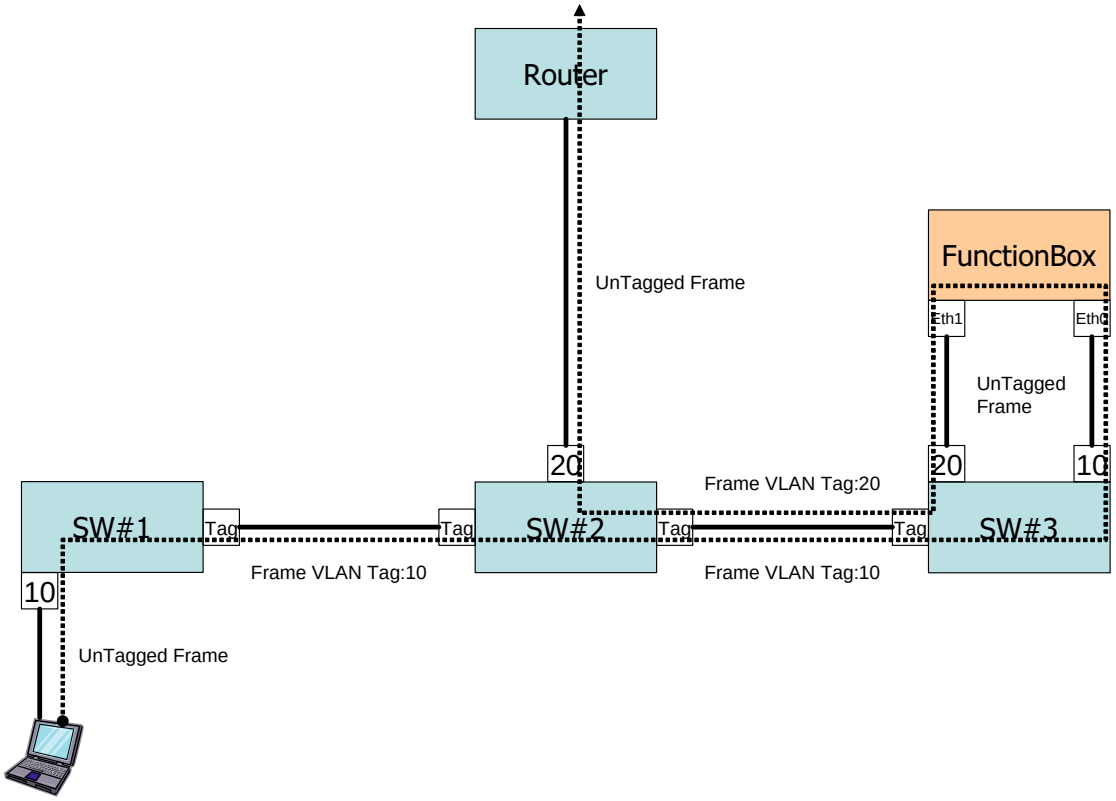


図 4.9: TagVLAN による通信経路の構築

信端末を結ぶパスを形成することにより、FunctionBox が接続されているスイッチに直接収容されていない端末による通信であっても、FunctionBox が設置されている nypass 経路を経由させることが可能である。このような送信端末/FunctionBox/受信端末を結ぶ経路を構築する TagVLAN による論理ネットワークを本論分では「VLAN パス」と呼ぶ。

VLAN パスにより FunctionBox の機能をネットワーク全体に提供する様子を図 4.9 に示す。

異なるスイッチに収容されている送信端末と受信端末が bypass 経路を経由する通信について説明する。図 4.9 における送信端末は図中 SW#1 に接続されている PC であり、受信端末は図中のルータ (Router) である。

本来ならばフラッディングによる学習により構成された各スイッチの FDB に基づいたスイッチングが行われ、PC からルータへは、SW#1/SW#2 を経由して到達する。よって SW#3 に接続されている FunctionBox を経由することはない。

しかし TagVLAN を用いて VLAN パスを構成することにより、PC からルータに向かう通信を FunctionBox を経由するよう構築することができる。

ではその VLAN パスの構成方法と、VLAN パスによる通信の様子について説明する。図 4.9 中の□はスイッチおよび FunctionBox の物理インタフェース (ポート) である。□中の数字は当該インタフェースに設定された VLAN ID を示す。「Tag」はトランクポートに

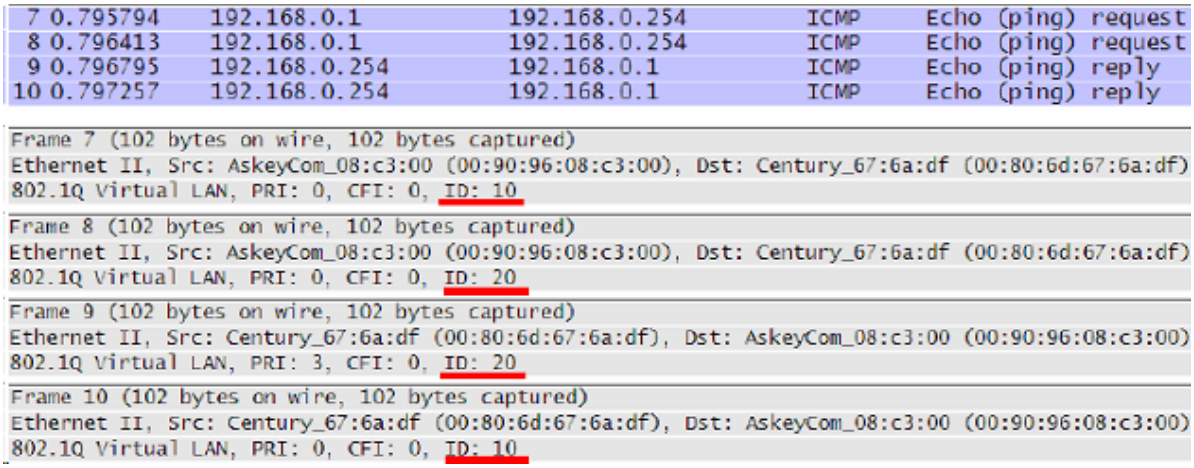


図 4.10: Wireshark によるパケットキャプチャ

設定されていることを示す。

SW#1 にフレームが届くと、PC が接続されているポートは VLAN ID 10 に設定されているため、フレームのヘッダにに VLAN タグ「VLAN ID 10」が付与される。フレームは SW#1 のトランクポートから VLAN ID 10 が付与されたまま SW#2 のトランクポートに到達する。ここで、SW#2 とルータが接続されているポートは VLAN ID 20 に設定されているため、フレームはここを通過することができず、SW#2 のトランクポートから SW#3 のトランクポートへ転送される。SW#3 に到着したフレームは、VLAN ID 10 が付与されているため、同じ VLAN ID 10 が設定されているポートから VLAN Tag が外され、FunctionBox の Eth0 インタフェースに到着する。フレームは再び FunctionBox の Eth1 インタフェースから SW#3 に到着し、VLAN Tag「VLAN ID 20」を付与される。SW#3 のトランクポートから SW#2 に転送され、フレームは、VLAN ID 20 が付与されているため、同じ VLAN ID 20 が設定されているポートから転送され、VLAN Tag が外され、ルータのインタフェースに到着する

図 4.9 中の SW#2-SW#3 間で、PC(IP アドレス 192.168.0.1) からルータ (IP アドレス 192.168.0.254) への Ping 発行をキャプチャしたものを図 4.10 に示す。SW#2 から SW#3 へ向かう通信の際に VLAN ID(10) が使用され、FunctionBox を経由したことで、SW#3 から SW#2 に戻る通信では VLAN ID(20) が使用されていることがわかる。

使用したい機能を提供する FunctionBox を経由する VLAN パスを設定することで、機能の提供を制御する。機能ごとのパスの使い分けについて図 4.11 に示す。

ここで本論文で提案する機能の追加手法の基本アイデアについてまとめる。

- スイッチを VLAN により論理的に分割し、分割された論理グループ (VLAN) 同士をつなぐ bypass 回路を構築する。
- bypass 経路上にインライン配置する FunctionBox(Ethernet インタフェースを二つもつ PC) にフレームを解釈・処理するプログラムを実装することで機能を実現する。

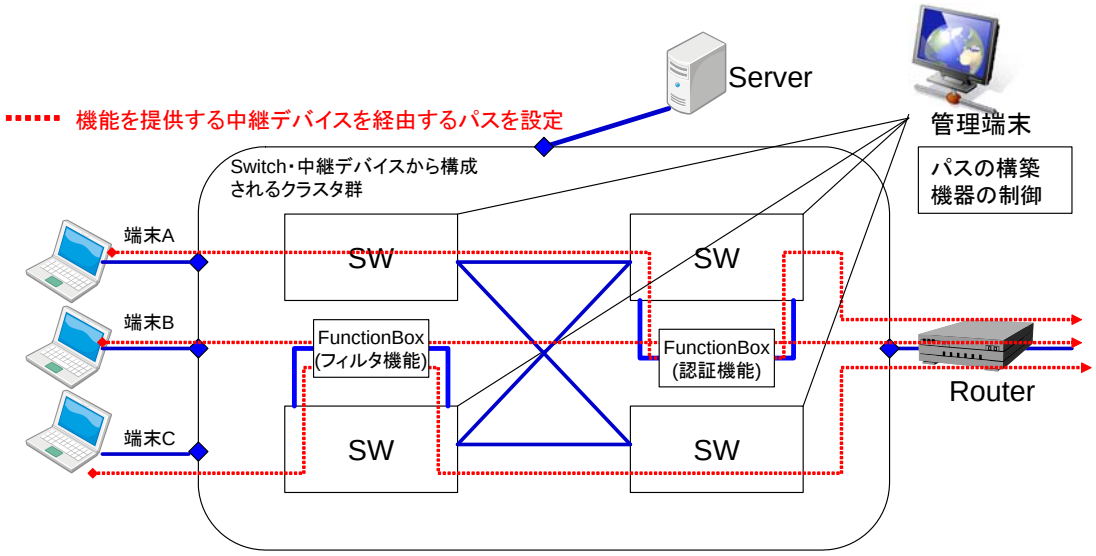


図 4.11: 利用機能の選択とパスの構築

- 複数のスイッチにより構成される LAN において、Tag-based VLAN の仕組を用いて FunctionBox により提供される機能をネットワーク全体で利用可能とする。

4.4 基本アイデアを用いた機能追加手法の提案への課題

次々と開発・考案される機能を、スイッチのリプレースによらず、スイッチの内部処理機構に手を加えず、どのように追加していくのかという本論文の基本アイデアを前節までまでまとめた。これらの基本アイデアを用いて、本論文ではネットワークに機能を提供する手法を提案する。

次に、提案にあたって検討しなければいけない課題がいくつか存在する。まず、追加した機能の使用有無の自由度を提供する方法が必要になる。ネットワークに、前節にてまとめた基本アイデアにて機能を追加した後、当該機能を「使用する/ しない」、「適用する/しない」といった自由度を与えることができなければ、MiddleBox のインライン配置による機能の強制適用と変わらなくなってしまう。自由度についてはネットワーク全体で全機能の ON/OFF を行うといった「0 か 1 か」といったものでなく、

- どの範囲について、

  ネットワーク全体のうち、機能提供範囲をどのように指定するか等

- どの機能を、あるいはどのような機能の組合せを、

  機能の選択、機能の組合せ、機能の適用順序等

- どのような粒度で適用するのか、

ネットワーク全体、物理的なスイッチ単位、物理ポート単位等

また、機能の ON/OFF を FunctionBox を経由する VLAN パスで実現するにしても、ネットワーク管理者やユーザが、「FunctionBox と機能の対応」、「VLAN ID と経由するパスの対応」を意識せずに利用できる仕組についても考案しなければならない。そしてそれを実現する機構が必要になる。VLAN により物理トポロジーに拘束されない、自由な論理トポロジーを構築できることは、目視だけでは把握しにくい複雑な構成を構築することを意味する。物理 Port や提供する機能を把握しながら VLAN を割当て、ネットワークを設計することは繁雑な、そしてミスを招きやすい作業である。また設計・構築した担当者だけが管理できるネットワークでは意味をなさず、それを利用する方法も考案しなければならない。

そして、実際に設計どおりにネットワークを構築し、機能の提供有無により構成を変更するためにはネットワーク機器を制御する手段が必要になる。従来の CLI のような人間が直接コマンドを実行する仕組だけでは、上記の作業の度に、ネットワーク機器のコマンドを習得した作業員の出動が発生してしまう。

基本アイデアを用いた提案システムを実現するための、これら「機能提供の自由度」、「利用・使用方法」、「ネットワーク機器の制御手段」について次章以降で述べる。

第5章　提案システム

4 章では「機能追加のたびに既設機器のリプレースが強いられ、コスト増を招く」という問題に対する先行研究や既存手法に代わる、新しい手法の基本アイデアを述べた。

- スイッチを VLAN により論理的に分割し、分割された論理グループ (VLAN) 同士をつなぐ bypass 回路を構築する。
- bypass 経路上にインライン配置する FunctionBox(Ethernet インタフェースを二つもつ PC) にフレームを解釈・処理するプログラムを実装することで機能を実現する。
- 複数のスイッチにより構成される LAN において、TagVLAN の仕組を用いて FunctionBox により提供される機能をネットワーク全体で利用可能とする。

本章ではそれら基本アイデアを実際に使用し、機能を追加する手法を提案する。提案にあたっての課題を 4.4 にて挙げていた。本章ではまず先に挙げた課題についての検討を行う。その後、課題の検討から得られた知見をもとに、提案手法を NCC(Network Circuit Compiler) 試作し、手法の動作原理について解説を行う。

では、まず 4.4 にて挙げた検討すべき具体的な課題を検討し、再度整理したものをに以下に挙げる。

- 機能提供の自由度

  本提案手法では、各スイッチの物理ポートを単位として、機能提供の自由度を提供することとした。収容されている物理スイッチが異なっていても、ロケーションによらず物理ポート単位で機能制御できることを可能とする。MiddleBox のインライン配置のように、そこを通過する全てのトラフィックが当該 MiddleBox の影響を受けるような構成ではなく、各物理ポート毎に独立した機能または機能の組合せを提供できることを目的とする。

- 利用・使用方法

  本提案手法によるネットワークが構築された後、ネットワーク管理者は各物理ポート毎に制御を施すため、物理ポートを指定して、その物理ポートに接続される端末に適用したい機能 (フィルタ、認証、通信ログ監視) を選択することを可能にする。この物理ポートの指定と機能の選択により、自動的に VLAN パスが形成され、切り替わる仕組を提供する。

- 実現方法

  4 章で述べた基本アイデアを利用して、機能の追加および提供を実現するためには、「任意の VLAN パスを設計する機構」と設計された VLAN パスを構築するために「実際にスイッチに指示・操作する機構」が必要となる。本提案手法では、「任意の VLAN パスを設計する機構」として「NCC:Network Circuit Compiler」を提案する。

提案手法 NCC については後述するが、設計された VLAN パスを構築するためには、4.3 で述べたような物理ポート毎の VLAN ID の設定やスイッチ間をつなぐトランクポートの設定などをスイッチに対して行わなくてはならない。次節ではこれらスイッチの制御方法について、NCC で用いる手段について述べる。

5.1 スイッチ制御の仕組—AX-ON-API—

ネットワーク機器の制御を行う場合、その多くはネットワーク機器のコンソールポートと PC を直接コンソールケーブルで接続してオペレーションを行う CLI による制御が一般的である。またはコンソールケーブルの直接接続ではなく、ネットワーク機器と telnet セッションを構築し、telnet からの CLI 入力による方法も採られる。コンソールケーブルか telnet によるネットワーク経由かの違いこそあれ、入力される内容は人間が直接入力する CLI コマンドであり、基本的には CLI コマンドの投入とネットワーク機器からの応答、および設定内容の確認という手続きでネットワーク機器の制御が行われる。従来のネットワーク機器制御用の CLI は人間が直接入力・確認を行う制御方法としては有力であるが、ネットワーク機器の制御を自動化したい、または外部事象の変化により動的にネットワーク機器の設定を変更したいといった、「ネットワーク機器設定の自動化」には不適なインタフェースであった。CLI はあくまでも人間が入力し実行するためのインタフェースであり、プログラムから実行するには使いにくいインタフェースである。ネットワーク機器の自動制御を行うために、CLI をスクリプト言語等から実行するため CLI over Telnet や Expect[18] 等の試みが行われてきた。プログラムからネットワーク機器を制御するためのインタフェース (API) への希求は根強いものがある。

さらに CLI の問題点として、ベンダーや機種が異なるとコマンド体系が変わり、同様のオペレーションを行うにしても、コマンドやパラメータが異なるという問題もある。オペレータは同様のオペレーションを行うために、ベンダーや機種がことなった場合、ベンダ、機種毎の CLI 体系を習得する必要があり、これもまたネットワーク機器の自動制御を妨げる要因ともなっていた。

また CLI を持たない、Web ブラウザから設定をする機種や、CLI ではあるが、コマンドラインによらないメニューモードを設定インタフェースとして備える機種もある。

```
<Cisco 社 Catalyst の例>

#config terminal
(config)#interface Ethernet0/1
(config-if)#ip address 192.168.0.1 255.255.255.0

<Junipper Networks 社 Summit の例>

#set ip address 192.168.0.1/24
```

5.1.1 NETCONF

AX-ON-API は、NETCONF[14] を採用し、通信手段に SOAP を用いたネットワーク機器制御用 API である。The Network Configuration Protocol (NETCONF) はネットワーク機器の情報取得と設定インターフェイスを標準化するプロトコルであり、RFC4741 で定義されている。NETCONF はネットワーク機器のベンダ、機種によらない制御機構の統一だけでなく、将来的には PC やサーバなどのコンピュータを含むネットワークシステム全体の運用管理も視野に入れているプロトコルである。設定の送受信には XML ファイルを利用する。

NETCONF API の登場によりネットワーク機器をプログラムから操作することが可能になり、プログラムによる運用の自動化や障害の自動復旧などへの活用が期待されている。

策定がほぼ終了しているのは、NETCONF が想定するレイヤ「トランスポート・プロトコル層」、「RPC 層」、「Operations 層」、「Content 層」のうち下位 3 層である。NETCONF プロトコルが想定する階層構造を図 5.1 に示す。トランスポート・プロトコルは XML を送受信するためのプロトコルレイヤであり、HTTP、SOAP、BEEP などを使用する。その上位に NETCONF インタフェースの呼出を規定する RPC 層とネットワーク機器に対するコマンドを操作である Operations 層がある。

最上位の Content 層は実際にネットワーク機器が実行する設定内容を記述する層である。しかしながらネットワーク機器が持つ機能を規定するデータモデル（記述形式）がまだ未策定のため、ベンダ固有の設定コマンドがそのまま使用されていることが多い。

5.1.2 AX-ON-API

2006 年末ごろから、NETCONF に対応しソフトウェア開発のための操作 API を提供するベンダも登場してきている。今回は実際にアプリケーションを試作するために、アラクサラネットワークス株式会社が提供する NETCONF 対応 API である AX-ON-API を取り上げる。

AX-ON-API(AX-Open Networking-Application Programming Interface) は、アラクサラネットワークス社製品の運用・管理の自動化を支援する OAN(Open Autonomic Networking) の基盤技術である [16]。NETCONF を採用しており、通信手段には SOAP を採

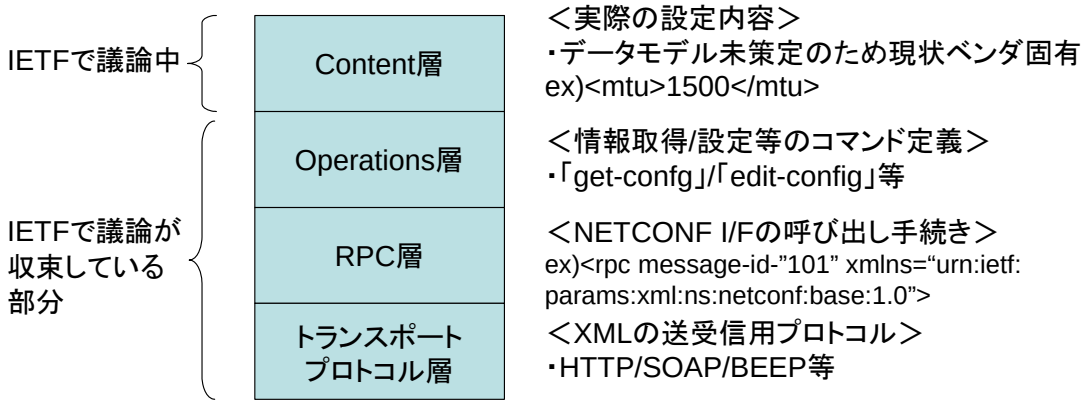


図 5.1: NETCONF のレイヤ構造

用している [17]。SOAP メッセージの交換保護に HTTPS を使用することも可能になっており、SOAP メッセージの交換によるネットワーク機器操作をセキュアに行うことも可能である。OAN-API1.3 で操作することのできる機能を次に挙げる。

- Node 操作。ノードの IP アドレスについての参照・変更 API。
- Line 操作。ノードのポート、インタフェースについての参照・変更 API。
- Abstraction ポート操作。リンクアグリゲーションについての参照・変更 API。
- VLAN 操作。VLAN についての参照・変更 API。
- AccessList IFAccessList 操作。アクセスリストについての参照・変更 API。
- IpRouteStatic 操作。スタティックルーティングについての参照・変更 API。
- MacAddressTbl 操作。MAC アドレステーブルについての参照 API。
- NodeStatus 操作。装置情報 (装置のモデル名、ソフトウェアバージョン) の参照 API。

AX-ON-API はアラクサラネットワークス社製品を制御可能とする Java クラスライブラリ集であり、Java によるプログラミングインタフェースを提供する。

本論文では研究ではネットワーク機器の制御に AX-ON-API を使用し、スイッチの VLAN 設定を行う Java プログラム「VlanController」を作成した。本プログラムはコマンドラインから、

- 制御対象装置の IP アドレス、
- 物理ポート、
- 所属させたい VLAN ID、

を指定して実行する。

VlanController を用いて、

- 制御対象装置 (IP アドレス 10.0.0.2) の、
- 物理ポート 5 番を、
- 所属させたい VLAN ID:2 に所属させたい場合、

以下のように指定して実行する。

```
VlanController.jar 10.0.0.2 "port 0/5" 2
```

5.2 NETCONF 非対応製品の制御

前節では VLAN の設定および設定変更に使用するネットワーク制御に使用する AX-ON-API について述べた。前述したように、AX-ON-API は NETCONF という統合インタフェースを採用しているが、現状では AX-ON-API から制御可能なネットワーク機器はアラクサラネットワークス社製品のみである。非アラクサラ製品の制御には CLI を telnet から操作する CLI over Telnet の仕組を使用する。

後述するが、提案システムでは操作対象機器が AX-ON-API 対応製品か他のインタフェースを用いる必要があるかを判断して、インタフェースを使い分ける機構を有している。「スイッチ A の物理ポート 3 番を VLAN 10 に所属させる」というオペレーションが発行されると、スイッチ A のタイプを判断し、AX-ON-API を用いるのか、CLI over Telnet(対象機器が Cisco 製品の場合の CLI は IOS コマンド) を用いるのかを判断し、制御コマンドを発行する。

NETCONF の根本理念である「ベンダ、機種の違いを問わない共通インタフェース」という理想は現状では停滞している。標準化をめぐるベンダ間の主導権争いや、IETF でのデータモデル構築が難航するなか、2006 年頃からベンダは各社各様の NETCONF を搭載した製品をリリースしている。

先に述べた AX-ON-API はアラクサラネットワークス社の NETCONF 搭載製品であるが、Cisco 社は「NX-OS」、Juniper 社は「JUNOS」といった NETCONF 準拠の製品をリリースしている。現状では各社の NETCONF 間での相互運用は確保されておらず、NETCONF の理念からは遠ざかっている。

本論文が対象としているような、マルチベンダで構成されたネットワークを前提としたシステムを考えるとき、ベンダや機種を問わない統合管理インタフェースとしての NETCONF API の存在が待望されるところである。しかしながら、本論分の執筆時点では統合的な NETCONF は統合管理インタフェースとして過渡期にあり、「ベンダ、機種を問わない共通インタフェース」としての役割を果たすには至っていない。よって、現状のネットワー

ク製品の制御において一般的な CLI による制御と、期待される制御インタフェースとして NETCONF API による制御をスイッチのタイプによって使い分ける機構とした。

ネットワーク機器の制御方法は、前章で述べた機能拡張の基本アイデアを実現するための「手足」である。今回試作においては AX-ON-API と IOS が「手足」として使われるが、手足の汎用化や多機種対応は本論文のテーマとしてはメインではないため、試作段階でシステムに組み込むことが可能なネットワーク機器は、現状ではアラクサラネットワークス社製品と Cisco 社製品となっている。

5.3 機能オン/オフの VLAN による表現手法

5.1 節ではネットワーク機器の制御の仕組について述べた。

次に、物理ポートへの機能の提供をどのように実現するかという手法について述べる。もしネットワーク管理者が、物理構成図と論理構成図、および FunctionBox と機能との対応表をたよりに、スイッチの制御を CLI 等のインタフェースを用いて、任意の物理ポートに任意の機能を提供するための VLAN パスを構築することは、

まず物理構成図をもとに、機能を適用する物理ポート、機能を提供する FunctionBox の確認、FunctionBox とスイッチとの接続状態および使用ポート、ゲートウェイの位置とスイッチとの接続状態および使用ポートを把握する。次に VLAN ID をどのように割り当てるかを設計する。クライン後端末が接続された物理ポートと FunctionBox の入力側インタフェースが接続された物理ポートを同じ VLAN ID に所属させ、FunctionBox の出力側インタフェースが接続された物理ポートとゲートウェイが接続された物理ポートを同じ VLAN ID に所属させる。この場合 2 つの VLAN ID が必要になるので、デフォルト VLAN(VLAN ID:1) やすでに使用されている VLAN iD 等をさけて割り当てる。その後、上記の設計をもとに、各スイッチに対しオペレーションを行うことになる。このような運用の仕方は、繁雑でありかつミスも多く、頻繁に機能のオン/オフを行ったり、機能を適用する物理ポートを増減したりといったケースを想定すると、実用に耐えない。

そこでネットワーク上に存在する FunctionBox が物理スイッチ上のどの物理ポートに接続されているか、端末が接続される物理ポートにはどの VLAN 番号設定すればよいかという、ネットワークの物理構成や論理構成を意識せずに使用できる機構を考案した。

ある機能をを提供する FunctionBox を「F」としたネットワーク構成図を図 5.2 にしめす。

動作の説明のためとと、ネットワーク管理者が本機能を使用するイメージの補助として、簡単な GUI 画面を試作した。GUI について 5.3 に示す。

この GUI は、

- どのスイッチの、
- どの物理ポートに、
- どの機能を提供するか、

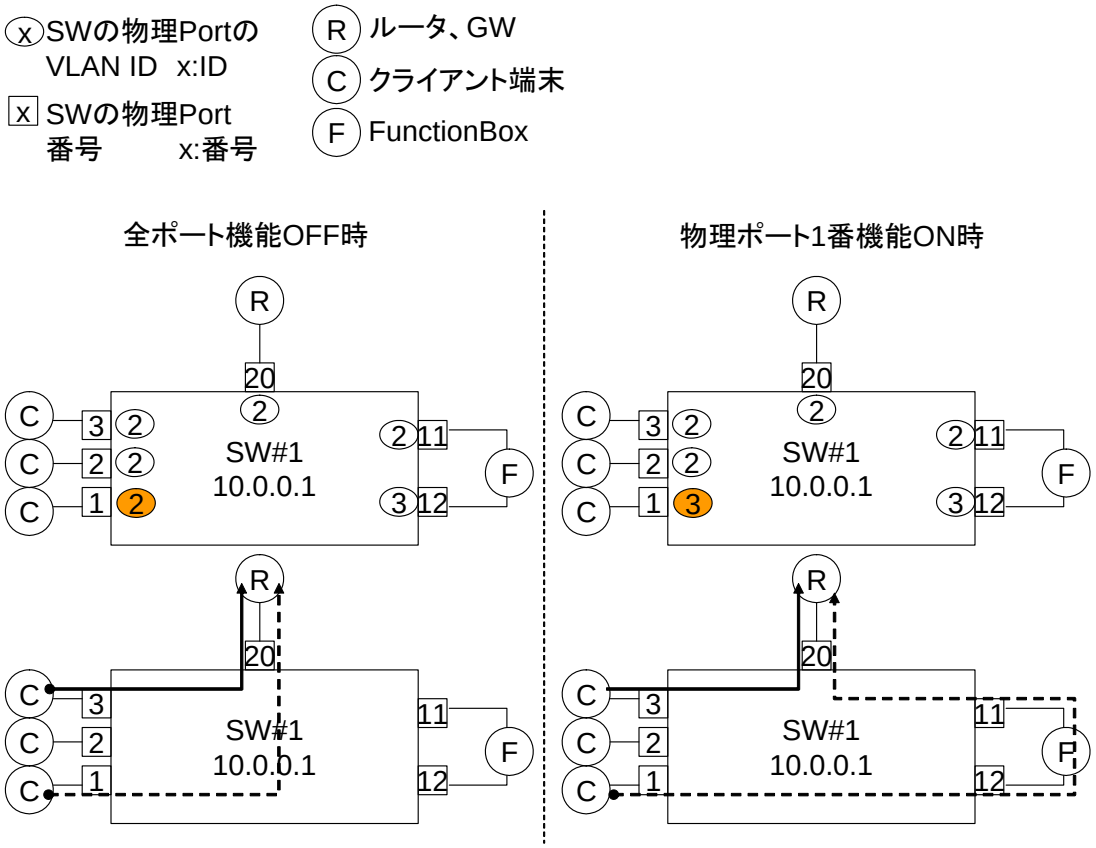


図 5.2: GUI アプリケーションによる機能の選択制御と構築される論理トポロジー

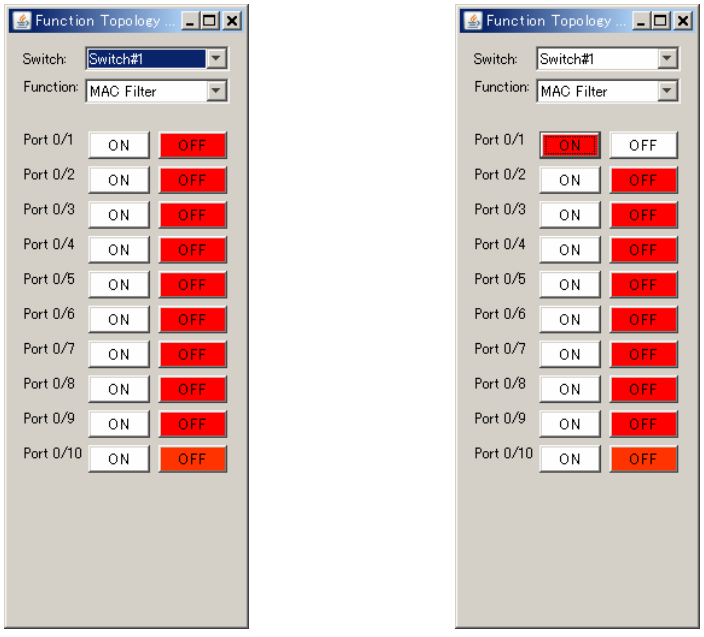


図 5.3: 機能の選択制御を行う GUI アプリケーション

ということを選択させる仕組になっている。

図 5.2 中のスイッチ SW#1 の物理ポート 1 番に接続された端末がルータを経由した通信を行う際に、機能 F を提供したいとした場合、GUI を操作し、スイッチの物理ポート 1 番について、機能「F(MAC Filter)」を選択することで、自動的に VLAN が構成される。

機能 F が OFF のときは図 5.2 左のように、1 番ポートは VLAN ID 2 に所属しており、同じく VLAN ID 2 に所属しているルータとの接続ポート 20 番と、スイッチの内部バス経由でルータと通信を行う。図 5.2 左上段が各ポートの VLAN 設定、左下段が通信経路を示している。

機能 F を 1 番ポートに提供 (機能 F を ON) にした際に構成されるのが、図 5.2 右である。図 5.2 右上段が各ポートの VLAN 設定、右下段が通信経路を示している。GUI からの機能 F:ON の指示で 1 番ポートは VLAN ID 3 に設定変更される。設定変更により通信経路も、変更される。機能 OFF 時にはルータと接続している 20 番ポートと同じ VLAN に所属していたため、スイッチ内部パスを経由してルータに到達していたが、ルータと接続している 20 番ポートとは所属 VLAN が異なるため、スイッチの内部バス経由では到達できない。よって、1 番ポートは VLAN ID 3 に設定されているため、同じく VLAN ID 3 に所属している、FunctionBox と接続されている 12 番ポートから FunctionBox 内の bridge インタフェースを経由して、11 番ポートへ至り、ここで VLAN ID 2 が Tag 付与され、20 番ポートへと到達する。つまり 1 番ポートの VLAN ID を変更したことにより、FunctionBox を経由する (機能の提供を受ける) ことになったわけである。

5.3.1 機能増加時の問題点

前節までに、機能の提供や機能の ON/OFF をどのように提供し、それに行うネットワーク機器の制御の仕組について述べた。機能の数が少ない場合は、組合せも少なく前節で試作した GUI でも簡単に操作可能であり、ネットワークもそれほど複雑にはならない。提供する機能数が少ない場合は前説の GUI を拡張することで対応可能である。

しかし機能数が増加するに従って、機能の組合せも増大し、VLAN 構成も複雑になり、。より多くの機能を取り込み、提供できることが望ましいが、それに応えるためには何らかの工夫や仕組が必要になる。次節では、機能数増加に伴う VLAN 構成の複雑化と、それに対処する方法について述べる。

5.4 複数機能の選択機構と論理トポロジー

提供する機能が増えるに伴い、送信端末から FunctionBox を経由し、受信端末に至る VLAN により構築されるパス (以後、「VLAN パス」) は飛躍的に増える。それは単なる機能数の増加だけでなく、機能の組合せや、提供の順といった順列・組合せの数だけ VLAN パスが必要になるからである。以下、例を挙げて説明する。

機能数が一つの場合、当該機能のオン/オフの 2 通りの VLAN パスが必要になる (図 5.2

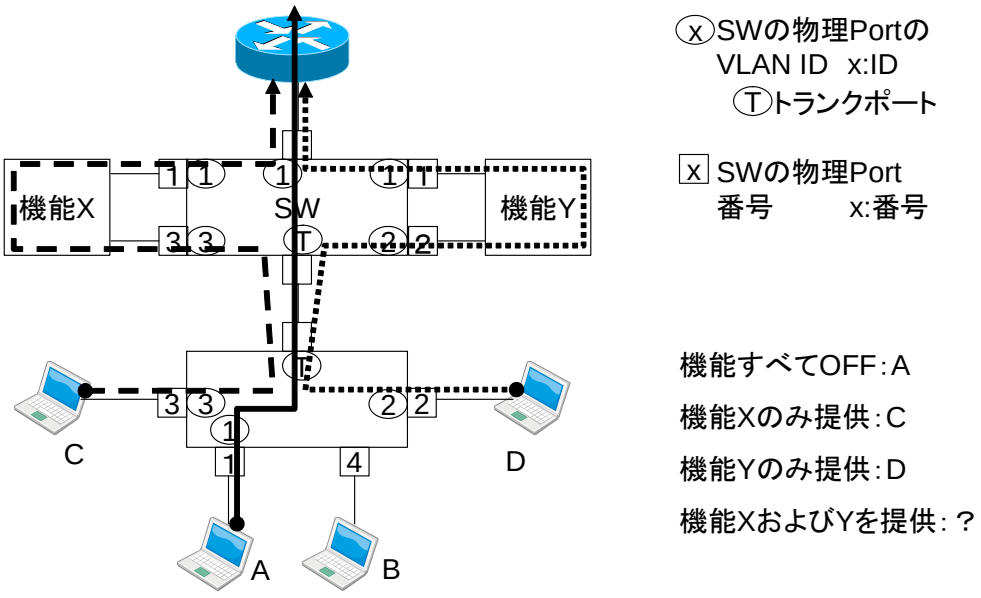


図 5.4: 機能を提供するトポロジー 1

の左右の構成)。

機能が増加し、機能数が二つの場合 (機能名を X、および Y とする)、

- 機能をまったく提供しない
- 機能 X のみ提供する
- 機能 Y のみ提供する
- 機能 X、Y を X → Y の順に提供する
- 機能 X、Y を Y → X の順に提供する

上記 5 つのパターンが存在し、5 つの VLAN パスが必要になる。実際にこれら 5 つのパターンを実現する VLAN パスを図示したのが図 5.4 である。

しかしこの論理ネットワーク構成では、機能 X、Y を提供する VLAN パスが描けていない。次に、機能 X、Y を提供する VLAN パスを構築した論理ネットワーク構成を示す (図 5.5)。

今度は機能 X、Y を提供する VLAN パスは実現できているが、機能 Y のみを提供する VLAN パスが存在しない。また図 1、2 ともに機能 A、Y を Y → X の順で提供する VLAN パスも存在していない。

このような問題点が発生する原因は、一般的に「VLAN のアクセスポートは必ず一つの VLAN に所属する」という制約があるためである。しかしこれは制約というよりはスイッチの VLAN 機能実現のため必要な仕組である。また本論文が採る提案の主旨もスイッチの内部処理機構には手を加えない立場にある。

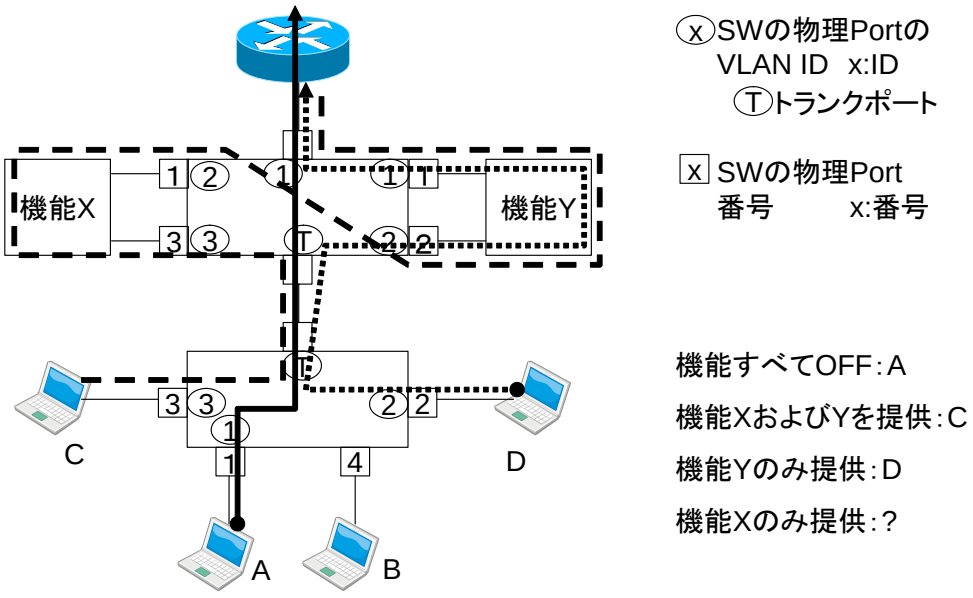


図 5.5: 機能を提供するトポロジー 2

そこで、提供機能が複数存在する場合に順列・組合せを満足する VLAN パスを構築手段として二つの方法が考えられる。

1. FunctionBox の Ethernet インタフェースを増設する
2. FunctionBox とスイッチとの接続ポートをトランクポートにし、FunctionBox の Ethernet インタフェースを VLAN インタフェース化する

1 つ目の方法は物理的な VLAN インタフェースを FunctionBox に増設することで、VLAN パスの増加に対応しようというものである。Ethernet インタフェースの増設による対応を図 5.6 に示す。

確かに Ethernet インタフェースの増設により新たな VLAN パスを追加することは可能になるが、PC に増設できる Eternet インタフェースの数には限界があり、また機能追加にともなう VLAN パスの増加に対応していくには効率が悪い。

次に 2 つめの方法について述べる。まず今まで FunctionBox とスイッチとの接続ポートをアクセスポートとして使用してきたものを、トランクポートとして設定する。そして FunctionBox の Ethernet インタフェースを VLAN インタフェース化することにより物理的なインタフェースの数は変えずにやりとりできる VLAN Tag の数を増加させる方法である。FunctionBox の Ethernet インターフェイスを VLAN インタフェース化して対応する方法を図 5.7 に示す。

Linux PC であれば vconfig コマンドを用いて Ethernt インタフェースに複数の VLAN インタフェースを作成することができる。作成した VLAN インタフェースを bridge インタフェースとして構成することで、単一の FunctionBox を経由する複数の VLAN パスを構成することが可能になる。

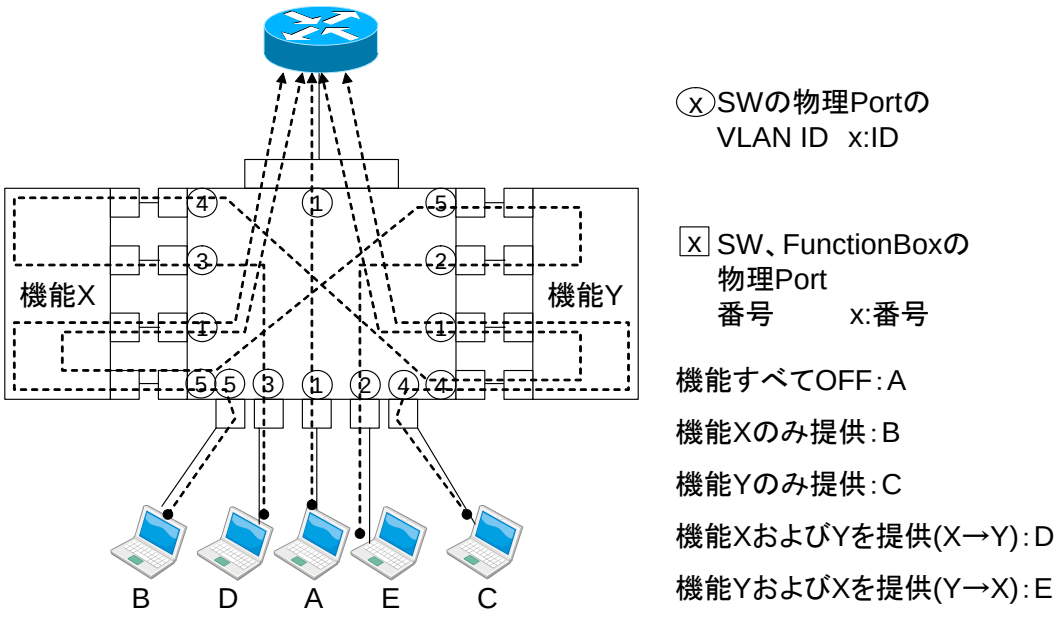


図 5.6: インタフェース増設によるパスの拡張

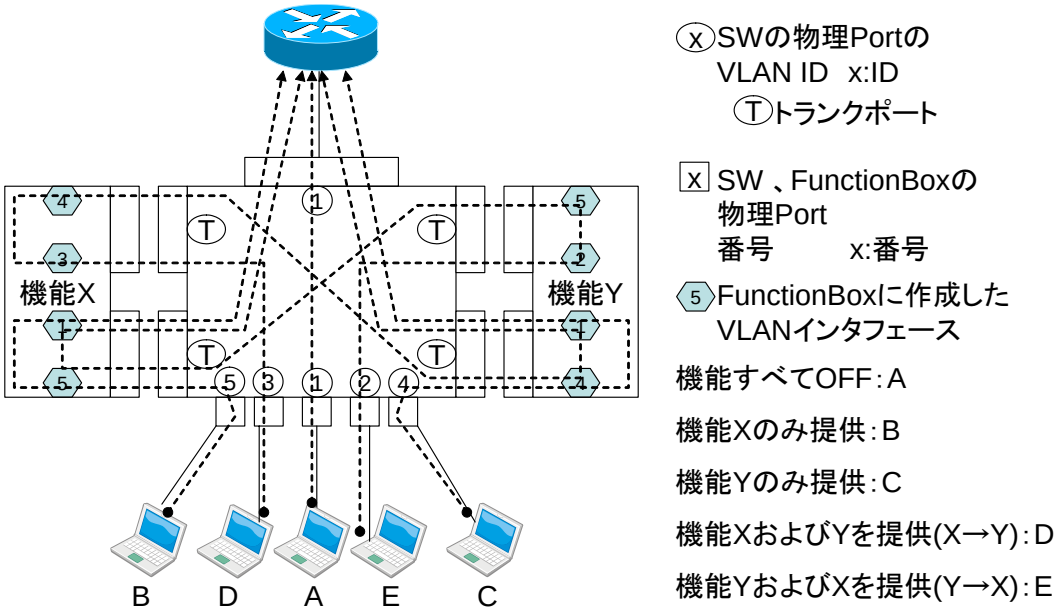


図 5.7: VLAN インタフェースによるパスの拡張

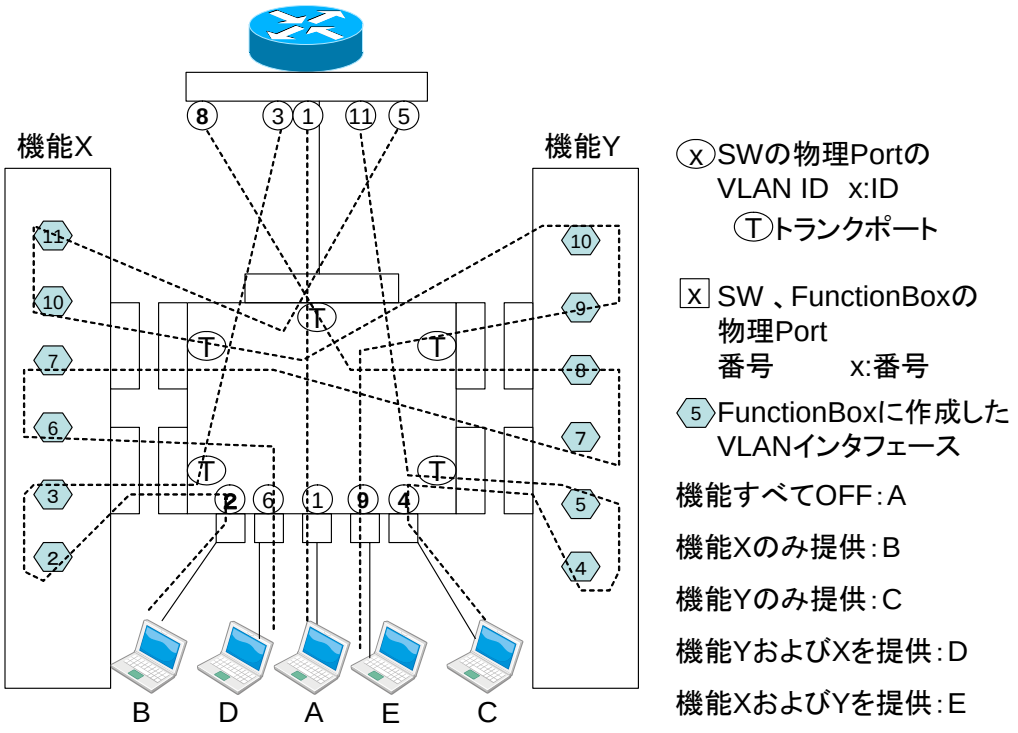


図 5.8: セグメントを分けるパス構築

1 または 2 の方法を採用、組み合わせることにより提供機能が複数になった場合でも自由に機能を使い分ける VLAN パスを構成することは可能になる。しかしながら、機能の数に応じた順列・組合せを満足させる VLAN パスを構築し運用することは、非常に複雑なネットワーク構成を必要とし、運用管理上での問題やミスを発生させる温床ともなる。

また、「VLAN パスの数」は「VLAN ID」の数ではなく、VLAN パスを構築するために使用する VLAN ID の数は VLAN パスの数に比べ飛躍的に増加する。図 5.6 および図 5.7 は最終的にルータに至る経路を重畳して使用し、できるだけ VLAN ID の消費数を節約するよう構成している。しかし、機能の提供形態毎にセグメントを分けて運用したい場合や、ルータ一つではなく出口をパス毎に設けたい場合は、図 5.8 のように計 VLAN ID は 11 個必要になる。

機能の数に伴う VLAN パスの数については前述したように、「機能数: 2」であっても「VLAN パス数:5」が必要になるが、このように、「VLAN パス数:5」を構築するのに要する VLAN ID の数は「VLAN ID 数:11」が必要になる。

この「機能数」「VLAN パス数」「VLAN ID 数」の関係を数式で表したものを以下に示す。

- 提供機能数 n に要する VLAN パスの数 m は、

$$m = \sum_{i=0}^{j} jPi$$

機能数	VLAN パス数	VLAN ID 数
0	1	1
1	2	2
2	5	11
3	16	49
4	65	261
5	326	4511
6	1957	11743
7	13700	95901

表 5.1: 機能数、VLAN パス数、VLAN ID 数の関係

- VLAN パスの数 m を構成するために必要となる VLAN ID の数 l は、

$$l = \sum_{i=0}^{j} jPi(i+1)$$

機能数、VLAN パス数、VLAN ID 数の関係を表にしたものを表 5.1 に示す。

先に、機能数が増加するに従い VLAN パス数が飛躍的に増加し管理が困難になることについて触れた。しかしこの表からは、管理の困難さという難易度の問題だけでない限界値が読み取れる。すなわち、機能数「5」においてこれを満足する VLAN パスを構築するために要する VLAN ID の数が VLAN ID の最大値 4095 を越えることが明らかになっている。つまり、機能数 n において、「すべての機能」を「すべての順列・組合せ」で提供する VLAN パスを構築することの限界が、機能数 4 で打ち止めになることを表している。

「すべての機能」を「すべての順列・組合せ」で提供する VLAN パスを構築することは、非常に複雑な管理困難な論理ネットワーク構成を必要とし、また構成しえたとしても機能数の限界値として 4 を越えられない。

「すべての機能」を「すべての順列・組合せ」で機能を提供できることは、機能を最大限の自由度で提供できることを意味する。しかしながら現実のネットワークシステムに照らした場合、必ずしも「すべての機能」を「すべての順列・組合せ」で提供できる自由度が必要となるわけではない。

例えば、「認証」、「フィルタリング」、「通信ログ蓄積」といった 3 つの機能を考えてみる。これらの機能の運用上の組合せ順として、

1. スイッチに接続したされた端末を「認証」し接続を許可し、
2. ある種類の通信を「フィルタリング」してブロックし、
3. 「通信ログ蓄積」機能でログとして蓄積する、

という機能の適用順は考えられる。しかしながら、機能の性質上「フィルタリング」機能を適用してから端末を「認証」するという使い方をとることは考えにくいし、また、ある

ポリシーに基づき運用されているネットワークが、物理ポート毎に機能の適用順を変えるということも想定しにくい。

よって本論文では、「すべての機能」を「すべての順列・組合せ」で提供するのではなく、追加された機能の中から、必要となる「任意の機能」を「任意の順」で提供する仕組を考案した。

次節では、機能追加の基本アイデアを使い、「任意の機能」を「任意の順」で機能提供するための、VLAN パスを構築する手法について述べる。

5.5 Network Circuit Compiler(NCC)

「フレームの送信端末から受信端末へと至る通信経路上に、ある機能を実装した FunctionBox を配置し、フレームを解釈・処理させる」それが、本論文で提案する機能追加方法の基本アイデアであった。FunctionBox を経由させる仕組が VLAN パスであり、VLAN パスをどのように構成するかによって、提供機能や機能を提供する順番が決まる。

「ネットワーク上に配置した、機能を実現する FunctionBox のうち、必要なものを選び、適用する順に経由するような VLAN パスを構成する」、この思考を表現する手段として、本論文の提案手法 NCC では「式」による表現を採用した。つまり、機能を「関数 (function)」としてとらえ、入力 (例: スイッチのクライアント接続ポート)、および出力 (例: スイッチの Router・GW との接続ポート) を指定した式を記述する。図 5.9 のように、ネットワークに配置された FunctionBox を機能ととらえ、クライアントが接続されたスイッチの物理ポートを入力、Router や GW などのネットワークの出口を出力として表現した場合の「式」の例を以下に示す。

- 認証機能を利用する

  $R = authentication(A)$

- フィルタを使用する

  $R = filter(B)$

- 認証・フィルタを使用する

  $R = authentication(filter(c))$

このような式で記述された、入力に「選択された機能」と「機能の適用順」を解釈し、出口までの VLAN パスを構成するために、ネットワーク機器への制御コマンドを生成するプログラムを本論文による機能追加の提案手法「NCC(Network Circuit Compiler)」と呼ぶ。

NCC は、ネットワーク上に配置された FunctionBox を「部品」として捉え、「入力」、「出力」、「部品」を接続した回路 (circuit) を生成する機構という意味で、Network Circuit Compiler と命名した。

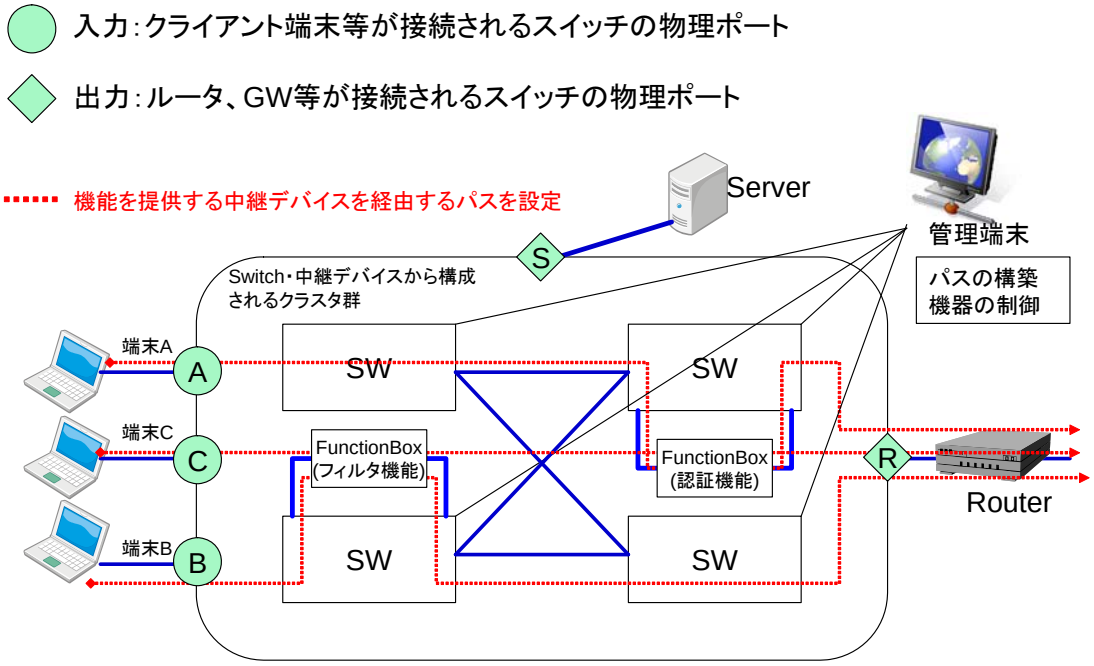


図 5.9: 式による機能提供パスの表現

NCC は、

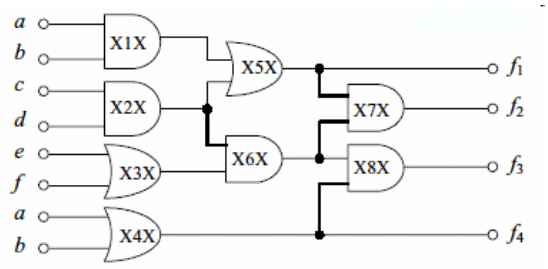
1. NW 上の資源 (スイッチや FunctionBox 等) を記述したファイル (以下、circuit.conf) を読み取り、
2. 「式」で記述された、機能と機能の適用順を字句解析し、
3. VLAN パスを構築するため、ネットワーク機器への制御コマンドを生成する

プログラムである。

NCC の動作概要を図 5.10 に示す。

NCC の設計には、HDL(Hardware Description Language) の考え方を参考にした。HDL はデジタル回路を設計するためのプログラミング言語である (図 5.11)。HDL は文字で電

図 5.10: NCC の動作概要




```
module a684 (a, b, c, d, e, f, f1, f2, f3, f4);

input a, b, c, d, e, f;
output f1, f2, f3, f4;

      assign f1 = a&b|c&d;
      assign f2 = (a&b|c&d)&c&d&(e|f);
      assign f3 = c&d&(e|f)&(a|b);
      assign f4 = a|b;

endmodule
```

図 5.11: HDL による記述表現

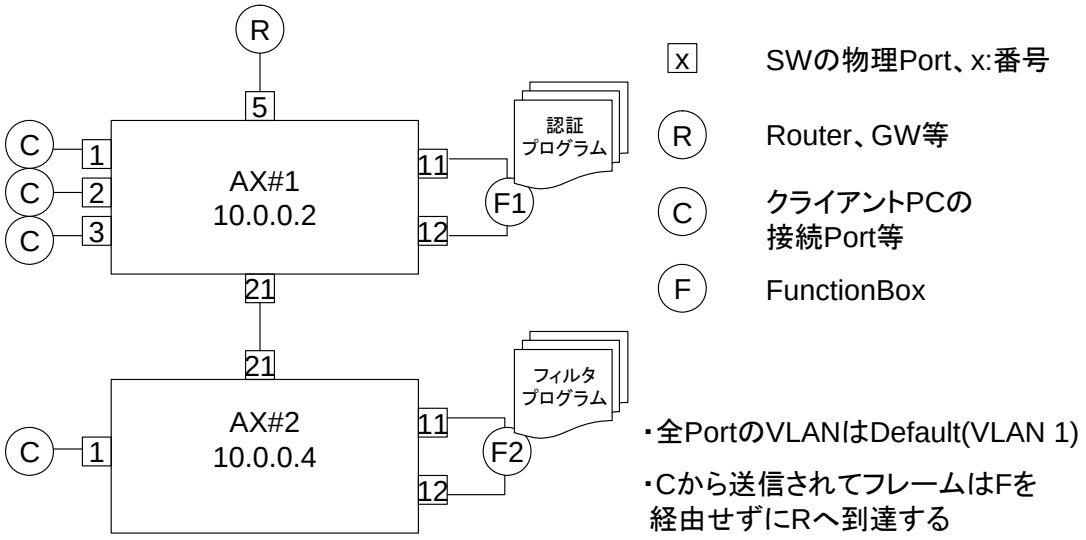


図 5.12: 例に用いる物理ネットワーク構成図

子回路の素子間の接続関係や動作仕様を記述するのに使用される言語である。

NCC が読み込む circuit.conf では、

- 末端の入力をスイッチのクライアント接続ポート、input
- 最終的な出力先を Router や他セグメントへの GW、output
- 機能を実現する FunctionBox を function

として表現した。

NCC は、circuit.conf を読み込み、入出力とその間に経由される function の接続 VLAN で表現 (実現) するため、VLAN パスを形成すたためのネットワーク機器制御コマンドを生成する。

図 5.12 の物理ネットワークを例にあげ、circuit.conf 記述と NCC により構成される論理ネットワーク図 5.13 を例にについて解説する。

次節では、NCC が読み込む circuit.conf について説明する。

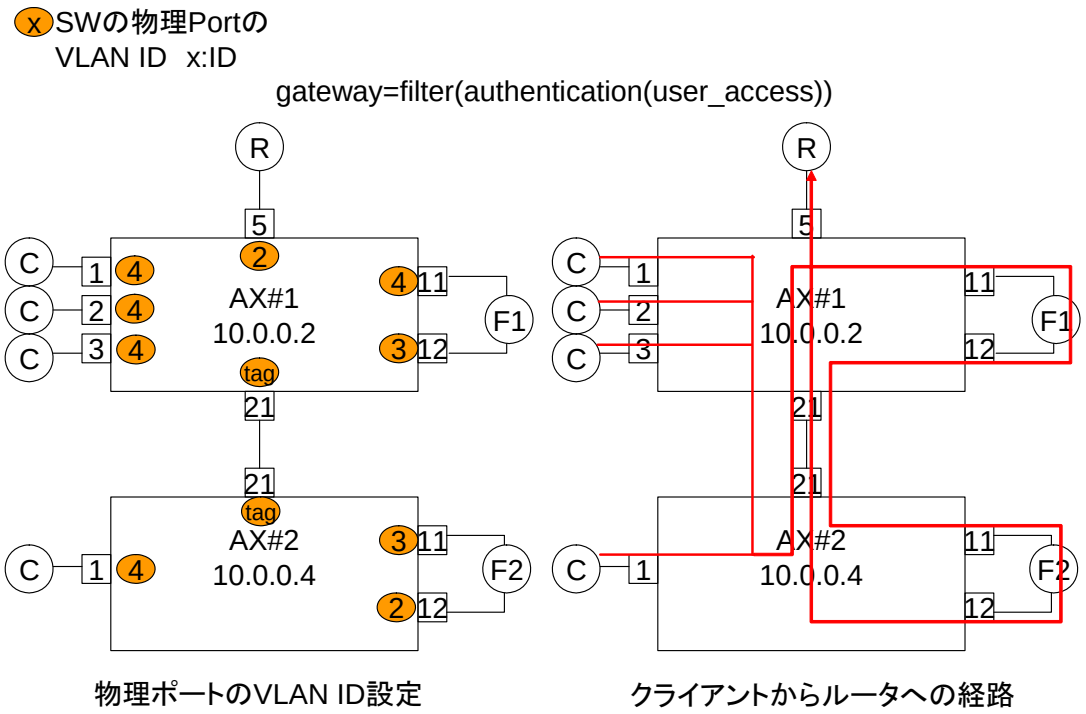


図 5.13: NCC により構築される論理ネットワーク図

5.5.1 circuit.conf

実際に記述される circuit.conf ファイルを以下に掲載する。

```
%curcuit.conf

%物理構成・資源割当記述部
:component

:sw
sw_name:ax_1
sw_addr:10.0.0.2
sw_type:oan
sw_access_port:
port 0/1
port 0/2
port 0/3
port 0/4
sw_trunk_port:
port 0/21
```

```
sw_func_port:
port 0/11
port 0/12
:endsw

:sw
sw_name:ax_2
sw_addr:10.0.0.4
sw_type:oan
sw_access_port:
port 0/1"
port 0/2"
port 0/3"
port 0/4"
sw_trunk_port:
port 0/21
sw_func_port:
port 0/11
port 0/12
:endsw

:endcomponent
```

%機能提供機器記述部

```
:function
```

%認証機能 (hostapd,open1x_authenticator)

```
:function_box
function_name:authentication
function_location:ax_1
input_port:port 0/11
output_port:port 0/12
:endfunction_box
```

%フィルタ機能 (eptables,dps)

```
:function_box
function_name:filter
function_location:ax_2
input_port:port 0/11
```

```
output_port:port 0/12
:endfunction_box
:endfunction

%入力定義部
:input
:input_group
input_group_name:user_access
input_group_member:
ax_1:port 0/1
ax_1:port 0/2
ax_1:port 0/3
ax_2:port 0/1
:endinput_group
:endinput

%出力定義部
:output
:output_group
output_group_name:gateway
output_group_member:
ax_1:port 0/5
:endoutput_group
:endoutput


%経路記述部
:path
:formula
formula_name:Auth and Filter
gateway=filter(authentication(user_access))
:endformula
%:formula
% formula_name:Auth only
% gateway=authentication(user_access)
%:endformula
:endpath
```

circuit.conf ファイルの内容について解説する。NCC はこの circuit.conf を読み込んで、NW 上の資源情報を把握し、式に対応した VLAN パスを構築するため、ネットワーク機器を制御するコマンドを生成する。circutit.conf 内の%で始まる行はコメントとして処理される。

「:compornent」から「:endcompornent」にて記述される「Compornent 部」は、ネットワーク内に存在するスイッチについての情報が記述される。「:sw」から「: endsw」までが一つの物理スイッチについての記述になる。

- sw_name:

  スイッチ名:ネットワーク内部でスイッチを識別するための名称、ID

- sw_addr:

  制御用 IP アドレス:telnet や oan-api で制御を行うための宛先 IP アドレス

- sw_type:

  制御タイプ:当該機器が、ios(Cisco 社の IOS コマンドによる制御) による制御機器なのか、oan(アラクサラネットワークス社の oan-api) による制御機器なのかを登録

- sw_access_port:

  スイッチに実装されている物理ポートのうち、クライアントのアクセスポートとして使用されているポートを記述する。

- sw_trunk_port:

  スイッチに実装されている物理ポートのうち、スイッチ間接続など、トランクポートとして使用されているポートを記述する。

- sw_func_port:

  スイッチに実装されている物理ポートのうち、FunctionBox と接続されているポートについて記述する。

「:function」から「:endfunction」にて記述される「Function 部」は、ネットワーク内に配置された FunctionBox についての情報が記述される。「: function_box」から「:endfunction_box」までが一つの FunctionBox についての記述になる。

- function_name:

  FunctionBox を識別する名称を記述する。機能を関数としてとらえる際の、関数名として使用される。

- function_location:

  当該 FunctionBox が接続されているスイッチの名称について記述される。

- input_port:

  FunctionBox が有する Ethernet インタフェースのうち、(クライアント端末側から見て) 入力側に位置するインタフェースが接続されているスイッチの物理インタフェースについて記述する。

- output_port:

  FunctionBox が有する Ethernet インタフェースのうち、(クライアント端末側から見て) 出力側に位置するインタフェースが接続されているスイッチの物理インタフェースについて記述する。

「:input」から「:endinput」にて記述される「Input 部」は、ネットワーク内に配置されたスイッチの物理ポートを指定して、機能が提供されるグループについての情報が記述される。「:input_group」から「:endinput_group」までが一つの入力インタフェース群についての記述になる。

- input_group_name:

  入力インタフェース群を識別する名称を記述する。式において入力として解釈される。

- input_group_member:

  当該入力インタフェース群に所属するスイッチの物理ポートが記述される。記述にあたっては「スイッチ名:物理ポート」という書式で記述する。

「:output」から「:endoutput」にて記述される「Output 部」は、ネットワーク内に配置されたスイッチの物理ポートを指定して、Router や GW と接続されている物理ポートについてのの情報が記述される。「:output_group」から「: endoutput_group」までが一つの出力インタフェース群についての記述になる。

- output_group_name:

  出力インタフェース群を識別する名称を記述する。式において出力として解釈される。

- output_group_member:

  当該出力インタフェース群に所属するスイッチの物理ポートが記述される。記述にあたっては「スイッチ名:物理ポート」という書式で記述する。

「:path」から「:endpath」にて記述される「Path 部」は、前節にて述べた「式」が記述される。「:formula」から「:endformula」までが一つの式についての記述になる。

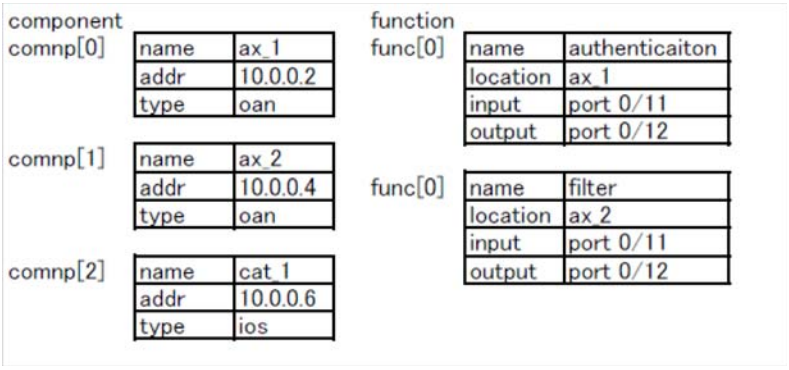


図 5.14: NCC による circuit.conf のメモリへの格納 1

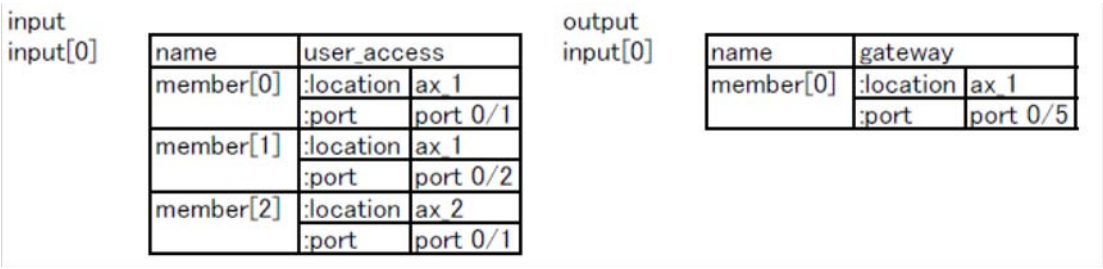


図 5.15: NCC による circuit.conf のメモリへの格納 2

- formula_name:

  式の名称を記述する。そして改行後、式を記述する。

5.6 NCC の実装

NCC の実装について解説する。NCC はプログラミング言語 Ruby を用いて実装した。circuit.conf に記述される「Compornent」、「Function」、「Input」、「Output」の各要素を Ruby の Class で表現し、「sw」、「function_box」、「input_group」、「output_group」にて記述される個別のデータを各々オブジェクトして格納する。各 Class には「sw_name」などの値を格納するメンバと、メンバにアクセスするためのアクセッサが定義されている。

circuit.conf を NCC が読み込んだ後、メモリ上に格納されているデータの様子を図 5.14、図 5.14 に示す。

circuit.conf ファイルの「Compornent 部」から「Output 部」の読み込みが完了すると、「path 部」の解析が行われる。

実装においては、隣合う要素が VLAN パス上でも隣接関係にあることに着目し、字句解析部の実装を行った。

まず、Path 部で与えられる式を要素に分割し配列に格納する。

- 式:

ペア名	VLAN ID
gateway_filter	2
filter_authentication	3
filter_user_acess	4

表 5.2: 要素ペアと VLAN ID のハッシュtable

$gateway = filter(authentication(user\_access))$

- 分解された要素

  gateway、filter、authentication、user_access

- 要素が格納された配列

  word[0] = ”gateway”

  word[1] = ”filter”

  word[2] = ”authentication”

  word[3] = ”user_access”

次に、要素のペアを作成し、で使用する VLAN ID を決定する。これを Hash テーブルとして保持する。ここで NCC は VLAN ID の払出しを行う。「VLAN ID:1」はデフォルト VLAN であるので、デフォルトの設定では「VLAN ID:2」から順次払い出すこととしている。要素ペアと割り当てられた VLAN ID の対応表を表 5.2 以下に示す。

使用する VLAN ID が決定されたことから、次は具体的にスイッチに接続されている各インタフェースに設定すべき VLAN ID を、今度は要素の持つインタフェース毎に登録していく。「Input 部」や「Output 部」はメンバを複数持つ可能性があり、「Function 部」にはクライアント端末に近い、フレームが入力されるインタフェースと、次の機能あるいは出口に出力されるインタフェースとがあるため、ここで個別に VLAN ID が指定される。ポートとポートの所属するペア名、およびポートへ適用する VLAN ID の対応を 5.3 に示す。

これまでの処理で、スイッチに設定されるべき情報、つまり各物理ポートをどの VLAN に所属させれば任意の VLAN パスが構成できるのかという情報が定まった。よって最後に各スイッチに発行すべきコマンドを生成する処理が行われる。

Path 部で入力された今回の式のうち、要素「gateway」を例に処理を説明する。gateway は式のうち左端 (word[0]) に位置するため、出力部として解釈される。メモリ上に格納されたデータから”gateway”がどのスイッチのどの物理ポートに合致するのかを探索し特定する。探索されたデータ「input[0]」の内容を表 5.4 に示す。

今回例ではメンバが 1 つのため、以下の処理が一度だけ行われるが、複数のメンバが存在する場合は、メンバの数だけ処理が行われる。

ポート	ペア名	
gateway	gateway_filter	2
filter_out	gateway_filter	2
filter_in	filter_authentication	3
authentication_out	filter_authentication	3
authentication_in	authentication_user_access	4
user_access	authentication_user_access	4

表 5.3: Pswitch に設定されている Rule

name	gateway	
menber[0]	location:	ax_1
	port:	port 0/5

表 5.4: 検索された input[0] の値

”gateway”のメンバはスイッチ名:ax_1 の物理ポート 5 番が所属しており、ペア名 gateway_filter に割り当てられた「VLAN ID:2」を使用すればよいことが決定され、コマンドが生成される。スイッチ名からスイッチを制御するためのタイプおよび IP アドレスが探索できるため、タイプ毎のコマンドに沿い宛先を指定したコマンド生成が可能となる。”gateway”の設定のために必要となるスイッチ制御用のコマンドはタイプが「oan」のため、以下のように生成されるる。

- VlanControler.jar 10.0.0.2 port 0/5 2

式全体を解析し生成されたコマンドを以下に示す。

```
VlanController.jar 10.0.0.2 "port 0/5" 2
VlanController.jar 10.0.0.4 "port 0/12" 2
VlanController.jar 10.0.0.4 "port 0/11" 3
VlanController.jar 10.0.0.2 "port 0/12" 3
VlanController.jar 10.0.0.2 "port 0/11" 4
VlanController.jar 10.0.0.2 "port 0/1" 4
VlanController.jar 10.0.0.2 "port 0/2" 4
VlanController.jar 10.0.0.2 "port 0/3" 4
VlanController.jar 10.0.0.4 "port 0/1" 4
```

NCC は Path 部において式を入力されることで、VLAN ID を指定されることなく、VLAN ID を「動的」に割り当てながら機能を提供するための VLAN パスを構成する。物理的なネットワーク構成が変化しない限り、Path 部に与える式だけを変更すれば、提供する機能の追加・削除や機能の提供順を任意に変更することができる。クライアントを接続するスイッチの物理ポートと、ネットワークの出入口となる GW を特定し、FunctionBox の物理的な設置場所や Ethernt インタフェースが接続されているポートを特定し、各々に所属させる VLAN ID を設計することは非常に手間がかかる。これを自動化し、機能提供の変更に関わる論理構成の再構築作業を自動化したのが NCC である。

次章では、NCC にいくつかの式を投入し、ネットワークに機能を追加する動作実験や、基本アイデアによる機能追加の性能面についての評価を行う。

第6章　評価

ネットワークに求められる要求事項の多様化により、種々様々な機能追加の必要性が増してきている。前章までで、ネットワーク機器をリプレースして機能追加するのではなく、継続して機能追加可能なアイデアを考案し、それを実現するNCCを提案・試作した。

本章では、具体的にNCCを使用してネットワークへの機能追加を実施し、動作確認を行う。レイヤ2スイッチとホストのみで構成されたネットワークにFunctionBox を導入し、任意の機能を提供するネットワーク構成をNCCにて実現する。

また本論文で提案した手法の性能面での実効値を計測する。本論文の主旨は、「既設のスイッチをリプレースせずに機能追加を実現する」アイデアや手法の提案にある。しかしながら、実効値としてどの程度の数値が示されるのかを参考値として示しておくために測定を行う。容易に予測しうる事項として、VLANによるバイパス回路構成による遅延や、FunctionBoxをインライン配置した際のスループット低下などの影響が考えられる。

本論分の主旨は、ネットワークに機能を追加する手法を提案することにある。しかし、まったく実用途として使用できない手法であっては提案の意味も半減してしまう。よってスループットおよびレイテンシーをSmartBitsによって計測し、計測結果を測定値として本章に付した。

実験による動作や実効値の結果を踏まえて、2章において紹介した他の手法との比較を行い、改めて本論文の提案について検討を行う。

最後の提案システムの拡張案や、残された課題についての考察を行う。

6.1　機能提供ネットワークの構築

NCCの動作実験に使用するネットワークを図6.1に示す。配置されているスイッチはいずれもoan-apiでの制御が可能なアラクサラネットワークス社製スイッチAX2430である。

初期状態では、クライアントC(IPアドレス: 192.168.0.24024)からゲートウェイR(IPアドレス:192.168.0.25424)への通信は、CからAX#2の内部バスを通って物理ポート21番を通り、AX#1の物理ポート5番からRへと送信される。この状態ではスイッチの全ポートがデフォルトVLAN(VLAN ID:1)に所属しており、一つのFDBにより管理されたネットワークになっている。よって、FunctionBox_F1やFunctionBox_F2を経由せずに通信が行われる。

この動作実験では、NCCによる制御により、意図した機能を追加するためのVLANパスが構築され、期待動作するかを検証する。つまり、正しく記述されたcircuit.confを読

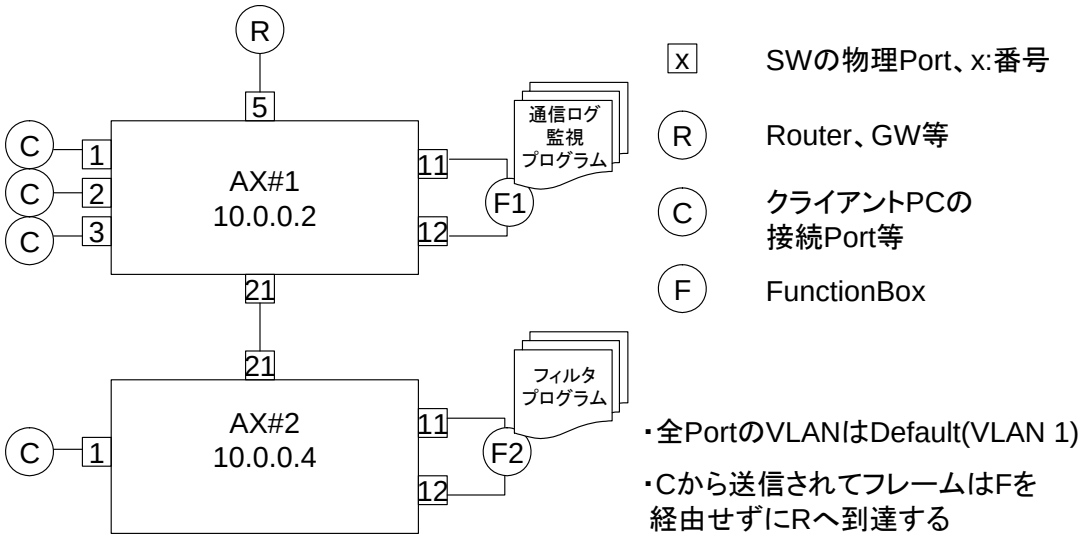


図 6.1: 実験に用いる物理ネットワーク

み込んだ NCC に、様々な式を投入し、意図した機能追加が実現しているかを検証する。初期状態ではどの FunctionBox も経由しないが、式により機能追加を指示すれば、当該機能に対応する FunctionBox を経由する VLAN パスが構成される。また機能を順番に指定してすることで、意図した順に意図した機能を提供する FunctionBox を順に経由するような VLAN パスが構成される。

次に、動作実験にて使用する「追加機能」について説明する。今回の実験で追加する機能と機能の内容、および実装に使用したプログラムについて説明する。

1. フィルタリング機能
   FunctionBox_F1 に実装。機能名を「filter」とした。FunctionBox の Ethernet インタフェースをブリッジ化し、フレームの送受信を行う。受信したフレームの MAC アドレスが登録しておいた送信元 MAC アドレスに合致した場合、当該フレームを破棄する。Ethernt インタフェースのブリッジ化には bridge-utils を、MAC アドレスフィルタリングには ebtables を用いた。
2. 通信ログ蓄積機能
   FunctionBox_F2 に実装。機能名を「logging」とした。FunctionBox の Ethernet インタフェースをブリッジ化し、フレームの送受信を行う。受信したフレームのペイロードを確認し、ICMP を監視するアプリケーションを実装した。tcpdump を起動し、プロトコル番号が 1 に設定されているフレームを受信すると、送信元をファイルに記録する。

動作実験のシナリオを以下に示す。circuit.conf ファイルは前章のものを一部改編したファイルを NCC に読み込ませる。そして Path 部に登録する式を変更し、投入した式の意図する VLAN パスが構築され、機能が適用されるかを調べる。

投入する式と期待動作を以下に示す。

1. gateway=filter(user_access)
   user_access に指定した物理ポートに接続されたクライアント端末が送信するフレームにレイヤ 2 フィルタを適用する。
2. gateway=logging(user_access)
   user_access に指定した物理ポートに接続されたクライアント端末が送信するフレームを監視し、ICMP パケットをファイルに記録する。
3. gateway=logging(filter(user_access))
   user_access に指定した物理ポートに接続されたクライアント端末が送信するフレームにレイヤ 2 フィルタを適用し、ser_access に指定した物理ポートに接続されたクライアント端末が送信するフレームを監視し、ICMP パケットをファイルに記録する。(「フィルタ」→「通信ログ監視」の順に機能を適用する)
4. gateway=logging(filter(user_access))
   user_access に指定した物理ポートに接続されたクライアント端末が送信するフレームにレイヤ 2 フィルタを適用し、user_access に指定した物理ポートに接続されたクライアント端末が送信するフレームを監視し、ICMP パケットをファイルに記録する。(「フィルタ」→「通信ログ監視」の順に適用を適用する)

6.1.1 フィルタ機能の追加

FunctionBox_F1 には、今回クライアント端末として使用する PC の Ethernt インタフェースの MAC アドレスが、既にフィルタとして登録されている。NCC によるネットワーク構築を以前に、user_access として指定された物理ポートに接続した PC から R への Ping を発行すると、Reply が正常に返された。

次に、circuit.conf ファイルの Path 部に式、
”gateway=filter(user_access)” を記述し、NCC を起動する。NCC の起動により生成された制御コマンドを以下に、形成されたネットワークを図 6.2 に示す。

```
VlanController.jar 10.0.0.2 "port 0/5" 2‘
VlanController.jar 10.0.0.4 "port 0/12" 2‘
VlanController.jar 10.0.0.4 "port 0/11" 3‘
VlanController.jar 10.0.0.2 "port 0/1" 3‘
VlanController.jar 10.0.0.2 "port 0/2" 3‘
VlanController.jar 10.0.0.2 "port 0/3" 3‘
VlanController.jar 10.0.0.4 "port 0/1" 3‘
```

---

user_access として指定された物理ポートに接続した PC から R への Ping を発行すると、Reply は返ってこなかった。別の PC を当該物理ポートに接続して接続しての Ping を発行すると、Reply が正常に返された。これにより NCC が期待動作を実現するネットワークを構築していることが確認された。

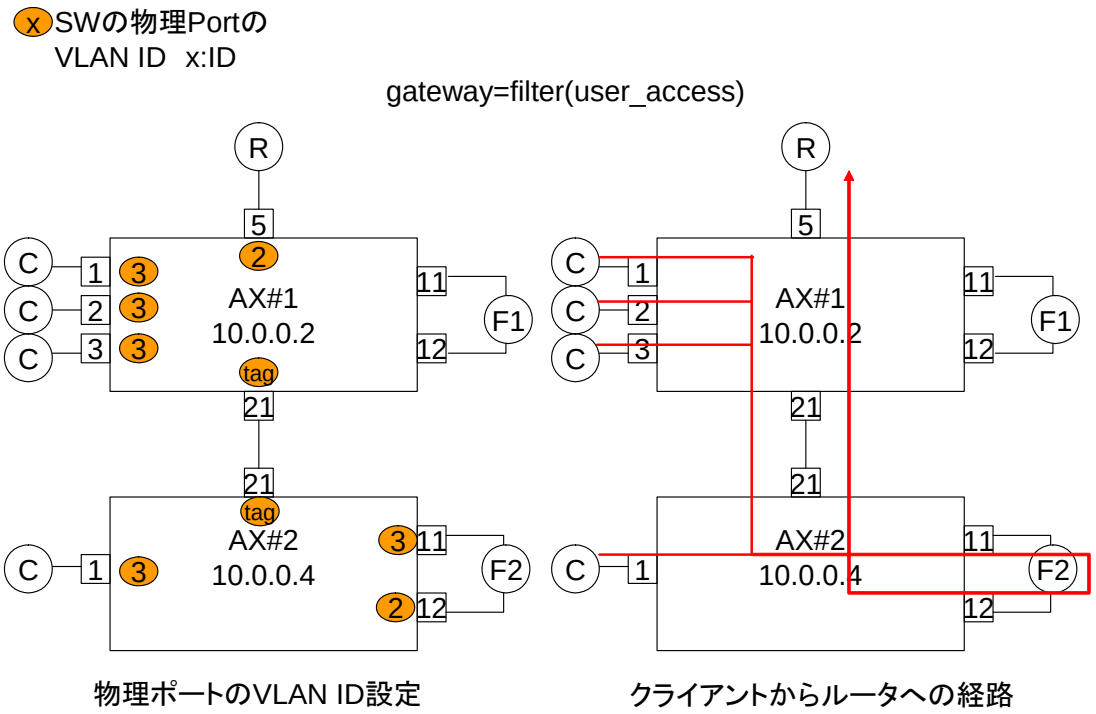


図 6.2: フィルタ機能を提供する論理トポロジー

6.1.2 通信ログ鑑識機能の追加

FunctionBox_F2 では tcpdump が起動されており、プロトコル番号が 1 に設定されているフレームを受信すると、送信元をファイルに記録するプログラムが動作している。NCC によるネットワーク構築を以前に、user_access として指定された物理ポートに接続した PC から R への Ping を発行し、FunctionBox_F2 において ICMP を記録するファイルを確認すると、何も記録されていないことを確認した。

次に、circuit.conf ファイルの Path 部に式”gateway=logging(user_access)”を記述し、NCC を起動する。NCC の起動により生成された制御コマンドを以下に、形成されたネットワークを図 6.3 に示す。

```
VlanController.jar 10.0.0.2 "port 0/5" 2‘
VlanController.jar 10.0.0.2 "port 0/12" 2‘
VlanController.jar 10.0.0.2 "port 0/11" 3‘
VlanController.jar 10.0.0.2 "port 0/1" 3‘
VlanController.jar 10.0.0.2 "port 0/2" 3‘
VlanController.jar 10.0.0.2 "port 0/3" 3‘
VlanController.jar 10.0.0.4 "port 0/1" 3‘
```

---

user_access として指定された物理ポートに接続した PC から R への Ping を発行し、FunctionBox_F2 において ICMP を記録するファイルを確認すると、PC から送信された ICMP パケットおよび、R からの変身が記録されていることを確認した。これにより NCC が期待動作を実現するネットワークを構築していることが確認された。

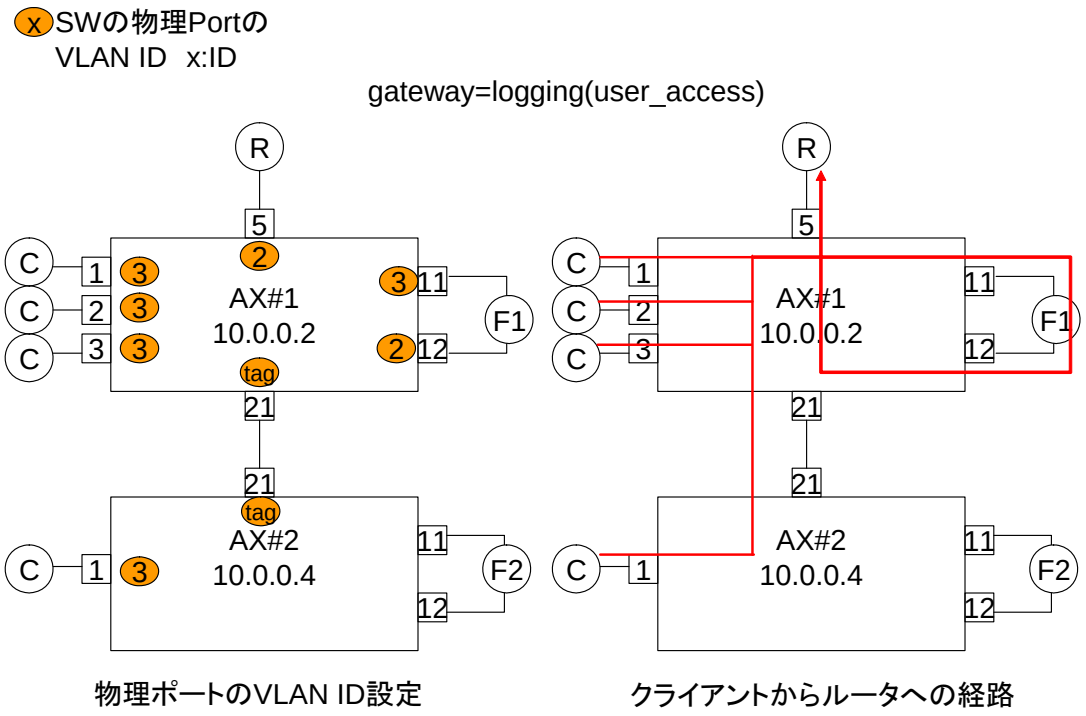


図 6.3: 通信ログ監視機能を提供するトポロジー

6.1.3 フィルタ機能および通信ログ監視機能の追加

FunctionBox_F1 に実装されたフィルタリング機能と、FunctionBox_F2 に実装された通信ログ監視機能が指定した順に適用され動作するネットワークが構築されているかの動作実験を行った。circuit.conf ファイルの Path 部に式”gateway=logging(filter(user_access))”を記述し、NCC を起動する。NCC の起動により生成された制御コマンドを以下に、形成されたネットワークを図 6.4 に示す。

```
VlanController.jar 10.0.0.2 "port 0/5" 2‘
VlanController.jar 10.0.0.4 "port 0/12" 2‘
VlanController.jar 10.0.0.4 "port 0/11" 3‘
VlanController.jar 10.0.0.2 "port 0/12" 3‘
VlanController.jar 10.0.0.2 "port 0/11" 4‘
VlanController.jar 10.0.0.2 "port 0/1" 4‘
VlanController.jar 10.0.0.2 "port 0/2" 4‘
VlanController.jar 10.0.0.2 "port 0/3" 4‘
VlanController.jar 10.0.0.4 "port 0/1" 4‘
```

---

user_access として指定された物理ポートに接続した PC から R への Ping を発行し、FunctionBox_F2 において ICMP を記録するファイルを確認すると、何も記録されていないことを確認した。これは先にフィルタリング機能が適用され、PC の送信フレームが破棄されたためである。次に別の PC を当該物理ポートに接続して接続しての Ping を発行すると Reply が正常に返され、かつ記録ファイルに ICMP の通信記録が記録されていることが確認できた。これにより NCC が期待動作を実現するネットワークを構築している

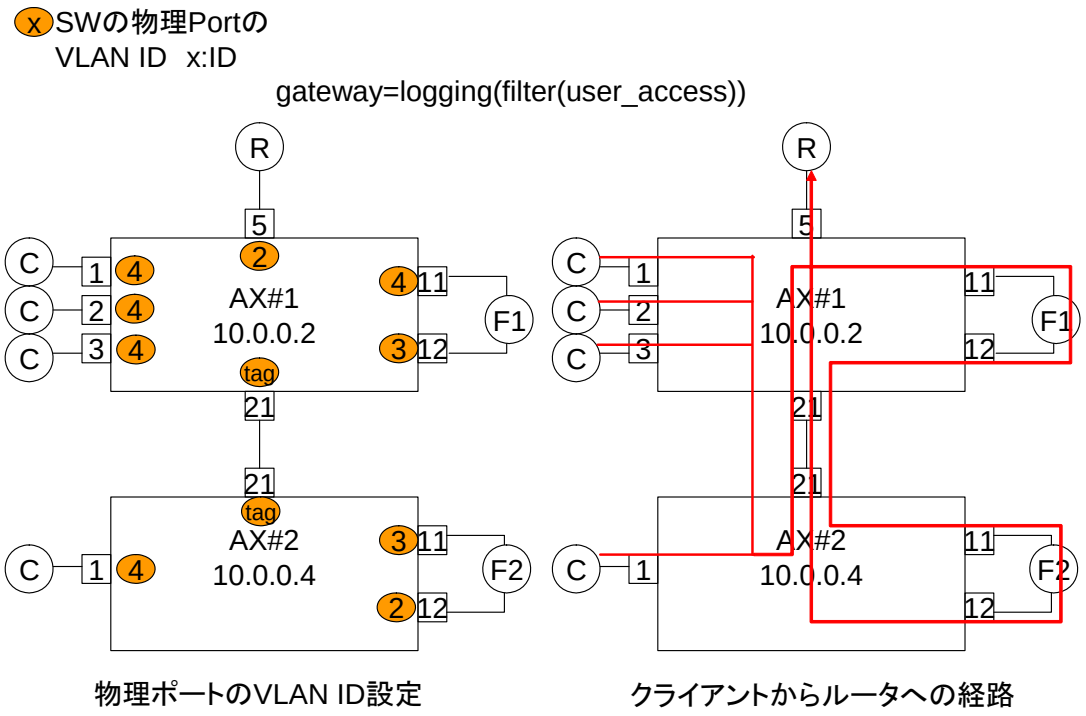


図 6.4: 通信ログ監視、フィルタの順で機能を提供する論理トポロジー

ことが確認された

次に、circuit.conf ファイルの Path 部に式”gateway=filter(logging(user_access))” を記述し、NCC を起動 NCC の起動により生成された制御コマンドを以下に、形成されたネットワークを図 6.5 に示す。

```
VlanController.jar 10.0.0.2 "port 0/5" 2‘
VlanController.jar 10.0.0.2 "port 0/12" 2‘
VlanController.jar 10.0.0.2 "port 0/11" 3‘
VlanController.jar 10.0.0.4 "port 0/12" 3‘
VlanController.jar 10.0.0.4 "port 0/11" 4‘
VlanController.jar 10.0.0.2 "port 0/1" 4‘
VlanController.jar 10.0.0.2 "port 0/2" 4‘
VlanController.jar 10.0.0.2 "port 0/3" 4‘
VlanController.jar 10.0.0.4 "port 0/1" 4‘
```

user_access として指定された物理ポートに接続した PC から R への Ping を発行すると、PC から送信した ICMP は記録されているが、R からの Reply はなく、またファイルに記録がないことが確認できた。。これにより NCC が期待動作を実現するネットワークを構築していることが確認された。

6.2 定量的評価-スループットおよびレイテンシーの測定-

前節において、NCC の動作について実験を行い、期待動作することが確認できた。次に性能面での実効値を測定する。

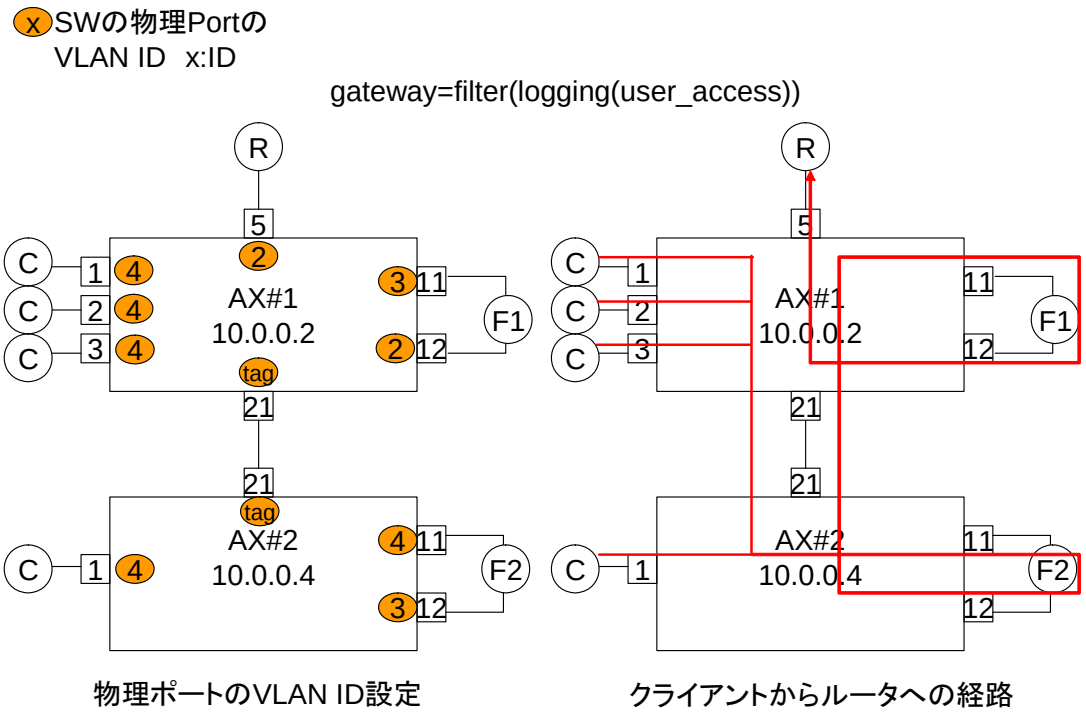


図 6.5: フィルタ、通信ログ監視の順で機能を提供する論理トポロジー

本論分の基本アイデアによる機能追加は、バイパス経路の構築と、そこに設置される FunctionBox によって実現される。スイッチの内部バスを経由する通信を、いったん外に出して処理してから、再びスイッチに戻す構成をとるため、スイッチの内部バスを経由する通信に比べ、オーバーヘッドは増加する。全て Gigabit Ethernet インタフェースを有する機器で VLAN パスを構築したとしても、PC で実現する FunctionBox の性能によりスループットは低下および、経由スイッチの増加による遅延時間 (レイテンシー) も発生する。よって、実際に計測することで、これらの要素がどの程度の値として計測されるのかを実験した。測定には SmartBits を用い、項目として「スループット」および「レイテンシー」を測定した。また計測された結果が実用においてどの程度影響を与えるのかを、Ping による Round-trip-time の値として示し、考察をおこなった。

FunctionBox として使用した PC の仕様を以下に示す。

- CPU AMD Opteron2220 2.8GHz
- Memory 8Gbyte
- GigabitEthernet × 2

スイッチの物理インタフェース、SmartBits インタフェースともに Gigabit Ethernet インタフェースを用いてネットワークを構築した。

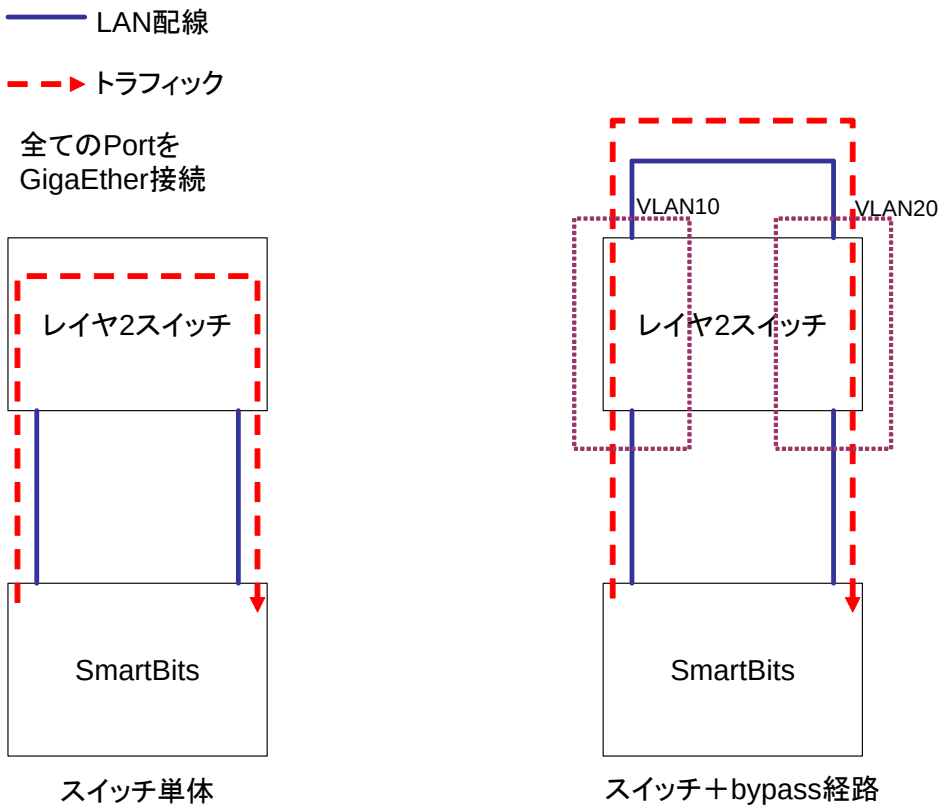


図 6.6: スループット測定対象トポロジー 1

6.2.1 スループットの測定

SmartBits による測定実験を行ったネットワーク構成を図 6.6、図 6.7 に示す。
計測は、以下の項目を対象として行った。

1. スイッチ単体
2. スイッチを VLAN で分割し、patch ケーブルでバイパス経路を構築
3. FunctionBox を Bridge として使用
4. スイッチを VLAN で分割し、バイパス経路上に FunctionBox を Bridge として配置
5. 4 の FunctionBox にレイヤ 2 フィルタを 1 行定義
6. 4 の FunctionBox にレイヤ 2 フィルタを 10 行定義
7. 4 の FunctionBox にレイヤ 2 フィルタを 100 行定義
8. 4 の FunctionBox にレイヤ 2 フィルタを 1000 行定義

まず図 6.6 左側の構成で、スイッチ単体のスループットを計測する (1)。次に図 6.6 右側の構成で、VLAN により論理的に分割したスイッチをパッチケーブルで接続し、FunctionBox を設置しないバイパス経路のスループットを計測する (2)。1 と 2 では SmartBits から流すトラフィックは等量であってもスイッチが処理するトラフィックは 2 倍になる。その影

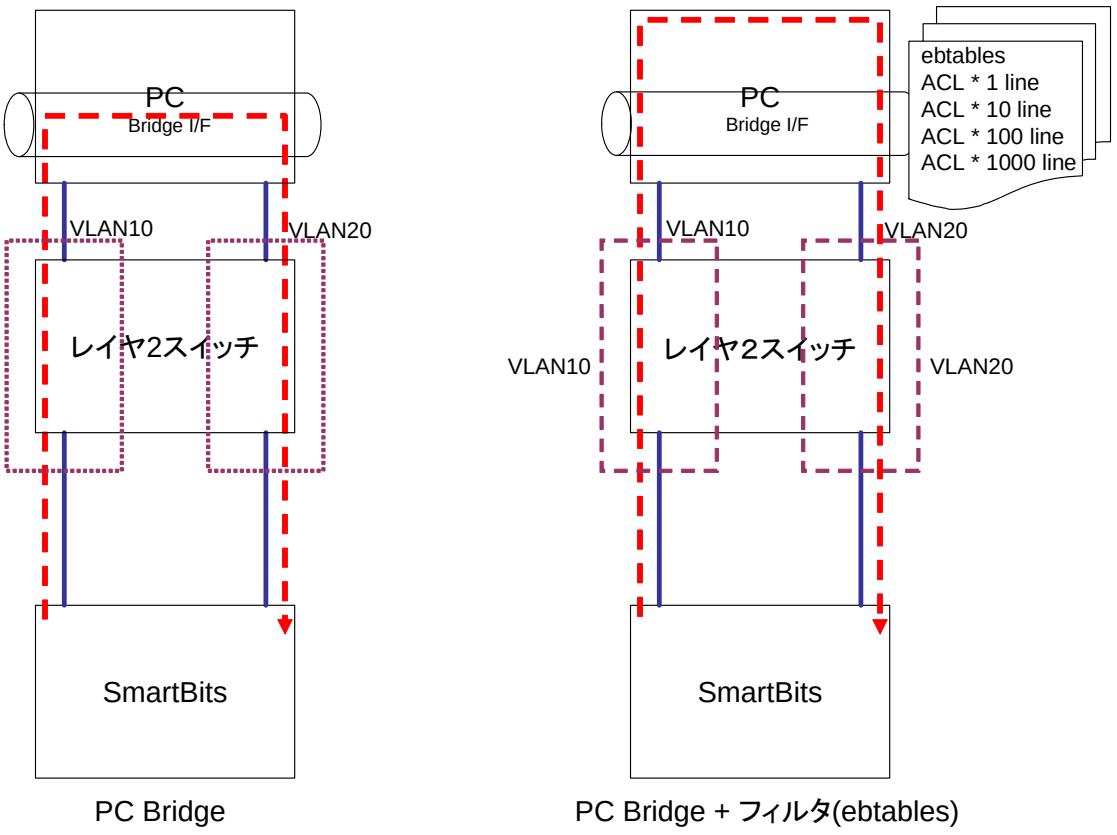


図 6.7: スループット測定対象トポロジー 2

表 6.1: フレームサイズ (byte) 毎のスループット (%)

	(1) スイッチ単体	(2) スイッチ+バイパス	(3)PC ブリッジ	(4) スイッチ+PC ブリッジ
64	100.00	100.00	56.00	56.00
128	100.00	100.00	77.08	77.08
256	100.00	100.00	100.00	100.00
512	100.00	100.00	100.00	100.00
1024	100.00	100.00	100.00	100.00
1280	100.00	100.00	100.00	100.00
1518	100.00	100.00	100.00	100.00

表 6.2: フィルタ設定数とフレームサイズ (byte) 毎のスループット (%)

	(1)1 行	(2)10 行	(3)100 行	(4)1000 行
64	53.16	42.42	8.68	3.10
128	77.08	72.55	16.16	5.60
256	99.28	99.28	30.60	11.22
512	100.00	100.00	68.56	22.47
1024	100.00	100.00	100.00	43.72
1280	100.00	100.00	100.00	54.35
1518	100.00	100.00	100.00	64.41

響がスループットに現れるかを見る。

続いて FunctionBox を Bridge として動作させた際のスループットを測定する (3)。FunctionBox として使用する PC やブリッジに使用するアプリケーション等によって値は上下するが、今回使用した機器においての値を計測する。次に図 6.7 左側の構成を構築し、バイパス経路上にブリッジとして動作させる FunctionBox を設置し、スループットを計測する (4)。この 4 の構成が、バイパス経路+FunctionBox による基本アイデアにより作成される構成であり、FunctionBox 上でフレームを転送する目安のスループットになる。この FunctionBox 上でレイヤ 2 フィルタを設定し、設定数を 1/10/100/1000 と増加させていき、処理負荷によるスループットへの影響を観測するのが (5)(6)(7)(8) になる。

SmartBits により計測したスループット値を表 6.1 に示す (100%=1000Mbps)。

上記の結果をグラフ (図 6.8) として示す。スイッチ単体、および VLAN 間のパッチケーブル接続は、いずれのフレームサイズにおいても 1Gbps を計測しており、ワイヤレートの値が出ている。PC をブリッジとして使用した場合、ショートフレームにおいては 560Mbps にまで落ち込むが、256byte 以上のフレームサイズでは、1Gbps を記録している。バイパス経路上に PC Bridge を配置した場合も PC ブリッジの性能値がそのまま (4) の値になっている。これらの測定値から、スループットは PC ブリッジの性能によって決まり、スイッチの論理分割やバイパス経路はスループットに影響していない。

次にフィルタ設定数毎のスループット値を見る。PC Bridge にフィルタを設定し、設定したフィルタ数によるスループットへの影響を計測した値を表 6.2 に示す。

上記の結果をグラフ (図 6.9) として示す。

ショートフレーム性能はフィルタを 100 行設定したあたりで 100Mbps 程度にまで落ち

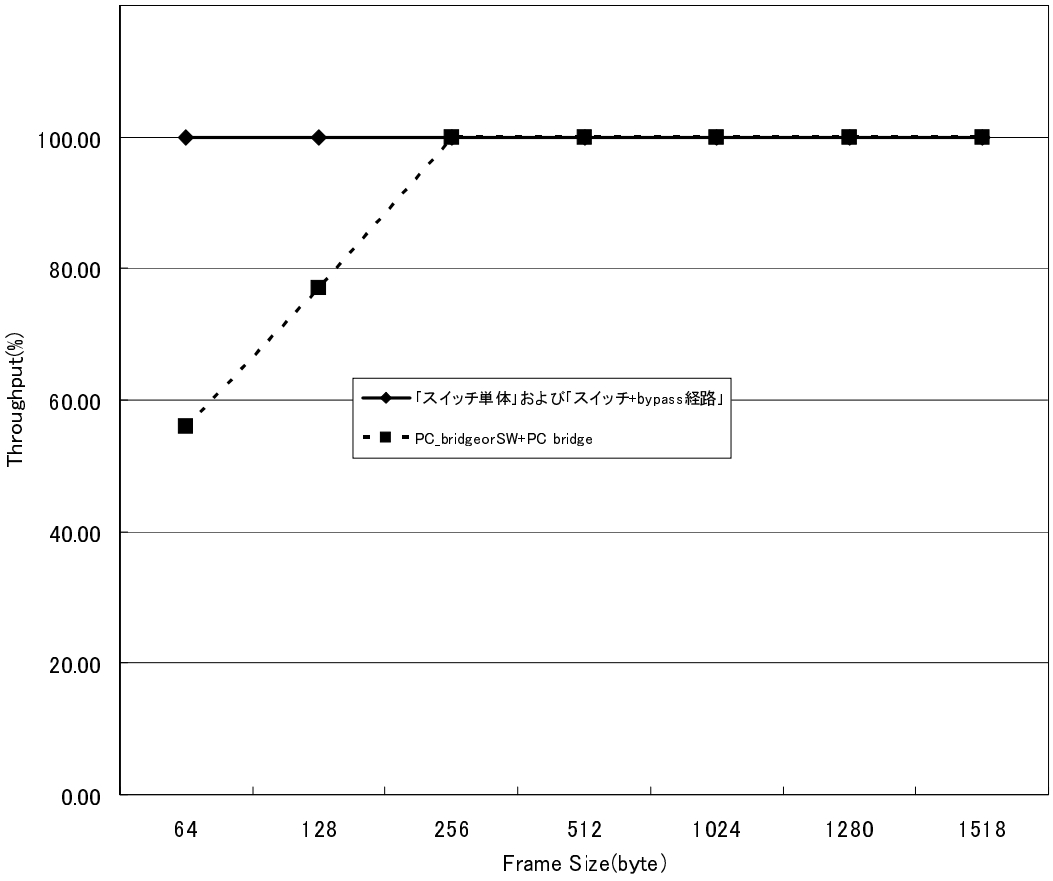


図 6.8: スループット測定値のグラフ

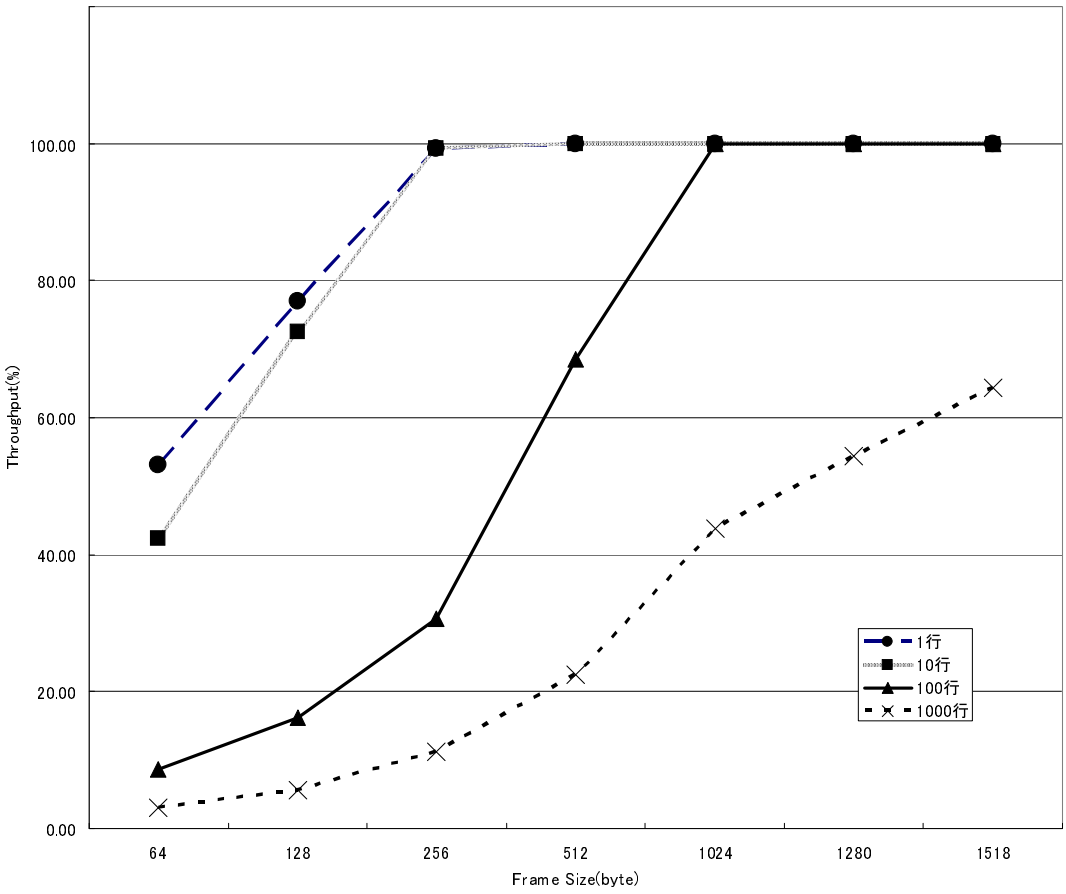


図 6.9: フィルタ設定数ごとのスループット測定値のグラフ

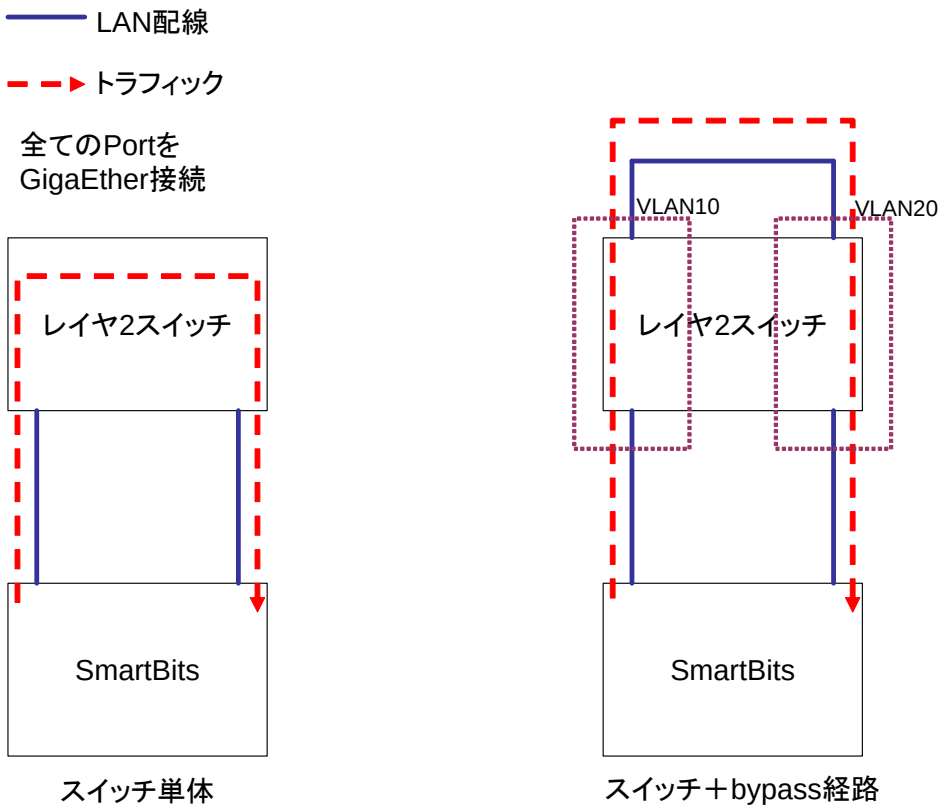


図 6.10: レイテンシー測定対象トポロジー 1

る。しかし 1024byte 超のロングフレームにおいては 1000 行設定しても 500Mbps 程度を計測している。

6.2.2 レイテンシーの測定

SmartBits による測定実験を行ったネットワーク構成を図 6.10、図 6.11 に示す。
計測は、以下を対象として行った。

1. スイッチ単体
2. スイッチを VLAN で分割し、patch ケーブルでバイパス経路を構築
3. FunctionBox をブリッジとして使用
4. スイッチを VLAN で分割し、バイパス経路上に FunctionBox をブリッジとして配置
5. 4 の FunctionBox にレイヤ 2 フィルタを 1 行定義
6. 4 の FunctionBox にレイヤ 2 フィルタを 10 行定義
7. 4 の FunctionBox にレイヤ 2 フィルタを 100 行定義
8. 4 の FunctionBox にレイヤ 2 フィルタを 1000 行定義

まず図 6.10 左側の構成で、スイッチ単体のレイテンシーを計測する (1)。次に図 6.10 右

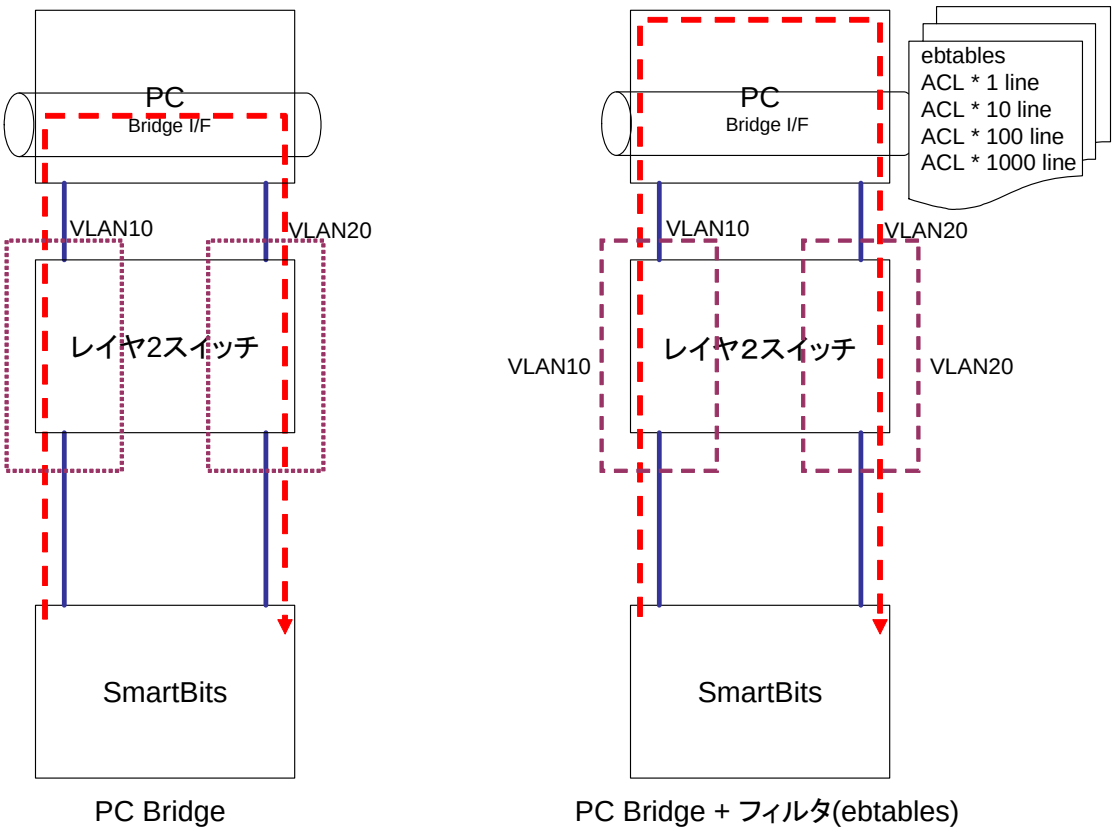


図 6.11: レイテンシー測定対象トポロジー 2

表 6.3: フレームサイズ (byte) 毎のレイテンシー (ns)

	(1) スイッチ単体	(2) スイッチ+バイパス	(3)PC ブリッジ	(4) スイッチ+PC ブリッジ
64	4.5	8.6		
128	5.0	9.6		
256	6.0	11.6		
512	8.0	15.7	81.5	87.3
1024	12.1	23.9	97.7	113.7
1280	14.3	27.9	112.5	130.4
1518	16.1	31.9	122.5	145.2

側の構成で、VLAN により論理的に分割したスイッチをパッチケーブルで接続し、FunctionBox を設置しないバイパス経路のレイテンシーを計測する (2)。1 と 2 では SmartBits から流すトラフィックは等量であってもスイッチが処理するトラフィックは 2 倍になる。その影響がレイテンシーに現れるかを見る。

続いて FunctionBox をブリッジとして動作させた際のレイテンシーを測定する (3)。FunctionBox として使用する PC やブリッジに使用するアプリケーション等によって値は上下するが、今回使用した機器においての値を計測する。次に図 6.11 左側の構成を構築し、バイパス経路上にブリッジとして動作させる FunctionBox を設置し、レイテンシーを計測する (4)。この 4 の構成が、バイパス経路+FunctionBox による基本アイデアにより作成される構成であり、FunctionBox 上でフレームを転送する目安のレイテンシーになる。この FunctionBox 上でレイヤ 2 フィルタを設定し、設定数を 1/10/100/1000 と増加させていき、処理負荷によるレイテンシーへの影響を観測するのが (5)(6)(7)(8) になる。

SmartBits により計測したレイテンシーを表 6.3 に示す。以下に挙げる測定値について補足する。レイテンシーの測定値は SmartBits の結果表示中、カットスルーモードでの平均値を掲載した。表中の数値の単位は n(ナノ)sec である。また、スループットの計測値にて示したように、PC ブリッジの性能限界として、256byte 以下のフレームサイズについてはパケットロスが生じる (1Gbps までスループットがでない) ため、PC Bridge を使用する測定 (フィルタも含む) については、512byte 以上のフレームサイズにてレイテンシー計測を行った (フィルタを 100 行設定したものは 1024byte 以上)。フィルタを 1000 行設定したものについては 1518byte においてもワイヤレート負荷で計測できない (644Mbps) ため、レイテンシー計測は行っていない。

上記の結果をグラフ (図 6.12) として示す。スイッチ単体のスイッチング処理は非常に高速であり、64byte のショートフレームを 5nsec 以下で処理している。VLAN 分割してバイパスした際には同じフレームを 2 回処理することになるので、単純に約倍の時間がかかる処理となっている。PC Bridge と組合せた場合は、PC-ブリッジの性能限界にスイッチの処理時間が加算された値がおよそ合算値に近い値として計測されている。

次にフィルタ設定数毎のレイテンシー値を見る。PC ブリッジにフィルタを設定し、設定したフィルタ数によるレイテンシーへの影響を計測した値を表 6.4 に示す。

上記の結果をグラフ (図 6.13) として示す。フィルタの追加に伴い低下していくスルー

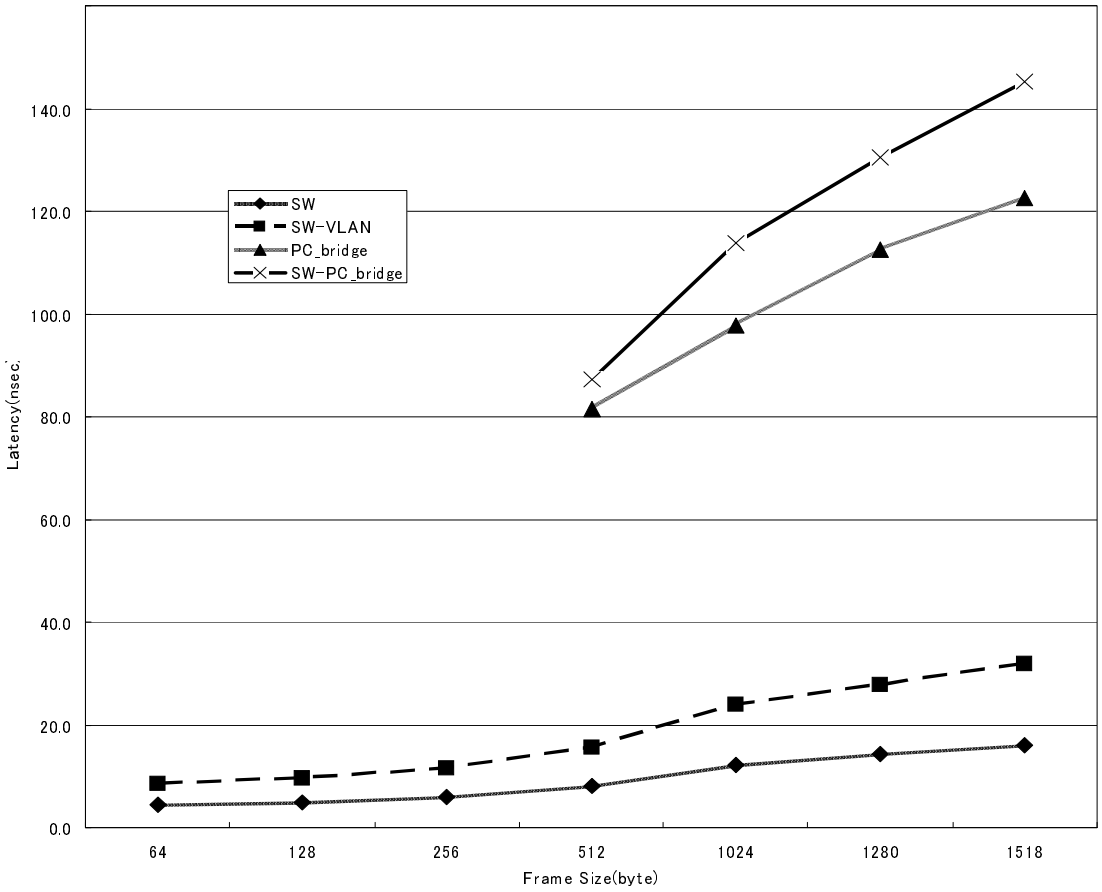


図 6.12: レイテンシー測定値のグラフ

表 6.4: フィルタ設定数とフレームサイズ (byte) 毎のレイテンシー (ns)

	(1)1 行	(2)10 行	(3)100 行
512	88.2	88.7	
1024	115.6	117.2	116.7
1280	130.2	131.9	134.0
1518	146.6	146.8	148.5

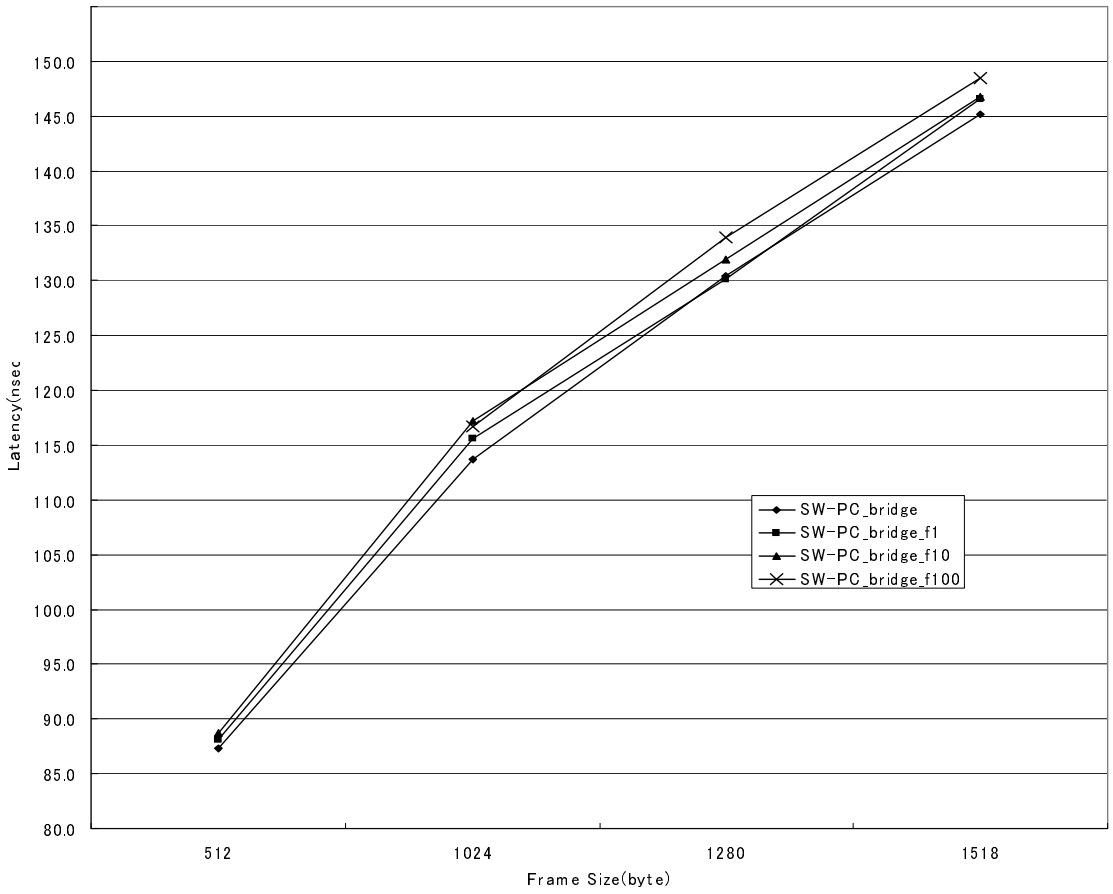


図 6.13: フィルタ設定数ごとのレイテンシー測定値のグラフ

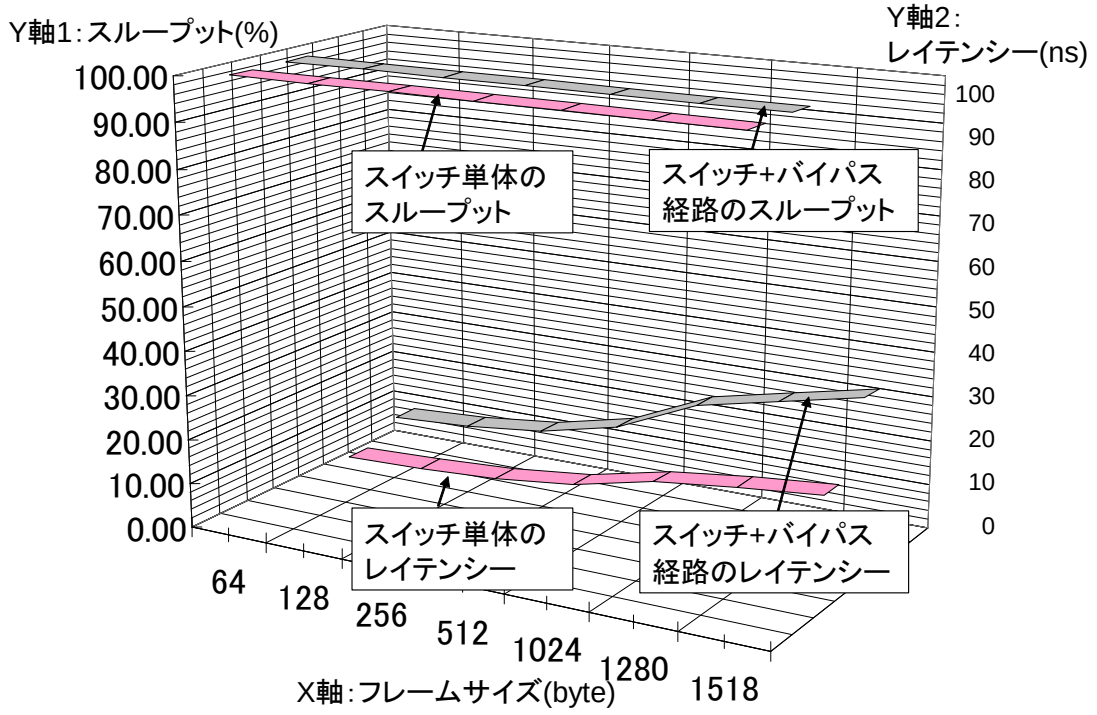


図 6.14: スイッチ単体、およびバイパス経路構成の比較

プット値に比して、レイテンシーはフィルタ設定数に従い増加するものの、フィルタ設定数間での差がほぼ無い。メモリの制約が厳しいネットワーク機器に比べ、PC であれば搭載メモリを増強した分だけフィルタの設定数を増やすことができる。

6.2.3 スループットおよびレイテンシーについての考察

FunctionBox に用いる機器の性能によりスループットおよびレイテンシーの値は変化する。よって、FunctionBox をバイパス経路上に設置しない状態での比較結果だけをまとめたものをグラフ (図 6.14) として示す。

スループットは 1Gbps を維持しており、スループットへの影響はないことがグラフからわかる。一方レイテンシーはスイッチ単体時に約 15ns であったものがバイパス経路構成時には倍の 30ns へ増加している。これは単純にスイッチを二回経由するためであると考えられる。

6.2.4 計測値についての考察

6.2.1 および 6.2.2 にて実際にスループット、レイテンシーの測定を行った。これらの測定結果が実用上問題ない、または実用に耐えうるという判断を行うには、FunctionBox に

表 6.5: Ping による round-trip-time の計測 (単位:msec)

	最小値	平均	最大値
PC—GW—Google	191.977	194.805	197.234
PC—SW—GW—Google	192.403	194.413	197.215
PC—SW—SW—GW	191.736	194.767	197.221
PC—SW—Filter—SW—GW—Google	192.039	194.668	197.223

実装した機能や、機能が使われる場面、状況等をにより判断がわかれる。リアルタイム制御を行う必要がある非常にシビアな環境で使用することを考えると、FunctionBox によるレイテンシーは許容が難しいかもしれない。また逆に、認証機能などの負荷があまりかからない機能であれば、FunctionBox のスループット性能で十分と言えるかもしれない。

そこで一つのケースとして、FunctionBox に実装したフィルタ機能 (フィルタは 1000 行設定) をファイアウォールとして動作させた場合の影響度を、ping 試験によって計測した。「ローカルエリアネットワークに配置するファイアウォール機能」としてこのフィルタリング機能を想定した場合、このスループットとレイテンシーの値がどのように影響するのかを、簡単な Ping 試験で考察してみた。

ローカルエリアネットワークに配置した PC から、インターネット上の WWW サーバへの Ping 試験を行った。試験は下記のような項目で行い、それぞれの差を見ることにより影響度を考察する。項目に対応する機器構成を図 6.15 に示す。

1. PC—GW—Google
2. PC—SW—GW—Google
3. PC—SW—SW—GW—Google
4. PC—SW—PC(bridge))—SW—GW—Google

計測結果を表 6.5 以下に示す。いずれの計測値も 191msec から 197msec であり、大きなばらつきはなかった。PC から直接インターネットに抜ける経路が線路長・経由機器ともに最短経路であるにも関わらず最速の値ではない。逆に本論文の基本アイデアに基づく機能追加を行ったパターンは最速値でないし、最遅値でもない。6.2.1 および 6.2.2 にて実際に測定したスループット、レイテンシーの値は、このような用途においてはほぼ影響しないという結果となった。本論文で示した基本アイデアによる機能追加が実用途としても使用に耐えうるという一つの証左を示す結果となった。

製品にもよるが、ルータを一台経由すると約 2 から 3ms の遅延が生じ、距離に換算しても 1ms は約 200km であり本実験のようにファイアウォールとして本論文の手法による機能追加を使用する場合には、本手法により生じる遅延は、無視できる数値である。

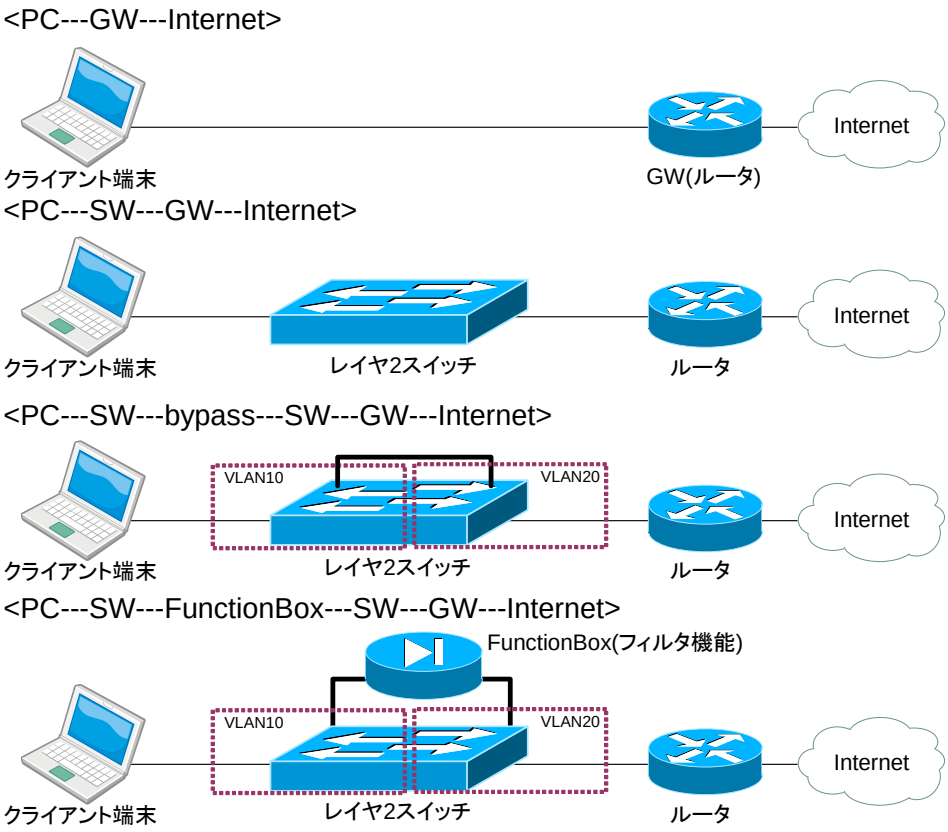


図 6.15: Ping による Round-trip-time の計測

	NCC	Pswitch	単機能App
アイデア	VLAN bypass	Policy switching	目的別単機能 機器(箱)の設置
キーデバイス	既存Switch (VLAN機能有り)	新規開発 Pswitch	目的別単機能 機器(箱)
制御の単位	◎ SWの物理Port毎	◎ Flow 5tuple(src/dst IP,port,orotocol etc…)	× broadcast domain
導入コスト	◎ 既存SWでOK	△ 既存SWを置換え	○ 箱を設置
新規機能の 拡張方法	◎ Function_Box	◎ MiddleBox	△ 箱の開発
スケーラビリティ	○ SOHO～小規模LAN	◎ 3層モデルに対応	△ broadcast domain

図 6.16: NCC と他の手法との比較 1

6.3 定性的評価-提案システムと関連研究・既存手法との比較-

動作実験および性能測定を踏まえ、3 章にて紹介した他の手法との比較を行う。

High-end-Switch は追加的な機能追加に対応するアプローチではないので、Pswitch&Player および単機能アプライアンスとの比較を挙げ、図 6.16 および図 6.17 にまとめた。

High-end-Switch 導入による手法は、結局は新規に開発される機能への対応が約束されていない以上、継続的に機能を追加することはできない。また単機能アプライアンスによる手法も、全ての機能を単機能アプライアンスで提供できるわけではなく、機能の提供範囲を管理できなかったり、設置台数増加に頼らざるを得なかったりと、柔軟さに欠ける点が大きい。Pswitch に代表されるレイヤ 2.5 層を構築する手法は、MidleBox のインライン配置によるトポロジーの硬直化から、Flow 毎に異なる機能提供を可能とすることで、柔軟性においては成功している。しかしながらその柔軟性を提供するためには、全てのトラフィックを Pswitch に集中させる必要がある。また、Pswitch の処理負荷の懸念がある。Pswitch は集められる全てのトラフィックについてレイヤ 2 の処理とレイヤ 2.5 の処理をこなさなければならない。事実、論文 [9] には計測値として Pswitch を経由し F/W の機能を提供した場合、スループット値が 350Mbps、レイテンシーが 0.6msec という値が掲載されていた。フレームサイズについての記述がなく、かつ実装形態も違うため単純な比較はできないが、スイッチをスイッチング処理に特化させ、他の処理を外部機器 (FunctionBox) にて行っている本論文の提案手法がロングパケットにおいてはほぼワイヤレートの性能を

	NCC	Pswitch	単機能App
管理のし易さ	○ NCC	◎ GUI(?)	×
トポロジの把握	△ 複雑なVLAN	○ middleboxのoff-path化	◎ トポロジ変更なし
既存資産との親和性	◎ VLANヘッダ処理のみ	△ 元フレームの書換え	◎
機能提供の自由度	○ VLANパス	◎ Policyの記述力	△ broadcast domain
Throuput /latency	◎ SW	○ PswitchのH/W化	？
新規機能の取込	◎ 2I/F bridge+処理	◎ 既存想定Middlebox	△ broadcast domain

図 6.17: NCC と他の手法との比較 2

発揮している。

第7章　議論

本章では、基本アイデア拡張案やその他のアイデア、本報告書の試行により見えてきた「機能指向型ネットワークデザイン」について述べる。

7.1　基本アイデアの拡張

本報告書で示した基本アイデア (VLAN によるバイパス回路の形成と FunctionBox による機能実装) には、指摘されうる事項や応用範囲を広げるアイデアがいくつかある。今回の試作や実装には非常にシンプルな形式を採用したが、様々な事項のうちいくつかについては対処を考案した。以下、指摘事項およびアイデアとして数例を挙げる。

7.1.1　スイッチ-FunctionBox 間の接続物理ポート数の軽減

本報告書の基本アイデアとして、フレームの解釈・処理を行う FunctionBox の存在がある。この FunctionBox について、提供する機能数の増加は FunctionBox の設置台数増を意味し、提供機能数が増加すると接続に消費されるスイッチの物理ポートが増加し、クライアントの収容にあてる物理ポート数が圧迫されるという指摘がある。

一つの機能につきスイッチの二つのインタフェースを FunctionBox に割り当てることになるため、ポート実装密度が低いスイッチやいくつもの機能を少ないスイッチ台数環境に組み込む際にはこれが問題になる場合がある。

そこで物理インタフェース一つで FunctionBox と接続する、スイッチポートの節約案を考案した。以下 7.1 に示す。

FunctionBox とスイッチの物理ポートをトランクポートととして接続し、FunctionBox の物理インタフェースに作成した VLAN インタフェース同士を Bridge 化することで、1 ポートでのバイパス経路作成と FunctionBox の配置が可能になる。

Ethernet ポート 0 に作成された VLAN ポート「Eth0.10」から入力されたフレームを

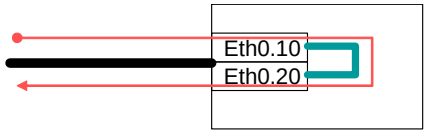


図 7.1: 1port-bridge による使用物理ポートの節約

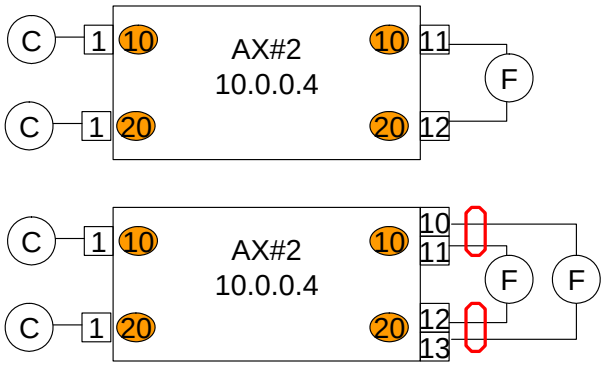


図 7.2: Link-Aggregation によるスループットの改善

FunctionBox にて処理し、同じく Ethernet ポート 0 に作成された VLAN ポート「Eth0.20」から出力することで、FunctionBox 側の物理ポートは 1 ポートで済み、同じく対向するスイッチの物理ポートも 1 ポートで済むため、機能追加に伴い消費される物理ポート数の節約が可能になる。物理ポートを二つ使用する従来の方式に比して、スループット性能が減少するが、認証機能のように一度認証した後はしばらく処理トラフィックが通過しないような機能であれば実用に足り得ると考えられる。

7.1.2 ボトルネックの解消

数 GigaByte のスループット性能を有するスイッチの内部バスに比して、バイパス経路にもちいる物理インタフェースはスループット性能が劣る。そのため機能追加のために内部バスではなく、物理インタフェースを用いる本報告書の基本アイデアはは、高いスループット性能を必要とする環境にとっては問題になる場合がある。

この問題に対する対応として、同じ機能を持った複数の FunctionBox を Link-Aggregation して使用することで帯域を倍にする構成を考案した。構成を図 7.2 に示す。

この手法であれば、帯域を追加的に拡張することができる。

7.1.3 FunctionBox の VM 化

一つの FunctionBox で一つの機能を提供する形でこれまで例示を行ってきたが、複数の機能を実装した FunctionBox による複数の機能提供も考えることができる。物理的な装置が一台であり、この装置の中に複数の機能提供主体となるプログラムが複数ある場合、どの機能を使うのかという選択をどのようにして実現するかが課題となる。複数の処理をシーケンシャルに適用して一連の機能提供とするのであれば、入出力は一つずつしか存在しないので問題とならないが、提供する機能を取捨選択できるよう用意するのであれば、方法を講じなければならない。本報告書の基本アイデアおよび NCC では、機能を提供する FunctionBox にトラフィックが経由するようパスを構築することで機能の選択を実現し

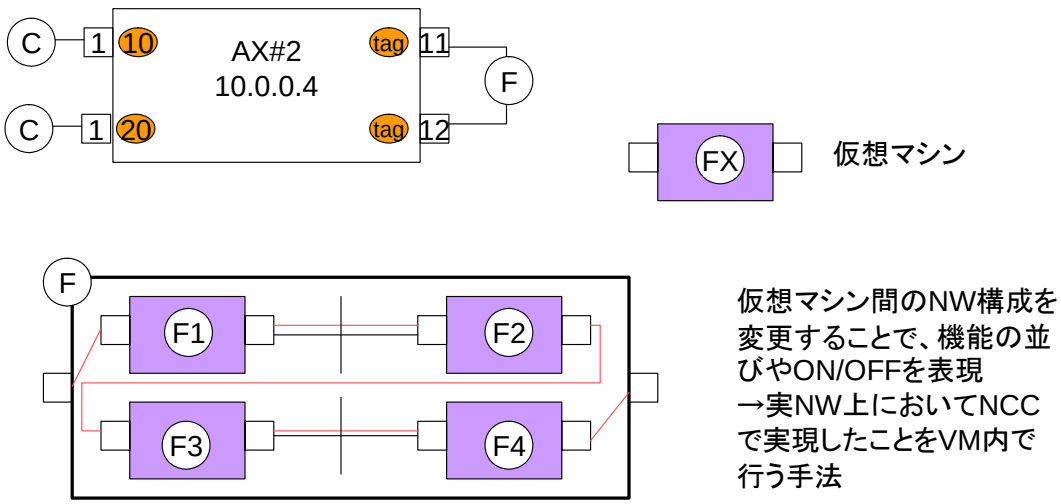


図 7.3: FunctionBox の VM 化

たが、物理的な FunctionBox が一つでその中に複数の機能が存在する場合、トラフィックの経由方法で制御することができない。そこで考案したのが物理的な FunctionBox の中に、機能を提供する VM 化した FunctionBox を構築する手法である。この手法を図 7.3 に示す。

FunctionBox の物理的な入出力インタフェースは一つずつであるため、現行の NCC ではこの FunctionBox にトラフィックを運ぶか否かの制御しかできない。よって FunctionBox 内のどの機能を使うかは、FunctionBox 内に設ける NCC 相当の機構により制御する。

7.1.4 機能の MiddleBox による提供

本報告書では、実際にフレームの処理、解釈を行う装置として、Ethernet インタフェースをブリッジ化して使用する FunctionBox を提案した。これは任意の解釈、処理機構を実現する機能を自由にプログラミングし、作成する機構として考案したものであり、既存機器で望む機能を実現しているものが存在するのであれば、それを使用すれば良い。本報告書の中でも述べた MiddleBox がそれであり、ファイアウォールや NAT 等の機能を有する MiddleBox をバイパス経路上に設置すれば、本報告書の基本アイデアや NCC に取り込むことができる。Pswitch では利用することができなかった MiddleBox も、本報告書の基本アイデアや NCC においては問題なく使用することができる。FunctionBox と MiddleBox は本質的に同一であると言える。

7.1.5 基本アイデアによる機能の簡易実装

本報告書では、既存のスイッチをリプレースせずに機能を追加するための、機能追加手法として、基本アイデアや NCC を試作・提案した。しかし、別の視点で、スイッチに何

か機能を追加したいといった、スイッチに手を加えたいという要望への対案として、本報告書の試みが使用できる。スイッチに機能を追加したり、仕様を変更いたい場合、現状では利用者が自由にスイッチの内部処理機構に手を加えることはできない。もし何かスイッチに改造を施したい場合、スイッチベンダーに直接依頼するしかない。スイッチベンダー側としてもたびたびファームウェアに手を入れるのは面倒であり、両者にとって好ましくない。しかし本報告書の基本アイデアによれば、スイッチベンダーに仕様変更を依頼したり、自らスイッチを自作することなく、機能を試すことができ、簡易的な試作や実験を試みることができる。

7.2 NETCONF への期待

本報告書でスイッチをコントロールする手段に、NETCONF を一部使用した。論文内でも述べたが、ベンダ、機種を問わない統一的な設定インタフェースを提供するのが NETCONF の大目標である。しかしながら、標準化に向けた動きは現在は停滞している。もし NETCONF が将来的にその大目標であるベンダ、機種を問わない統一的なインタフェースを提供できるようになれば、ネットワーク機器の制御が楽に行えるようになるというだけでなく、本報告書で扱ってきた「機能」というテーマににも大きな影響を与える。

本報告書の試みは、ネットワーク機器にフォワーディングプレーン以上の処理を行わせず、FunctionBox に代行させる仕組であった。しかし NETCONF による制御がより現実的なものになれば、ネットワーク機器が持つ情報を FunctionBox が取得し、加工することや、FunctionBox からネットワーク機器を制御することで、いろいろな機能が実現できるようになる。例えば、障害検知による自動的なネットワーク構成の変更、バックアップへの以降、障害復旧時の通常系への移管など、異常を検知する装置からネットワーク機器の制御ができれば、これらが自動で行えるようになる。ネットワーク機器を手足として自由に使うことが可能になれば、その応用例は考えうるアプリケーションや機能の範囲は格段に広がる。

7.3 機能指向型ネットワークデザインの考察

本報告書では、既設のスイッチをリプレースすることなく機能を追加する手法について述べてきた。ここでは視点をかえて、ネットワークデザインの観点から、「機能」を考察した「機能指向型ネットワークデザイン」の考え方について述べる。

7.3.1 従来型のネットワークデザイン

従来型のネットワークデザインでは、「機能」を装置単位で捉え、ネットワークに導入してきた。例えば、ある機能を搭載したネットワーク機器や、ある機能を実現する MiddleBox

図 7.4: 機能を搭載したネットワーク機器による機能導入

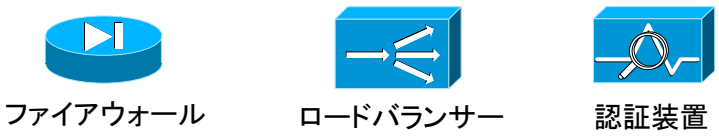


図 7.5: MiddleBox による機能導入

の設置がそれにあたる。機能付ネットワーク機器の例を図 7.4 に、MiddleBox の例を図 7.5 に示す。

このような「機能」を装置単位で捉え導入する従来型のネットワークデザインには、次のような問題点がある。

- 機能追加の度にネットワーク機器のリプレースが行われてしまう
- 装置の設置場所による機能提供の制約が生じる

機能追加の度にネットワークのリプレースが行われるという問題については、2 章にて例を交えて述べた。ネットワーク機器としてのスループット性能には何ら問題が無いにもかかわらず、機能を追加したいがために機器リプレース加が行われる。認証機能を導入するために IEEE802.1X 機能対応機種に既設のネットワーク機器をリプレースする例を 2.4 にて述べた。このようなリプレースによる機能追加は、新しい機能の導入の度に機器のリプレースを発生させる。認証機能の導入後、さらに検疫機能の導入する場合には、また機器のリプレースを検討しなければならない。

次に装置の設置場所による制約の問題について述べる。これは 3.3 節の Pswitch&Player の紹介の中で述べた。MiddleBox をネットワークの根元、例えばゲートウェイルータの直下に設置し、インライン配置する場合、配下のネットワーク全てに機能を提供することができる半面、ネットワーク内で別々のポリシーを当てはめたい場合には、柔軟な対応が困難になる。ネットワークへの MiddleBox のインライン配置を図 7.6 に示す。

逆に MiddleBox を分散配置する場合、より細かなポリシーによる制御を可能とするが、半面 MiddleBox を大量に設置する必要があり費用面での負担が大きく、管理コストも増大する。MiddleBox の分散配置を図 7.7 に示す。

7.3.2 機能指向型ネットワークデザイン

前節では「機能」を装置単位で捉え導入する従来型ネットワークデザインとその問題点、

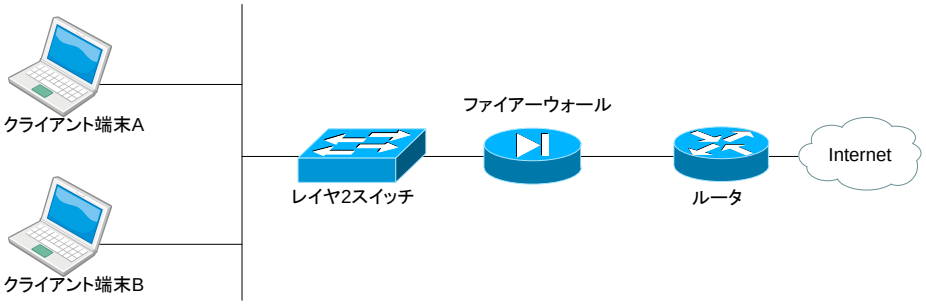


図 7.6: MiddleBox のインライン配置

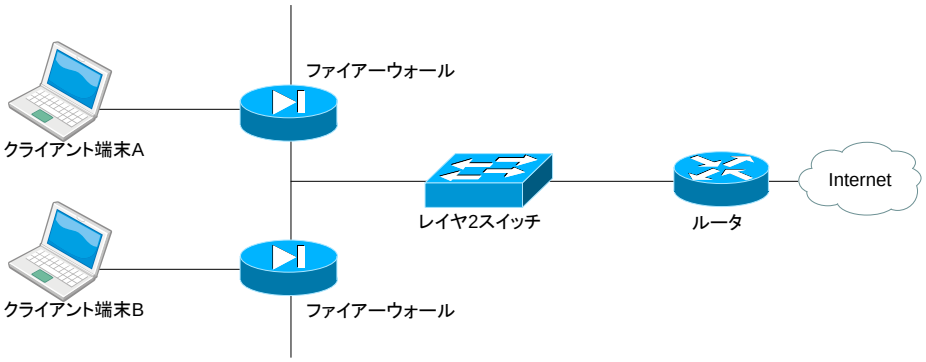


図 7.7: MiddleBox の分散配置

- 機能追加の度に生じる機器リプレース
- 装置の設置場所による機能提供範囲の制約と制限

について述べた。

このような従来型のネットワークデザインの問題を克服する新しいネットワークデザインとして提案するのが、「機能」を基底とした「機能指向型ネットワークデザイン」である。機能指向型ネットワークデザインは、以下を実現する。

- 機能の継続的な追加、拡張
  - 既設のネットワーク機器をリプレースせずに機能を追加を可能とする
- 機能のネットワーク全体への提供
  - 機能実現装置の機能をネットワーク全体へ提供可能とする
- 機能の柔軟な適用
  - 「機能の ONOFF」、「機能の組合せ」を装置の設置場所にとらわれずに適用可能とする

7.3.3 機能の「追加」、「提供」、「適用」

機能の「追加」、「提供」、「適用」の具体的な方法については、4 章 について述べた基本アイデアによる。スイッチの論理分割とバイパス経路の形成により、機能を「追加」する。スイッチ一台につき一つの FunctionBox(もしくは MiddleBox) を必要とするのではなく、提供される機能は、タグ VLAN により、ネットワーク全体に提供される。また提供された機能は、当該機能の使用、不使用および機能同士の組合せ使用を自由に選択し適用することができる。

従来の機能追加手法や機能の提供方法は、ある機能を実装したネットワーク機器や MiddleBox により実現され、当該ネットワーク機器や MiddleBox をネットワーク上のどこに配置するかで提供が決せされてきた。つまりネットワーク機器や MiddleBox といった「箱」をネットワークの「線」上のどこに置くかによって、提供範囲が決定されてきた。しかし本報告書の試みでは、追加的に機能を追加・拡張することを可能にし、「箱」をどこに設置しても、ロケーションを問わず機能を自由に提供できる仕組を提案できた。「機能」を「箱」と「線」の束縛から解放し、機能の提供を実現する、機能を中心に据えた「機能指向型ネットワークデザイン」の雛型が示せた。

第8章　おわりに

本論文では、既設のネットワーク機器のリプレースによらない機能追加手法の提案、実現、および評価を行った。さらには視点をネットワークデザインに転じ、本論文の機能追加手法による機能指向型ネットワークデザインの考え方を述べた。

「機能」を中心に据えたネットワークデザインにより、ネットワークへの機能追加、追加された機能のネットワーク全体での利用など、機能指向型ネットワークデザインの柔軟なネットワークデザインを示した。

従来型のネットワークデザインは、機能を装置単位で捉えており、柔軟性を欠くネットワークデザインを強いられてきた。本論文ではその一例として、機能追加がネットワーク機器のリプレースによって行われている事例を紹介し、機器の廃棄と新規導入、導入作業時のシステムダウンなど、リプレースによる機能追加の問題点を挙げ、機能指向型ネットワークデザインによる解法を計った。

柔軟なネットワークデザインを実現する、その他の先行研究やアプローチについて紹介し、「高機能スイッチの導入」、「単機能アプライアンスの導入」、「ネットワーク機器制御層構築による」手法をとりあげ、それぞれの代表事例や研究について解説し、各々の残存課題を指摘した。「継続的な機能追加ができないこと」、「機器の設置場所に機能適用範囲が束縛されること」、「既存のスイッチに代わる新しいスイッチを導入する必要があること」をそれぞれの残存課題として挙げた。

本論文で提案した機能指向型ネットワークデザインでは、これらの問題点を克服するアプローチ、すなわち「継続的な機能追加を可能とすること」、「機器の設置場所によらない機能適用が可能なこと」、「既存のスイッチをはじめとしたネットワーク機器を有効に活用すること」を基本概念として、機能指向型ネットワークデザインを模索した。

機能指向型ネットワークデザインは、ネットワーク管理者が意図する、任意の機能を任意の順で適用できる論理ネットワークにより実現される。このことは、機能の追加だけでなく、機能を組み合わせた適用や、ある部分に適用しないといった柔軟性をネットワークに持たせることを意味する。

ネットワーク管理者が意図する任意の機能を任意の順に適用する論理ネットワークを構築するプロトタイプ Network Circuit Compiler は、ネットワーク上の資源について記述されたコンフィグレーションファイルを読み込み、式で指定された要望に沿った機能提供ネットワークを構築するプログラムである。

実際にいくつかの機能追加や機能の適用組合せパターンを表現した式を NCC に投入し、評価をおこなった。構築された論理ネットワークの確認や通信試験によって、意図した機

能適用を行うネットワークが構築されていること、意図した機能適用がおこなわれていることを確認した。また、SmartBits によるスループットとレイテンシーの計測により、実用に耐えうる性能を保持できることを確認することができた。

謝辞

研究を行うにあたり、終始熱心に御指導いただきました主指導教員である篠田陽一教授に感謝申し上げます。また、研究の節目節目において貴重な御助言や指摘を賜わった丹康雄教授、敷田幹文准教授、副テーマを指導していただいた日比野靖教授に感謝申し上げます。

本学、知念賢一助教、宇多仁助教、小原泰弘助教には普段よりいろいろとお声かけいただき、様々な御指導、御助言を賜わりました。感謝申し上げます。

情報通信研究機構、三輪信介研究員、宮地利幸研究員、Razvan BEURAN 研究員には、ゼミナールでの発表時等、様々な場面で御指導、御助言を賜わりました。感謝申し上げます。

情報通信研究機構北陸リサーチセンター佐野正行氏には、実験環境の構築や機器操作について多大なご協力をいただきました。感謝申し上げます。

アラクサラネットワークス株式会社、新善文氏、木村浩康氏、木谷誠氏には共同研究にて大変御世話になりました。感謝申し上げます。

本学篠田研究室、LATT Khin Thida 氏、高野祐輝氏、安田真悟氏、井上朋哉氏、SABER ZRELLI 氏、Nguyen Lan Tien 氏、芳炭将氏、NGUYEN, Nam Hoai 氏、中井浩氏、松井大輔氏、佐川喜昭氏、石渡優祐氏、明石邦夫氏、川瀬拓哉氏、 栗原良尚氏には様々な場面で御指導、貴重なコメント、ご協力をいただきました。

最後に、研究や生活を支えてくれた母に感謝いたします。

参考文献

[1] 独立行政法人 情報処理推進機構, 2008 年コンピュータウイルス届出状況について http://www.ipa.go.jp/security/txt/2009/01outline.html#all, 2009.1.8 取得

[2] 独立行政法人 情報処理推進機構, 2008 年コンピュータ不正アクセス届出状況について http://www.ipa.go.jp/security/txt/2009/01outline.html#all, 2009.1.8 取得

[3] 独立行政法人 新エネルギー・産業技術総合開発機構, 技術戦略マップ 2008 情報通信ネットワーク分野 http://www.nedo.go.jp/roadmap/2008/info4.pdf 2008.1.12 取得

[4] The IEEE LAN MAN Standards Comittee, Standard for Port based Network Access Control, Std 802.1X-2001, 2001.

[5] Cisco Systems, Inc.Cisco Catalyst 6500 シリーズ, http://www.cisco.com/web/JP/product/hs/switches/cat6500/index.html, 2008.1.12 取得

[6] Softcreate Co.,ltd., L2Blocker, http://www.l2blocker.com, 2008.1.12 取得

[7] Panasonic Electric Works Co., ltd., IntraPOLICE, http://denko.panasonic.biz/Ebox/i_breaker/intrapolice/, 2008.1.12 取得

[8] 松谷健史, ARP を利用したローカルエリアネットワークにおける不正接続の排除, 第 12 回マルチメディア通信と分散処理ワークショップ,2004.

[9] D A. Joseph, A.Tavakoli, and I.Stoica, A Policy-aware Switching Layer for Data Centers, SIGCOM 2008.

[10] R. Gold, P. Gunnigberg, and C. Tschudin, A Virtualixed Link Layer with Support for Indirection, FDNA 2004.

[11] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, Ethane:Taking Contorol of the Enterprise, SIGCOM 2007.

[12] NetFPGA, http://netfpga.org

[13] The IEEE LAN MAN Standards Comittee, Networks:Virtual Bridged Local Area Networks, Std 802.1Q-1998,1998.

[14] R.Enns and Ed, NETCONF Configuraion Protocol, RFC4741, 2006.

[15] 新 麗, 企業を熱くする最新テクノロジ NETCONF 日経BP 社 日経コミュニケーション 2006.6.15 P80-85, 2006.

[16] ALAXALA Networks Corporation OAN ユーザーズガイド AX-ON-API 編, 2007.

[17] T.Goddard, Using NETCONF Protocol over the Simple ObjectAccess Protocol(SOAP), RFC4743, 2006.

[18] Expect, http://expect.nist.gov/

[19] Click, http://read.cs.ucla.edu/click/