修 士 論 文

ネットワーク検証実験環境における自由度の高いネットワーク構築に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

田部 英樹

2013 年 3 月

修 士 論 文

ネットワーク検証実験環境における自由度の高いネットワーク構築に関する研究

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　知念賢一 特任准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1110040 田部 英樹

提出年月: 2013 年 2 月

目 次

第 1 章　はじめに **1**
1.1 背景 1
1.2 目的 2

第 2 章　関連研究および関連技術 **3**
2.1 テストベッド 3
2.1.1 StarBED 4
2.1.2 Emulab 6
2.1.3 Planetlab 6
2.2 実験支援システム 7
2.2.1 SpringOS 7
2.2.2 AnyBED 9
2.2.3 Weevil 9
2.3 データリンク仮想化技術 11
2.3.1 Q-in-Q 11
2.3.2 MAC-in-MAC 11
2.3.3 VXLAN 13

第 3 章　自由度の高い実験ネットワーク構築機構 **17**
3.1 テストベッドにおけるネットワーク構築 17
3.2 ネットワーク構築に関する要求 21
3.2.1 クラウドサービスの視点からの分析 21
3.2.2 テストベッドの視点からの分析 22
3.3 実験ネットワーク構築の自由度 24
3.4 高い自由度を実現するために必要な性質・機能 24
3.4.1 実験リソースの抽象化 24
3.4.2 大規模な検証実験を並列に実施 26
3.4.3 End-to-End におけるフォーマットフリーなデータ通信 26

第 4 章　設計 **28**
4.1 概要 28
4.2 実験ネットワークの抽象化記述 29

4.3 実験リソースの管理 32
4.4 仮想ネットワークの作成 33
4.5 物理トポロジに依らないデータ転送 34
4.6 機器の自動設定 35

第 5 章 実装 **36**
5.1 構成機構 36
5.2 実験ネットワーク記述形式 36
5.3 ExperimentResourceManager 41
5.4 TopologyManager 42
5.5 BEEF-Controller 42
5.5.1 Beef's Ethernet Equivalent Forwarding(BEEF) 44
5.5.2 最短経路探索によるトラフィックループの回避 47
5.5.3 スイッチ間の物理ネットワークトポロジの動的把握 48

第 6 章 評価と考察 **54**
6.1 動作検証 54
6.2 BEEF のパフォーマンスに関する考察 57
6.2.1 ハードウェア実装との関係 57
6.2.2 ノード配置との関係 58
6.3 仮想インターフェースへの対応 60

第 7 章 おわりに **61**

図 目 次

2.1 StarBED のトポロジ概観 5
2.2 planetlab の概要 8
2.3 AnyBed 概念図 (文献 [1] より) 10
2.4 weevil 概要図 12
2.5 PB 15
2.6 PBB 15
2.7 PBB 16
2.8 VXLAN 16

3.1 検証実験の手順 18
3.2 検証実験の手順 20
3.3 研究開発のサイクル 26

4.1 提案システムの概念 30
4.2 ブロードキャストドメインに属する汎用ノード 31
4.3 ブロードキャストドメインの集合で実験ネットワークトポロジを構成 . . 32
4.4 実験リソースの管理 33
4.5 ポートベースのネットワーク仮想化 34

5.1 提案システムの概観 37
5.2 json のオブジェクト表記法 38
5.3 実験ネットワークトポロジ例 41
5.4 OpenFlow の動作 50
5.5 BEEF-Controller の処理手順 51
5.6 ダイクストラ法 52
5.7 LLDP によるトポロジ把握 53

6.1 テスト構成の物理トポロジ 55
6.2 テスト構成の論理トポロジ 55
6.3 使用するノードが集中している場合 59
6.4 使用するノードが分散している場合 59

概 要

インターネットは、人々の重要な社会基盤として発展を続けている。この発展は、ネットワーク技術の研究や開発の下支えによるものであり、ネットワーク技術の研究や開発は重要である。ネットワーク技術の研究や開発の過程では、その動作や性能の検証実験が行われる。検証実験をインターネットで実施すると、インターネットの特性や他のネットワークサービスからの影響を受ける。複合的な要因による影響を受けると、問題が発生した際の原因特定が困難になるため、必ずしもインターネットは検証実験を実施する理想的であるとはいえない。インターネットの特性から影響を受けず検証実験の実施が可能な環境として、インターネットから隔離されたテストベッドが存在する。

StarBED[2] は、複数の実験者で時分割に設備を共用することで、設備の利用効率を向上している。共用された空間の中で実験ネットワークを構築し、その上で複数の検証実験が並列に実施されるので、検証実験間で影響を及ぼし合う可能性がある。よって、実験ネットワークは、他の実証実験で生成されるトラフィックからの影響を受けないように独立させる必要がある。独立した実験ネットワークを実現する方法として、データリンクの仮想化が挙げられる。VLAN や Q-in-Q などのデータリンク仮想化技術は、データリンクメディアを仮想的に多重化する。仮想データリンクによって、それぞれ独立した実験ネットワークを構築することが可能である。また、データリンク仮想化技術は、実験ネットワークのトポロジを作成するためにも使用される。ノード同士を仮想データリンクで接続することで、実験ネットワークのトポロジを論理的に構築する。

テストベッドの需要の高まりから、より多くの検証実験を並列に実施することが求められている。しかし、従来の仮想データリンクの識別方式から生じる問題があり、要求の解決を妨げている。第 1 に、一般的に用いられている VLAN 等の仮想データリンクの識別に用いるインスタンス識別子の数が仕様で定められており、作成できるデータリンクの数に限界がある。このことから、検証実験の並列度と実験ネットワークの規模がトレードオフの関係にあり、大規模な検証実験を並列に実施することが困難である。第 2 に、実験ネットワークを構築している仮想データリンクの識別に干渉することを防ぐために、検証実験において同じデータリンク仮想化技術を使用することは制限される。これらの制限によって、実験ネットワークに対する要求に応えられない場合がある。

実験者に自由な実験ネットワーク構築を提供するためには、先に述べた制限を取り払わなくてはいけない。大規模な検証実験を並列に実施することを可能にし、検証実験の中で行われるデータ通信についてフォーマットフリーとすることが必要である。

本論文では、Beef's Ethernet Equivalent Forwarding(BEEF) と実験支援システムを提案した。提案システムを用いたテストベッドは、OpenFlow スイッチとそれに接続されたノード群で構成される。BEEF は、OpenFlow を利用したネットワークインスタンス識別子を用いないデータリンク仮想化技術である。OpenFlow スイッチのポートの集合で BOMAIN(Beef dOMAIN) を定義し、BOMAIN がひとつのデータリンクとするフロー制御を行うことでデータリンクを仮想化する。フレームの入力があった OpenFlow スイッチのポート番号によって BOMAIN が識別され、同じ BEEF 内の End-to-End の通信は、従来と同様のスイッチング処理が行われる。仮想データリンクの識別にネットワークインスタンス識別子を用いないため、VLAN のような仮想データリンク数の上限は無い。さらに、検証実験内のデータ通信についてフォーマットフリーな実験ネットワークが実現する。

また、構築したい実験ネットワークによっては、ノードが備えている物理ネットワークインタフェースの数が不足する場合がある。このような場合は、仮想ネットワークインターフェースを作成し、不足を補うことが考えられる。そこで、仮想ネットワークインタフェースに対応したデータリンク仮想化技術として BEEF-V を提案した。BEEF-V では、BEEF が BOMAIN を識別する条件に VLAN タグを加え、仮想インタフェースの識別を可能にした。BEEF-V を用いることで、仮想インタフェースを用いたより規模の大きい実験ネットワークの構築が可能となる。

提案システムについて、BEEF を用いた実験ネットワークの作成実験を行い、既存手法と比較した際の有用性を示した。BEEF のパフォーマンスを、OpenFlowSwitch のハードウェア実装と、実験用ノードの配置の 2 点について考察した。考察の結果から、BEEF の効果的な運用方法について示した。

提案システムを用いることで、テストベッドにおける自由度の高い実験ネットワークが実現すると考えられる。実験者は自由な創造のもとで検証実験を設計し実施することが可能となる。それにより、ネットワーク技術の研究開発が促進され、インターネットの益々の発展が期待できる。

第1章　はじめに

1.1　背景

インターネットは、生活・文化・産業・経済などの様々な面で重要な要素となっている社会基盤であり、その役割は発展を続けている。インターネットの発展は、日々行われている研究や開発に依るところが大きい。ネットワーク技術の研究や開発の過程では、その動作や性能の検証実験を行うことにより、後々に発生する不具合を発見することが重要である。検証実験は、実装段階のデバッグから運用公開までの各工程に関して行われ、実施する環境を構成する要素が結果に深く関わっている。検証実験をインターネットで実施することは、遅延やジッタ等の様々なインターネットの特性や、他のネットワークサービスからの影響を受けるので、想定する運用に即した結果を得ることが可能である。しかし、複合的な要因による影響を受けるので、検証実験に問題が発生した際に原因を特定することが困難である。また、検証実験が他のネットワークサービスに対して想定外の影響を及ぼす可能性があることから、必ずしも理想的な環境ではない。

他のネットワークサービスからの影響を受けること無く検証実験を行うために、多数の汎用コンピュートノードとネットワーク機器から構成された ICT テストベッドとして StarBED がある。StarBED は、複数の利用者で施設の設備を共有することによって利用効率を上げている。Virtual Local Area Network(VLAN) を用いて各実証実験のネットワークを論理的に分離することで、検証実験間のトラフィックが影響し合うことを防ぎ、施設の設備を共有しながらも独立した検証実験を行うことを可能にしている。利用者は、検証実験を構築する為のリソースとしてコンピュートノードと VLAN-ID の割り当てを受け、L2 スイッチの VLAN 設定を行うことによって論理的なネットワークを作成し、コンピュートノード上で任意のネットワーク技術を動作させ、検証実験を実施することが可能である。しかし、StarBED で利用者全体にリソースとして割り当てることが可能な VLAN-ID の数は、IEEE 802.1q の仕様で定められた数による上限があり、多くの実証実験を並列に実施すると、単一利用者に割り当てられる VLAN-ID の数が少なくなるため、単一利用者

あたりの検証実験の規模は制限される。

1.2 目的

大規模な検証実験を並列に実施するには、利用者に提供されるリソースについての高い拡張性が必要である。コンピュートノードは、ハードウェアの性能向上と仮想化技術の発展によって多くの多重化が可能であり、制限を生み出す大きな要因とはなっていない。ネットワーク機器は、機器に組み込まれているネットワーク制御技術によって、作成できる論理的なネットワークの数が決定し、StarBED では VLAN が用いられているので作成できる論理的なネットワークの数は 12bit 相当が上限であり、その拡張性が問題視されている。

そこで本研究は、トラフィックフロー制御によって論理的なネットワークを作成する Beef's Ethernet Equivalent Forwarding(BEEF) を提案する。BEEF は、パケットのヘッダに含まれる情報を用いてトラフィックフロー制御を行うことで、L2 スイッチの任意ポート間で論理的なネットワークの作成を可能にする。タグやラベル等の識別子を付加しないため、VLAN のように作成可能である論理的なネットワークの数に制限を受けず、大規模な検証実験を並列に実施させることが可能になる。

第2章　関連研究および関連技術

本研究における提案システムは実験支援技術に分類できる。実験支援技術の中には、テストベッド, 実験支援システムなどがある。また、データリンク仮想化技術は、テストベッドや実験支援システムの重要な構成技術である。

提案システムは、実験支援技術の中の実験支援システムに位置付けられる。そこで、既存の実験支援システムについて、関連研究として紹介し、各々の役割や機能を説明する。

実験支援システムが利用される場面として、テストベッドが最も代表的なものである。テストベッドのコンセプトやアーキテクチャは様々であり、それにより実験環境の構築方法やそこで実施される実験は異なる。関連研究として既存のテストベッドについて紹介し、実験環境の構築と利用のされ方に焦点を当てて説明する。

データリンク仮想化技術は、実験環境の構築について重要な要素である。前章で挙げた課題はデータリンク仮想化技術に由来するものであり、本研究においても重要な位置付けを占めている。代表的なデータリンク仮想化技術を関連技術として紹介し、実験構築において留意すべき話題について述べる。

2.1　テストベッド

既存の実験支援技術として、テストベッドと呼ばれる設備がある。テストベッドは、研究・開発・検証の為に用いられているプラットフォームであり、様々な分野で活用されている。各々のテストベッドのコンセプトによって、テストベッドを構成するアーキテクチャや運用方法は異なっており、それによって利用者の検証実験の内容や方法も様々である。

本節では、ネットワーク志向プロダクトに関する利用を主な目的とするネットワークテストベッドである StarBED[2]、Emulab[3]、Planetlab[4] について説明する。ここで挙げた以外にも、ADRENALINE[5]、ModelNet[6]、GX-Bone[7]、DRAGON[8] などの様々なテストベッドが存在しており、研究開発に活用されている。

2.1.1 StarBED

StarBED は、北陸 StarBED 技術センター [9] で提供されている ICT テストベッドであり、本物の実装であるバイナリレベルのプログラムコードを、PC アーキテクチャ上のソフトウェアならそのまま、それ以外ならプロセッサエミュレータ等と組合せた実際の環境に近い環境で、必要な周辺の環境を準備した上で実際に動作させ、その際に起こる正作用だけでなく様々な周辺環境との副作用などを含めた挙動を観測することで、実際の環境で動かした場合に何が起こりうるのか、どうしたら防げるのかなどを予見的かつ実証的に検証することが基本的なコンセプトである。

StarBED の構成の概観は図 2.1 となっており、1000 台を超える汎用計算機と、それらが接続されているネットワーク機器から成るクラスタである。汎用計算機は利用者に対して実験環境を構築するためのノードとして提供される。ノードはインターネットから隔離されており、インターネット上で動作しているネットワークサービスからの影響を受けずに検証実験を実施することが可能である。また、ノードは仕様に応じてグループと呼ばれる単位で区切られており、それぞれのグループに所属する計算機ノードの性能は同一である。

StarBED のノードは、最低 2 つのネットワークインタフェースを持ち、最低 1 つずつのネットワークインタフェースがそれぞれ管理用ネットワーク (Management Network) と実験ネットワーク (Experiment Network) に接続されている。管理用ネットワークでは、ノードのネットワークインタフェースの MAC アドレスに基いて、静的に IP アドレスが設定される。この IP アドレスは、利用者からの希望がない限りは変更されず、汎用ノードの設定中、実験中を問わずに接続性を提供する。この管理用ネットワークを通じて、ノードに対する OS やソフトウェアの導入を行う。実験ネットワークでは、L2 スイッチの VLAN 設定を行うことで、実験環境を構成する実験ネットワークトポロジを物理的な配線を変更することなく論理的に構築することが可能である。

利用者は実験環境を構築するために、提供されているノードの一部および実験ネットワークトポロジを構築するための VLAN を借りて実験を行う。また、実験を行う際には実験者が計算機やネットワーク機器などの機材を持ち込み、それらを StarBED の設備に接続して実験を実施することも可能である。

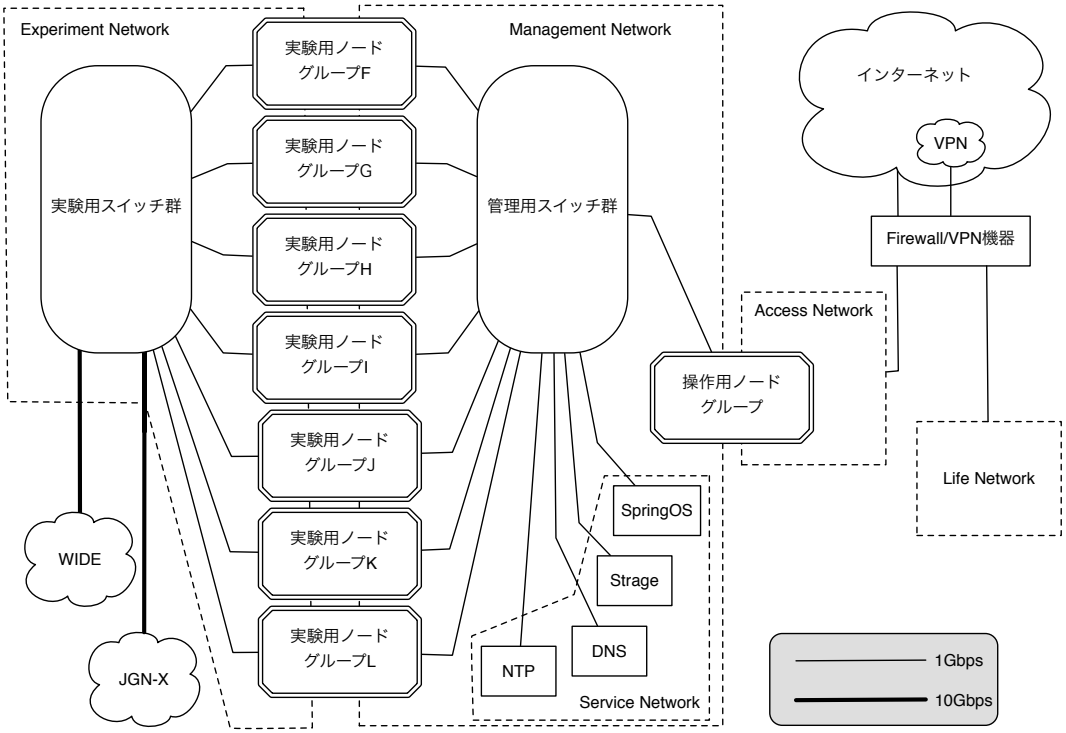

図 2.1: StarBED のトポロジ概観

2.1.2 Emulab

Emulab は、1999 年から開発が始まったネットワークテストベッドであり、計算機やネットワークなどの幅広い分野で、開発・デバッグ・評価などに関する実験を実施できるように設計されている。利用者は、ネットワークエミュレーション・ネットワークエミュレーション・実ネットワークを用いた実験などを実施することが可能である。Emulab は公共設備として、世界中の利用者に対して無償で公開されており、任意の合法的な研究や実験に対して、営利企業による使用を含めて許可されている。利用者は設備を使用する申請を提出し、施設の設備が利用可能な状態になり次第実験を開始することができる。

Emulab のアーキテクチャを採用したテストベッドは複数あり、最も主要となるのはユタ大学にて運用されている。その他には、世界中で 20 以上の拠点が存在し、各拠点を接続することで大規模な環境が構成されている。Emulab のアーキテクチャを構成する要素は、制御サーバと実験用ノードとそれらを接続するネットワークトポロジである。

実験用ノードは、汎用的な計算機でありクラスタを構成している。クラスタ内の実験用ノードは、複数の NIC が取り付けられており、プログラマブルなパッチパネルを介して相互に接続されている。パッチパネルは相互接続されたスイッチで構成されており、VLAN を用いて任意に実験ネットワークトポロジを構築できる。クラスタ内の実験用ノードは、実験の中で様々な役割を果たすことができ、利用が任意の実装を動作させるシナリオノード・ネットワークエミュレータを動作させ、シナリオノードの間に挿入されるリンクノード・シミュレータを動作させるシミュレーションノード・仮想マシンのホストなどがある。

構成管理機能を持つ制御サーバは、テストベッド全体を管理しており、テストベッド機能の制御・テストベッド内のデータを保持するデータベース・ノードの電源管理・テストベッドへの web からのアクセス機能・名前解決・ノードへのディスクイメージの配布などの役割を持つ。各実験用ノードは、パッチパネルに接続されているものとは別の物理インタフェースによって、管理ネットワークのスイッチを介して制御サーバに接続されている。

2.1.3 Planetlab

Planetlab は、世界レベルで展開されている公共のテストベッドであり、ネットワークサービスの開発・展開・検証を実環境に近い条件で行うことができる。特定のネットワークアーキテクチャやサービスに限らず、分散ストレージ・広域サービス・P2P アーキテク

チャなど様々な分野で用いられている。Planetlab は、インターネットに直接接続されたノードの集合であり、35ヶ国 480 拠点以上のメンバーが提供する 1000 以上のノードで構成されている。

実験用ノードは Linux VServer[10] によって多重化されており、複数の実験者が同時に利用可能である。このため、Planetlab 上での実験で使用されるソフトウェアは、Linux 用の実装である必要がある。実験用ノードは、インターネットに直接接続されているので、インターネットのトラフィックや遅延の影響を考慮した検証実験が可能であるという特徴を持つ。

Planetlab で、利用者に割り当てられるリソースは Slice と呼ばれ、Slice を 1 台のノードに対して割り当てた単位は Sliver と呼ばれる。Sliver は Linux VServer を用いて実装されており、1 つの Sliver が 1 つの仮想マシンとして実現されている。また、メモリなどの資源は、1 つの Sliver が大量に消費しないように制限が儲けられている。

Sliver の操作は SSH を用いて行うことが可能であり、利用者が自分の使用する公開鍵を Planetlab の管理システムである Planetlab Central(PLC) に登録すると、各実験用ノードに対して公開鍵が配布される。Slice の名前がユーザ名として使用されるため、Slice@hostname のようにして Sliver が指定でき、ssh Slice@hostname のように実行することで ssh を用いて Sliver への接続が行える。

2.2 実験支援システム

2.2.1 SpringOS

SpringOS[11] は、StarBED における検証実験の実施を支援する実験支援ソフトウェア群である。主な機能は、検証実験に必要な汎用ノードや VLAN-ID などの実験資源の割り当て、汎用ノードへの OS やソフトウェアの導入、実験用ネットワークのトポロジ設定、実験シナリオの実行などがある。これらの動作は、実験の実行者による設定記述や、コマンドの投入によって実行される。これら以外にも、汎用ノードの電源管理や、起動方法の変更を行う機能が提供されている。

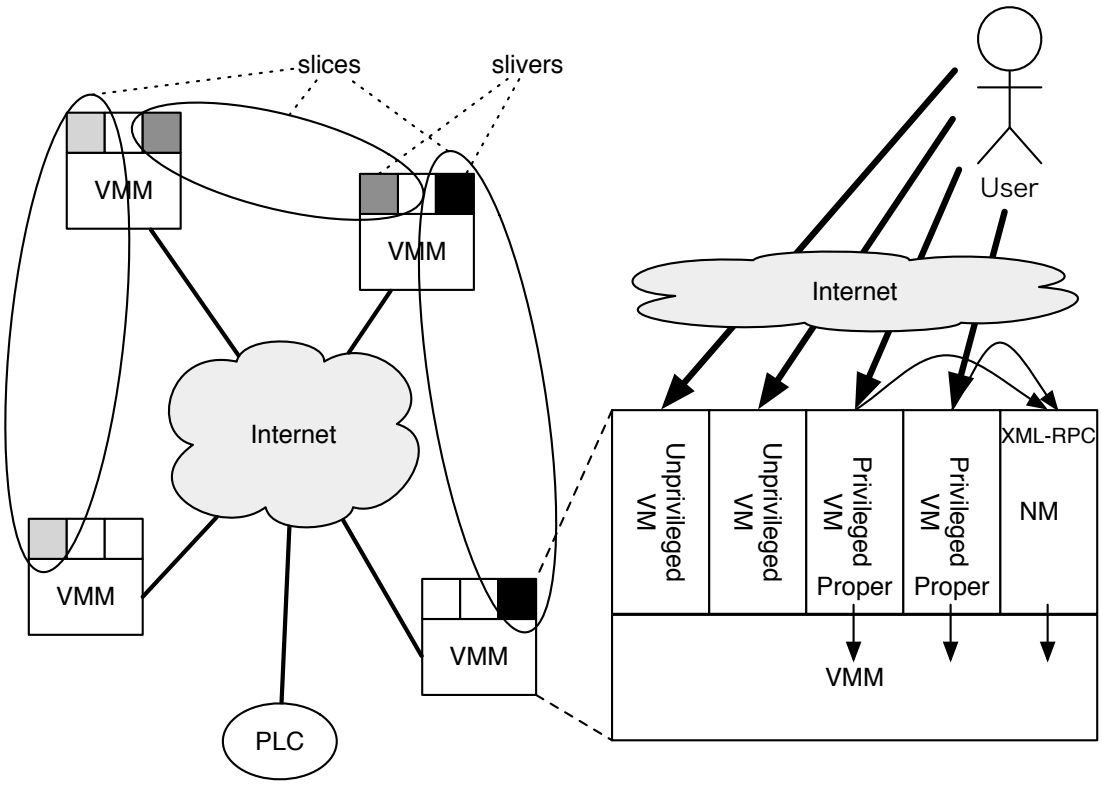

図 2.2: planetlab の概要

2.2.2 AnyBED

AnyBED[1] は、テストベッドでの検証実験における実験ネットワークトポロジの構築工程を支援するシステムである。AnyBED は 3 つの層から構成され、実験ネットワークトポロジ構築のための情報を集める層であるデータ収集層、資源割り当てを行う資源割り当て層、実際の実験ネットワークを構築する設定反映層である。データ収集層と資源割り当て層のデータ交換には XML 記述されたファイルが用いられており、記述形式が同じであれば各層を構成するコンポーネントは容易に交換可能である。

AnyBED では、実験ネットワークトポロジを論理トポロジと物理トポロジに分離して扱う。論理トポロジは、特定のクラスタ環境に依存しない形で、構築したい実験ネットワークのレイヤ 3 トポロジを記述する。物理トポロジは、AnyBED が利用されるクラスタ環境に固有の情報である物理ノード・配線・ネットワークインタフェースの帯域などの情報を記述する。

実験を行う際には、記述された物理トポロジを元に論理トポロジに対して資源の割り当てを行い、その割り当ての結果生成された設定ファイルをクラスタ環境の各物理ノードおよびレイヤ 2 スイッチに配布して設定を行い、実験ネットワークトポロジが構築される。つまり、実験者は論理トポロジファイルを作成し AnyBED に与えるだけで、実験ネットワークトポロジ構築のための大半の作業は AnyBED が自動的に実施する。

このように、AnyBED は資源割り当てを自動化することにより利用者の作業量を減らし、誤設定や誤操作を防ぎ、短時間での実験ネットワークトポロジの構築を可能としている。さらに、検証実験に利用するクラスタ環境の設備構成が異なる場合でも、実験ネットワークトポロジの再利用性が確保されているという特徴がある。

2.2.3 Weevil

Weevil[12] は、地理的に分散した設備で構成されるテストベッドにおいて、実験環境の構築と検証実験の遂行を自動化するために設計されたツールである。このツールは、実験環境を構成するコンポーネントに対して、サービスコールを生成し、発行するスケジュールを設定する。この実験シナリオを定義するためには、利用者は UML で定義された抽象構文に準拠する GNU m4 マクロファイルを作成する必要がある。Weevil は次の 2 つの構成要素から成る。

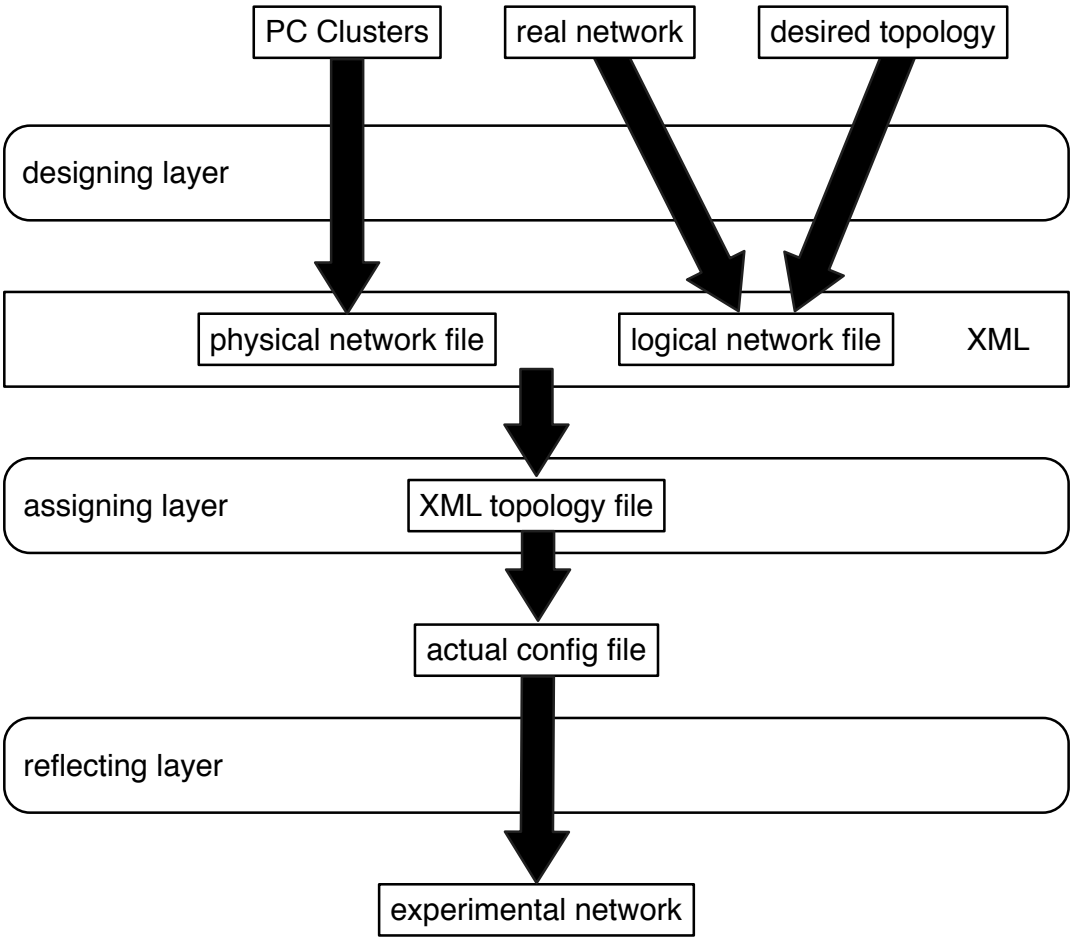

図 2.3: AnyBed 概念図 (文献 [1] より)

weevilgen はクライアントの状態を自動で生成するシミュレータを用いている。weevil は設定ファイルが与えられることによって、以下の実験シナリオを遂行する。

1. 各コンポーネントのスクリプトの始動と停止
2. 実ノード上での実験用のバイナリを展開
3. 実験コンポーネントとアクターの起動
4. 検証実験のログを収集
5. 実験環境の初期化

2.3 データリンク仮想化技術

2.3.1 Q-in-Q

PB(Provider Bridges)[13] は、広域イーサにおけるキャリア網内のスイッチングに採用されている技術で、IEEE 802.1ad として標準化されている。PB は企業内の VLAN で用いるフレームに 12bit の S-VID(Service-VLAN ID) を付加し、このサービスタグが付加されたフレームによってサービスを識別する。このように二重にタグを付与することによって論理ネットワークの識別できる上限を引き上げている。PB のフーレムフォーマットを図 2.5 に示す。

2.3.2 MAC-in-MAC

PBB(Provider Backbone Bridge)[14] は、広域イーサネットサービスを提供するための機能を規定している技術で、IEEE802.1ah として標準化されている。

広域イーサネットサービスで用いられる PB(IEEE802.1ad) では、サービスタグに含まれる S-VID によって VLAN が識別される。しかし、この S-VID の長さは 12bit であり、最大で 4094 までしか識別できないという問題点がある。そのため、PB による既存の広域イーサネットサービス網を PBB によって階層化して相互接続することによってこの問題を解決する。

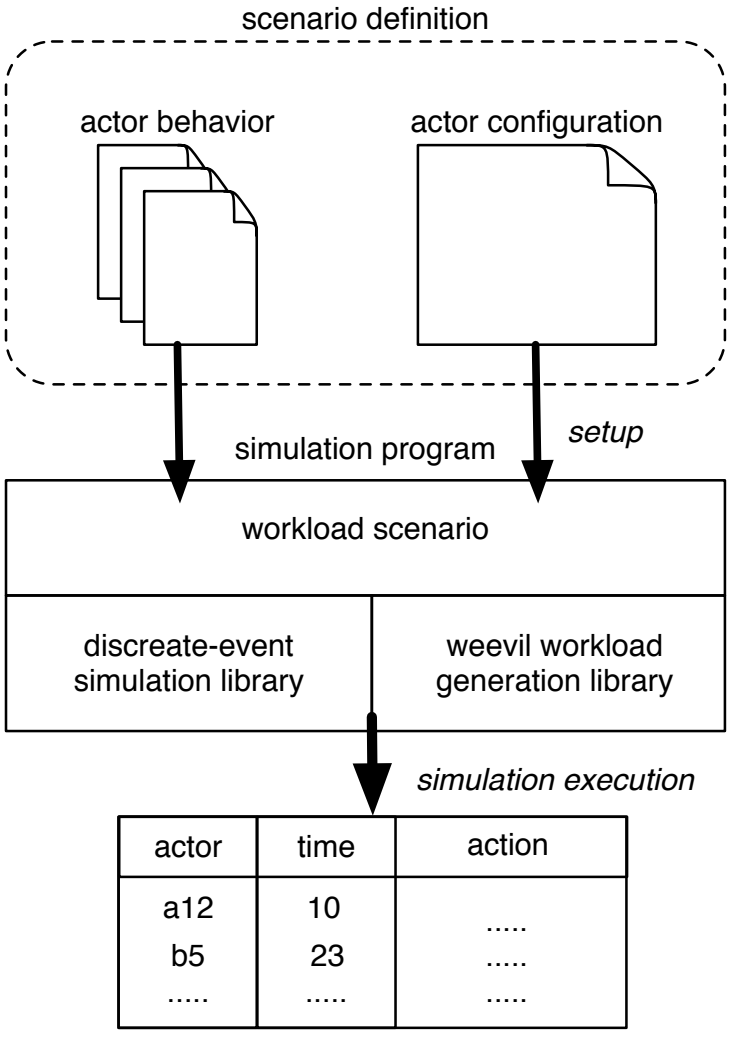

図 2.4: weevil 概要図

PBB では, PB のサービスタグが付いたフレームを、データセンタ間伝送用の情報が付与されたフレームにカプセル化して転送する MAC-in-MAC 方式を用いている。このカプセル化の際に, 拡張サービスインスタンス識別子である 24bit の I-SID を付加する、これによって VLAN を識別すると同時に VID のスケーラビリティの問題を解決している。PBB のフレームフォーマットとネットワークの概観を図 2.6 と図 2.7 に示す。

データセンタ間の転送では、カプセル化の際に付与されたエッジスイッチの MAC アドレスを使用する、これによってデータセンタ内のコアスイッチは、これまでのように全ての仮想マシンの MAC アドレスを学習する必要はなく、エッジスイッチの MAC アドレスだけを学習・保持すればよいので FDB が圧迫される問題が解決する。

また、トラフィックループへの対応として、カプセル化の際に付加される送信元 MAC アドレスを示す B-SA ヘッダを監視して, フレームを転送する際にそれが自分から送信されたものかをチェックし、自分から送信されたものだった場合にはフレームを破棄する、これによってトラフィックループの拡大を抑制している.

2.3.3 VXLAN

VXLAN は、L3 ネットワーク上に仮想 L2 セグメントである VXLAN セグメントを作成する技術である。現在、Broadcom・Cisco・VMware・Citrix・Red Hat、Intel が中心となり、IETF にて標準化作業が行われている。

VXLAN ネットワークは、物理スイッチか物理サーバのハイパーバイザに実装された Virtual Tunnel End Point(VTEP) によって構成される。VTEP は仮想トンネルの終端点となり、フレームをカプセリングすることで 8byte の VXLAN ヘッダを付与する。VXLAN ヘッダには、1bit の VXLAN フラグ・24bit の VXLAN Network Identifier(VNI)・予備の 39bit が含まれ、この VNI によって VXLAN セグメントの識別を行い、同一 VXLAN セグメントに所属するホスト同士の通信が許可される。VXLAN ネットワークの概観とフレームフォーマットを、図 2.8 に示す。

VXLAN ネットワークでは、ホストは VXLAN ネットワークを認識しておらず、従来通りのフレーム送信を行う。ホストが接続している VTEP は、ホストが所属する VXLAN セグメントに対応する VNI を取得し、その VNI を含む VXLAN ヘッダによって受け取ったフレームをカプセリングし送信する。受信先の VTEP はパケットを受け取ると VNI の有効性を確認し、MAC アドレスから適切なホストに転送する。それと同時に、送信元ホス

トの MAC アドレスと送信元 VTEP の IP アドレスのマッピングを学習する。

ホスト同士が通信を行う時、これらが同じサブネットに存在する場合は、アドレス解決は ARP Request のブロードキャストによって行う。これに対して、異なるサブネットに存在する場合は、ブロードキャストはマルチキャストでカプセリングされて、同じ VXLAN セグメントのマルチキャストグループに送信される。ARP Reply は、送信元のマッピングが完了しているので、従来通りのユニキャストによって送信される。

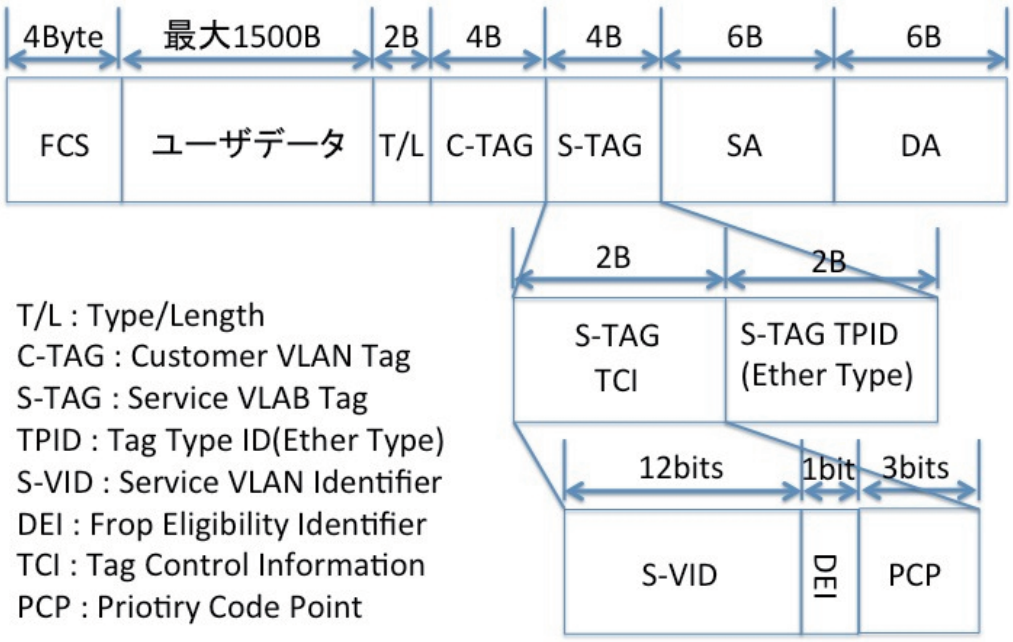

図 2.5: PB

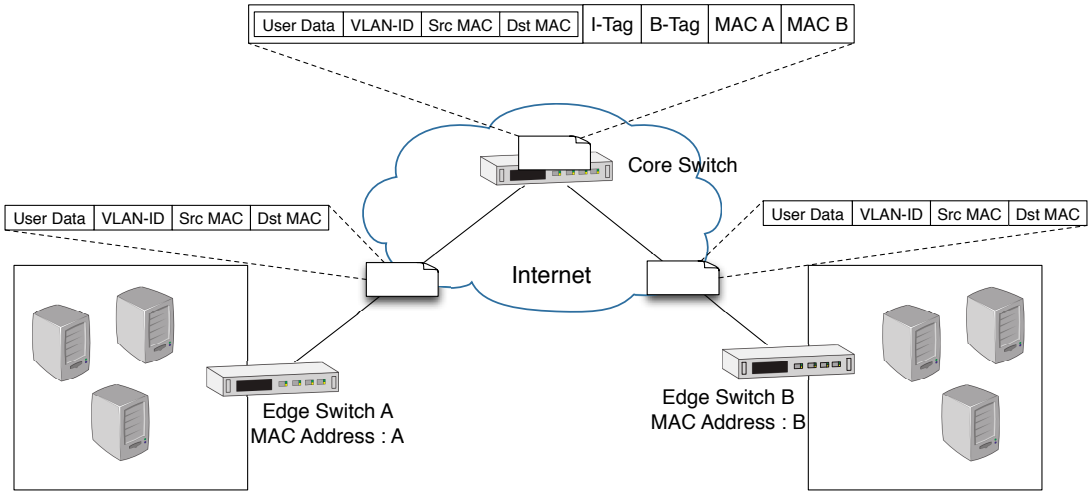

図 2.6: PBB

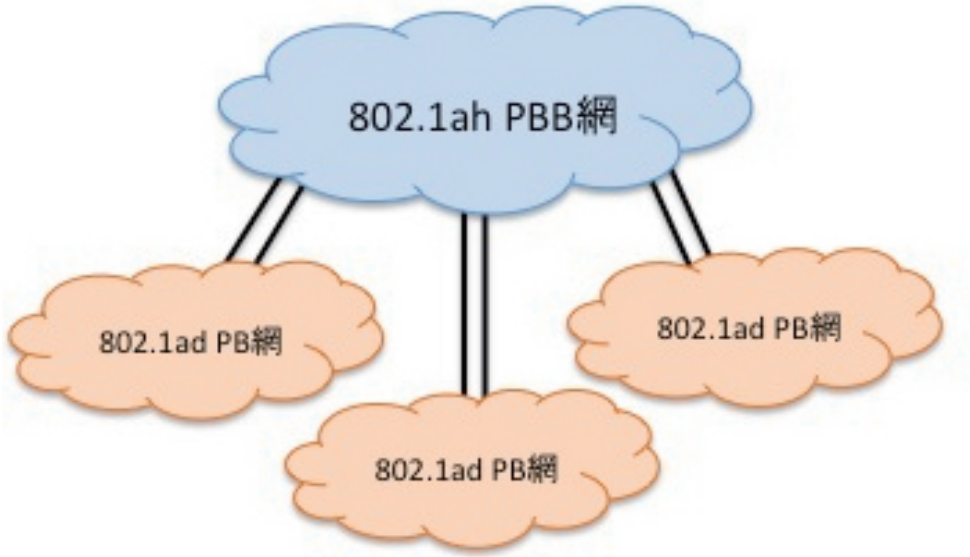

図 2.7: PBB

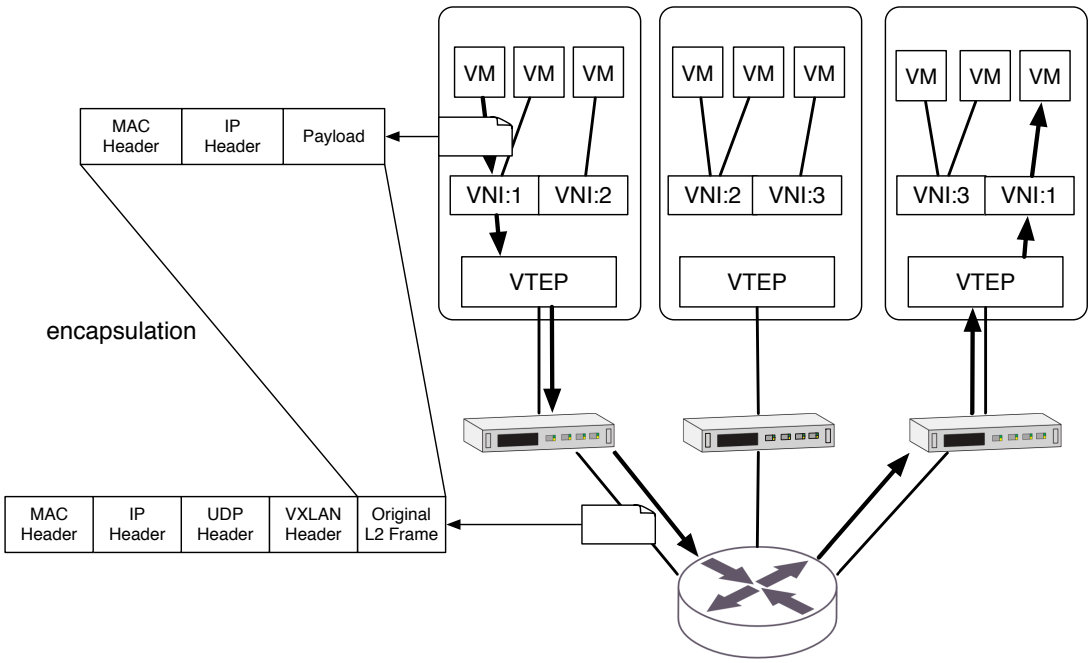

図 2.8: VXLAN

第3章　自由度の高い実験ネットワーク構築機構

3.1　テストベッドにおけるネットワーク構築

本節では、テストベッドにおける実験環境を構築し検証実験を実施する際の一般的な手順を述べ、その中でのネットワーク構築の位置付けについて述べる。

宮地らは、ネットワーク技術の開発について、図 3.1 のように論理的検証と実験環境、大規模な実験環境を用いラ検証を組合せた手順の提案を行った。[15] この提案では、論理的検証によりアルゴリズムの検証後、小規模な実験環境を構築し検証実験を行い、大規模な実験環境の検証を行うとしている。検証の各フェースにおいて問題が発生した場合に、同じフェーズでの検証を繰り返し、不具合が起こった実装や設定を修正して再実験を行う。ここでいう小規模な実験環境とは、実験者が個人で所有できるような数台から数十台の計算機で構成された環境であり、大規模な実験環境とは数十台以上の計算機で構成され、かつ実験支援ソフトウェアが導入されている環境のことである。検証に大規模な実験環境を必要とする技術では、このような検証手順が必要とされる。

また、実験設備での実験の実行手順の提案もされている。[2] 実験環境を構成するノードとして計算機の実機自体を利用するテストベッドでは、実験環境の構築と検証実験の実施を以下の手順で実施する。

1. 実験トポロジやシナリオの検討
2. ネットワークトポロジの作成
3. ノードへのソフトウェアの導入
4. 構築した実験環境でのシナリオ実行
5. 実験ログの解析

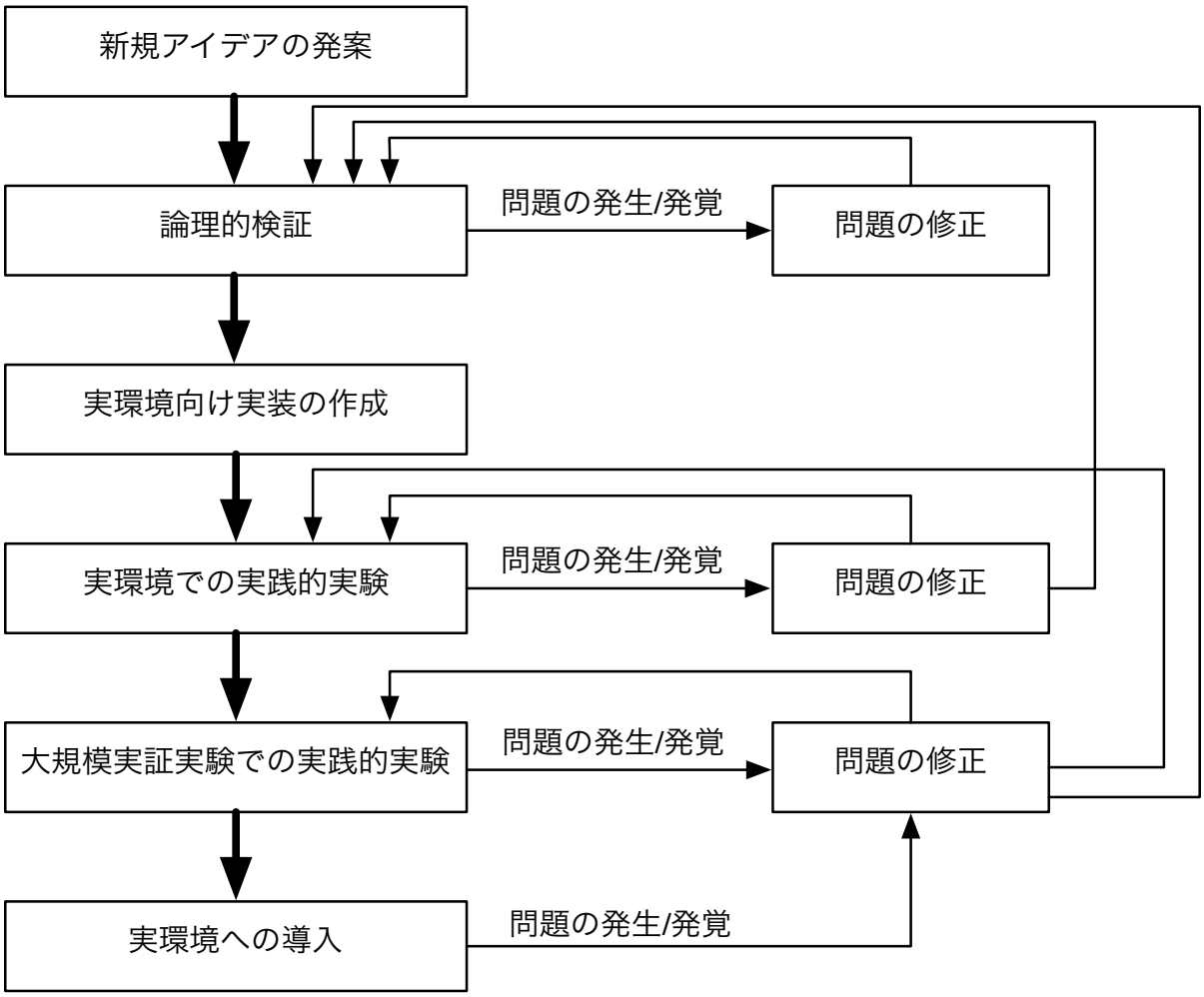
新規アイデアの発案
論理的検証
問題の発生/発覚
問題の修正
実環境向け実装の作成
実環境での実践的実験
問題の発生/発覚
問題の修正
大規模実証実験での実践的実験
問題の発生/発覚
問題の修正
実環境への導入
問題の発生/発覚

図 3.1: 検証実験の手順

実験計画は、実験シナリオと実験ネットワークトポロジからなる検証実験の設計である。利用者は、目的とする検証実験の内容にあわせてどのように検証実験を進行していくかという実験シナリオを検討する。その実験シナリオを元に、検証実験の実施に必要な実験環境の構成を検討し、構築する実験ネットワークトポロジを決定する。ネットワーク機器に相当する機材が実験リソースとして提供されないテストベッドでは、実験環境の構成にルータやスイッチなどのネットワーク機器が必要な場合、ノードに Quagga[16] や Openvswitch[17] 等のソフトウェアを導入することで必要な機能を導入することが考えられ、それを踏まえた検証実験の内容および構成の検討が必要となる。

決定された実験計画から必要な実験リソースの数が明らかになり、テストベッドの運用者に対しての申請が行われる。この際に、実験支援システムを含むテストベッドを利用するために必要なアカウントやパスワード等の決定も行う。テストベッドに実験リソースの空きがあり、利用者を受け入れることが可能な状態ならば、利用の許可が通知される。

次に、ノードを接続し、ネットワークトポロジを作成する。このネットワークトポロジは、物理的な配線作業は省力化のために行われないことが多く、ネットワーク仮想化技術によって論理的にネットワークトポロジを作成する。ネットワーク仮想化技術は様々なものがあるが、テストベッドでは VLAN が一般的に持ちられている。ノードが接続されている L2 スイッチに対して VLAN の設定を行うことで目的となるネットワークトポロジを作成する。

このようにしてノード同士を接続しネットワークトポロジを作成したら、各ノードにオペレーティングシステム (OS) をインストールする。そして、アプリケーションソフトウェアを導入し、検証実験を開始する。

このようにして汎用ノードの接続が完了したら、汎用ノードに対して OS やソフトウェアの導入を行い、各種設定を行う。この際に、ネットワークインタフェースに対しての IP アドレスの設定が行われる。

設定が完了したら、目的の検証実験を開始する。テストベッドによっては、作成したシナリオに基づいた検証実験の自動遂行の機能を提供している。

検証実験が終わったら、ログの解析などを行い、検証実験の結果を確認する。

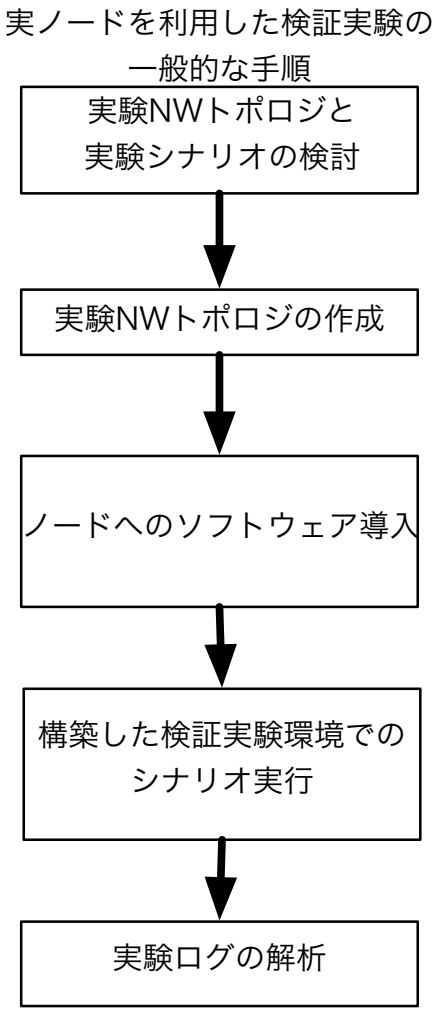
実ノードを利用した検証実験の
一般的な手順
実験NWトポロジと
実験シナリオの検討
実験NWトポロジの作成
ノードへのソフトウェア導入
構築した検証実験環境での
シナリオ実行
実験ログの解析

図 3.2: 検証実験の手順

3.2　ネットワーク構築に関する要求

3.2.1　クラウドサービスの視点からの分析

あらかじめ用意された計算機ノード群からなるクラスタ環境において、利用者の要求に応じてネットワークの構築が行われる具体例としてクラウドサービスがある。本小節では、ネットワーク構築に対する要求について、クラウドサービスの側面から考察する。

クラウドの定義は様々なものが提唱されているが、代表的なものとして [18] 以下の項目が挙げられている。これらの項目からは、様々な技術要件が導き出されるが、特にネットワーク構築に関わる事項を重点的に取り扱う。

1. 規模性
2. 即時性
3. 可用性

規模性は、利用者に資源の量を意識させずに、必要な環境を構築できる。即時性は、必要な時に必要な量の資源を利用できる。可用性は、単一の障害の影響を受けずにサービスを継続することができる。

サーバ仮想化技術の進歩により、管理コスト圧縮のためのサーバ統合が促進され、クラウドによるサービスの提供が盛んに行われている。[19] このような背景の中、以下のような要求が顕在化している

1. 物理サーバの利用効率の向上
2. 複数案件による共用 (マルチテナント化)
3. 運用効率の向上

前述したように、クラウドサービスではマルチテナント化のためにユーザ単位に隔離されたネットワークが提供できること、高集約化やマイグレーションに対応するためにネットワークの柔軟かつ俊敏な構築および構成変更ができることが求められている。しかし、既存の VLAN や製品を用いた方法では困難な課題が存在する。

その一つとして、ネットワーク設計の難しさがある。クラウドサービスの一つであるデータセンタでは、ユーザ単位でネットワークを VLAN で隔離する方法が一般的に利用

されている。しかし、VLAN の収容能力に起因する問題がある。VLAN は IEEE802.1q の仕様により隔離可能なネットワークの数は最大で 4094 であると定められている。データセンタのネットワーク全体で一意に VLAN-ID を定める必要があるので、4095 以上のユーザを同時収容することは困難である。増加する需要に対応する為には、より多くの利用者を多重して収容することが必要である。

より多くの利用者を収容することで、ネットワークが複雑化し運用管理コストが増大することも懸念される。利用者ごとにネットワークの設計・設定・管理を行う必要があり、ネットワーク設定変更時に他のネットワークへ影響を与えないようにするなど、設定変更時の信頼性を確保しなくてはならない。また、ネットワーク構造が複雑化すると、障害が発生した場合に、その障害箇所の特定が困難になる。これらのことから、仮想ネットワークの運用管理の効率化が必要であるといえる。

さらに、サーバの仮想化の成熟に伴い、Virtual machine(VM) を他のサーバに移動させ、設備の利用効率をや利便性を向上したいというマイグレーションの要求がある。しかし、VM をマイグレーションするためには、同じレイヤ 2 セグメントにサーバが属していなければいけない。そのため、VM を任意のサーバにマイグレーションしたい場合は、その都度、移動させたい側のサーバのネットワークを同じレイヤ 2 セグメントに設定するなどの作業が必要となり、即時性を妨げている。

サーバ仮想化技術の成熟に対してネットワーク仮想化技術が追いついていないという現状からこれらの課題が発生している。これに対して、様々なネットワーク仮想化技術が提案・運用され、その利活用によって諸問題を解決する取り組みが行われている。このように、クラウドサービスを効果的かつ効率的に構築し運用するためには、ネットワーク仮想化技術は重要な要素であるといえる。

3.2.2 テストベッドの視点からの分析

テストベッドは、クラウドサービスの一つの応用例であり、自由なネットワーク機能の創造を可能とする環境として期待されている。テストベッドを構成する要素技術として、ネットワーク仮想化技術に期待される役割は大きい。ネットワーク仮想化技術は、実験ネットワークトポロジ構築に関して重要な構成要素であり、テストベッドの利用と運用における様々な要求が存在する。

テストベッドにおいてネットワークの仮想化を行う目的は、利用者ごとのテナント分離

に加えて、実験ネットワークトポロジの構築も含まれる。実ノードを結ぶリンクはネットワークの仮想化によって作られた論理ネットワークであり、実験ネットワークトポロジの規模が拡大するに従ってより多くの論理ネットワークが必要となる。さらに、テストベッドの需要が増加していることから、より多くの検証実験を並列に実施し、施設の利用効率を向上することが求められている。すなわち、検証実験の並列化と実験ネットワークトポロジの規模性の両立を実現するネットワーク仮想化技術が必要である。

テストベッドを利用して研究開発される技術は、ユーザ向けのサービス技術だけではなく、それらを支えるサーバ実装技術やネットワーク実装技術など、インフラストラクチャ技術が多い。本来、テストベッドとは、利用者が検証実験の対象とする技術が主役であって、その要件にあわせて検証環境を提供することが目的である。そのため、テストベッドを構成するアーキテクチャは、多様なインフラストラクチャ技術の検証実験に対応可能であることが求められる。

この点は、仮想化されたサービスを提供するシステムとして、スケールメリットが追求可能なインフラストラクチャの構築を指向する一般的なクラウドコンピューティングとは異なる。すなわち、テストベッドインフラストラクチャにおける仮想化には、多種多様なインフラストラクチャ技術を用いた実証実験が実施可能であるアーキテクチャが必要である。

また、テストベッドは、研究開発で利用されているインフラストラクチャであり、その挙動は科学技術的観点から十分な精度で計測ができなければならず、かつ再現性が担保されなければならない。そのため、仮想化されたテストベッドインフラストラクチャにおいて、システムが定性的に正しく動作すればよいというわけではない。特に、実証実験などではシステム性能上限付近の検証が行われることが多いが、仮想化されたインフラストラクチャではオーバヘッドがあるため、事前にシステムを動作させ、詳細なシステム性能検証が必要となる。

テストベッドは、研究開発を目的とした利用者に対するサービスであり、その観点からは、一般的なクラウドコンピューティングサービスと同様に、ユーザに対して抽象化された一貫性のある簡素なインタフェースを提供する必要がある。すなわち、利用者が検証対象とする技術の観点からは、他の技術要素は隠蔽され、必要な操作インタフェースのみが提供されるサービスモデルの実現を目指すべきである。

3.3　実験ネットワーク構築の自由度

ネットワーク構築における要求の分析から、本研究における実験ネットワークトポロジ構築の自由度について定義する。本研究では、特に利用者に対して提供する機能・性質を重視し、以下の 3 点の性質を自由度を考える上での指標とした。

- 実験ネットワークの規模
- 検証実験の実施の並列度
- 検証実験内のデータフォーマット

データリンク仮想化技術によって作成できる仮想データリンクの数は、そのデータリンク仮想化技術が定義できるネットワークインスタンス識別子の数によって規定されている。ネットワークインスタンス識別子は、仮想データリンクの識別に用い、データリンクメディアを仮想的に多重化する。テストベッドでは、ノード同士を仮想的なデータリンクで接続することで、実験ネットワークのトポロジを論理的に構築する。本研究では、ノード間を論理的に接続する仮想データリンクの数が多いほどネットワークの規模が大きいとした。すなわち、テストベッドで構築できる実験ネットワークの規模は、テストベッドで用いられているデータリンク仮想化技術によって上限が定まる。

3.4　高い自由度を実現するために必要な性質・機能

本節では、実験ネットワーク構築に関して高い自由度を実現するために必要な性質について挙げる。

3.4.1　実験リソースの抽象化

宮地らはネットワーク技術の研究開発について、論理的検証と実験環境、大規模な実験環境を用いた検証を組合せた手順の提案を行った。宮地らの提案では、論理的検証によりアルゴリズムの検証後、小規模な実験環境を構築し実験を行い、大規模な実験環境での検証を行うとしている。ここで述べた小規模な実験環境とは、実験者が個人で所有できるような数台から数十台の計算機で構成された環境であり、大規模な環境とは数十台以上の計算機で構成され、かつ実験支援ソフトウェアが導入されている環境のことである。検証に

大規模な実験環境を必要とする技術では、上記の様な検証手順が必要とされる。このような手順で検証を行い、目的とした環境に技術を導入する。しかし、技術開発はそれで終わりではなく、運用後に発生した不具合の修正や、機能の追加のために図 3.3 のように再度の設計・実装・検証・運用のサイクルを繰り返すことになる。このような技術開発の過程において、検証実験は繰り返されることが特徴としてわかる。このとき、ある検証実験に用いた実験環境をそのまま用いれば実験環境構築の手間を省くことができる。しかし、前回の実験から時間を経る場合や、複数人で機器を共用している場合には、別の実験のための実験環境が構築されていることが考えられる。こういった場合には、前回の同じ実験リソースを仕様することはできない。

実験環境を構成する機器を独占することができれば、実験環境を維持することで実験環境構築の作業を省けるが、大規模な実験環境を特定の実験のために維持し続けることは効率が悪く困難である。そのため、検証実験ごとに固定的な同じ実験リソースを使用するのではなく、その時々に使用可能な機器を選択して使用することでこの問題に対応することができる。よって、検証実験の繰り返しを考慮した検証実験の再利用性は、検証環境構築の最初の段階である実験ネットワークトポロジの設計において特定の実験リソースに依存しないことが必要である。

ネットワークの設計は、(a) 実施したい検証実験の内容を基に、どのような構成にするべきかを考え、ルータやスイッチなどのネットワーク機器を含んだ抽象的な構成を決める (b) 各構成要素に用いる機器を具体的に決定する。という順で実施することが考えられる。本システムは、利用者の記述によって (a) の段階が行い、実験支援システムによって (b) の段階を行う。利用者は、ノードとそれらを結ぶ論理的なネットワークの接続関係を記述する。記述の要素はテストベッドのファシリティ構成に非依存となるように抽象的な値を用いる。

これによって、ファシリティ構成が異なった他のテストベッドにおいて、同一トポロジで実験を行う際に同じトポロジ記述ファイルを用いることができるため、再利用性が向上する。また、テストベッドは複数人で施設の設備を共有するため、利用の時期によって割り当てられるリソースは変化する。このような場合も、トポロジ記述の再利用性が求められる。すなわち、ネットワークを構成する実験リソースを体系的に記述し、利用者やサービスからの要求を形式的に指定し割り当て可能とする、資源の抽象化を実現する必要がある。

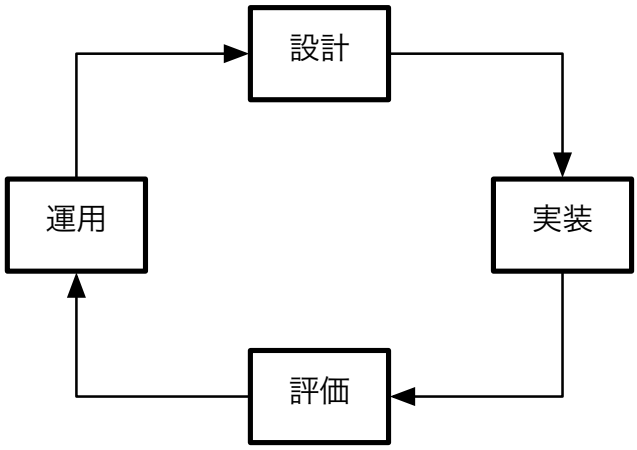

図 3.3: 研究開発のサイクル

3.4.2 大規模な検証実験を並列に実施

テストベットの需要が増加しているため、施設の設備の利用効率を向上させ、多くの検証実験を並列に実施することが求められる。利用者はテストベッドの施設を共有しているため、検証実験を並列に実施すると、各検証実験で生成されているトラフィックが混在し、検証実験の挙動や結果に影響してしまう可能性や、セキュリティを保てないという問題がある。これらの問題を解決するためには、共通のテストベッドインフラ全体を論理的に分割することで複数の検証実験を同時に収容し、それらの検証実験を並列に実施することが必要である。さらに、自由な創造を妨げない為に、独立したネットワークの中で利用者の任意の規模の実験環境を構築できるべきであり、大規模な実験ネットワークトポロジでも構築可能であることが必要である。すなわち、検証実験の規模と並列度の拡大を両立できる環境が求められている。

3.4.3 End-to-End におけるフォーマットフリーなデータ通信

検証実験内のトラフィックにタグやラベル等を付加することによってネットワークトポロジを構築している場合、検証実験においてノードで送受信されるデータのフォーマットに制限が発生する。これは、ネットワークトポロジを構築している技術と検証実験で用いようとしている技術の衝突が干渉するためである。このような環境では、検証実験の内容が制限され、利用者の自由な創造が妨げられる。そこで、構築された実験ネットワークの

中では、Ethernet に関してフォーマットを規定しないフォーマットフリーのデータが流れることを保証することが必要である。

第4章　設計

4.1　概要

提案システムは、自由度の高い実験ネットワーク構築を実現するために、実験ネットワークトポロジの設計と構築の役割を担う。提案システムは、以下の4層の概念で構成されており、各層の機能で段階的に処理が行われる。

A. 設計層 (designing layer)

B. 変換層 (conversion layer)

C. 定義層 (definition layer)

D. 制御層 (control layer)

検証実験の各段階と提案システムの各層の対応は図**??**のようになっている。実験ネットワークトポロジの設計の部分は設計層によって行い、ノードを接続して実験ネットワークトポロジを構築する段階は変換層・定義層・制御層によって行われる。

設計層では、利用者に対して、構築したいネットワークトポロジを表現する記述方式を提供する。利用者は、定められた記述方式を基にネットワークトポロジの設計を書き下し、検証実験の準備を行う。このようにして作成されたネットワークトポロジ設計ファイルは、システムへの入力して与えられる。

変換層では、設定層において利用者が作成したネットワークトポロジ設計ファイルを入力として受け取り、記述の内容に基づき必要な実験リソースの割り当てが行う。割り当てられた実験リソースの機器識別子やハードウェアの情報などを取得し、それの物理構成をネットワークトポロジ設計に適応させることで具体化する。このように、抽象的なネットワークトポロジの設計を変換層で記述内容の変換を行い具体化することで、利用者はテストベッドの設備や構成についての知識を必要とすることなくネットワークトポロジを設

計することが可能になる。変換層は、このような記述の具体化を行うための変換機能と、テストベッドの設備や構成の情報を管理する機能を持つ。

定義層では、物理構成が適用された設計から、利用者が想定する実験ネットワークトポロジを構築するためにネットワークの定義を行う。提案システムでは、テストベッドの物理構成に囚われずに実験ネットワークトポロジを構築することを可能とするために、ネットワークの論理構成を変更することで物理構成を変更せずに実験ネットワークトポロジを柔軟に構築することを可能にする。

制御層では、ネットワーク機器に対する設定を生成しデータ転送の制御を行うことで、変換層と定義層の処理によって物理トポロジと論理トポロジが明らかにされた利用者のネットワークトポロジ設計をテストベッド上に射影する。

このように、利用者はテストベッドの設備や構成の知識を必要とすることなく、実験ネットワークトポロジを設計する。また、設計に基いて提案システムが自動的にネットワークを構築するので、ネットワーク機器の操作に関する技術を利用者に対して要求しない。

4.2 実験ネットワークの抽象化記述

利用者は、構築したい実験ネットワークトポロジを何らかの形式で表現し、システムに与えることによって実際にネットワークを構築する。一般的にネットワークトポロジを構築する際、ネットワーク機器に対して CLI や GUI のインタフェースから直接に設定を施す作業を行う。しかし、複数の利用者が並列に検証実験を実施するテストベッドでは、ノードの割り当てによっては 1 つのネットワーク機器を複数の利用者が操作する状況もあり得る、そのような場合、ある利用者の誤設定や誤操作などのヒューマンエラーが他の利用者の検証実験に影響を及ぼす可能性がある。このようなヒューマンエラーとなり得る機会を減少する為に、利用者ごとにネットワーク機器に対して設定できる項目やパラメータを定め、他の利用者に関わる設定を制限するパーミッションの概念が必要となる。

また、ネットワーク機器に対する設定は、ベンダや機種によってコマンド体系やパラメータが異なる。利用者がネットワーク機器に対して直接に設定を施す場合、そのテストベッドで用いられているネットワーク機器の設定に関する知識が必要となり、ネットワーク機器のリプレースや異なるネットワーク機器を用いているテストベッドを利用する際は必要となる知識が変わる可能性もある。このようなネットワーク機器の設定方法について

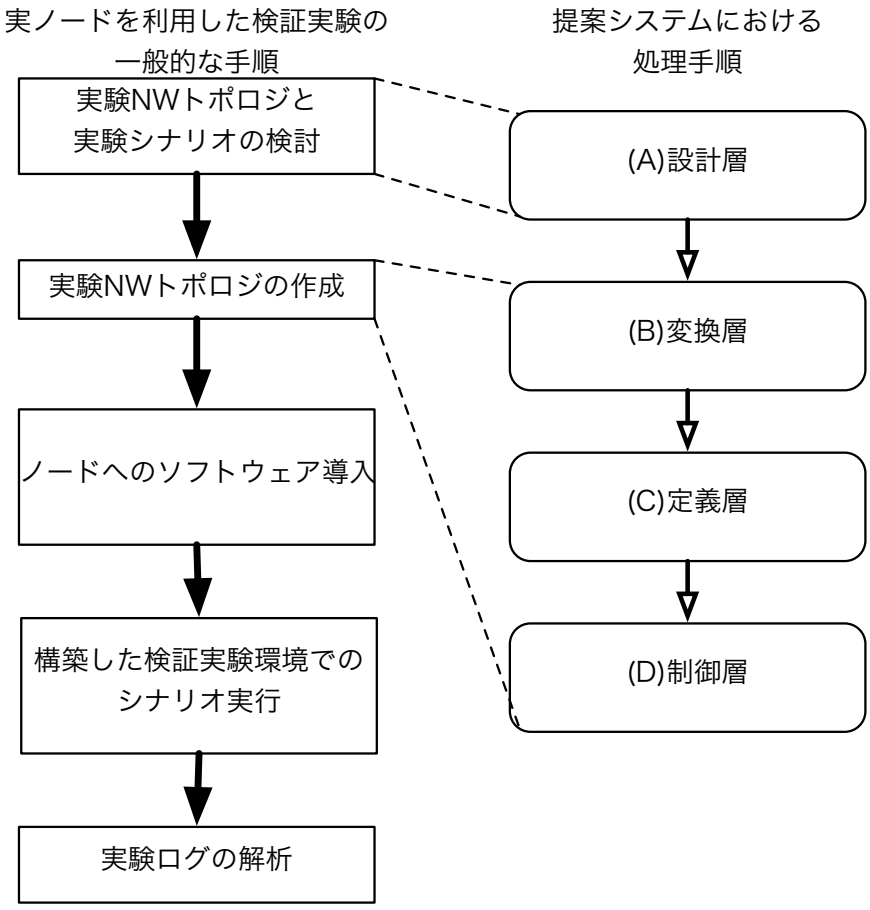

図 4.1: 提案システムの概念

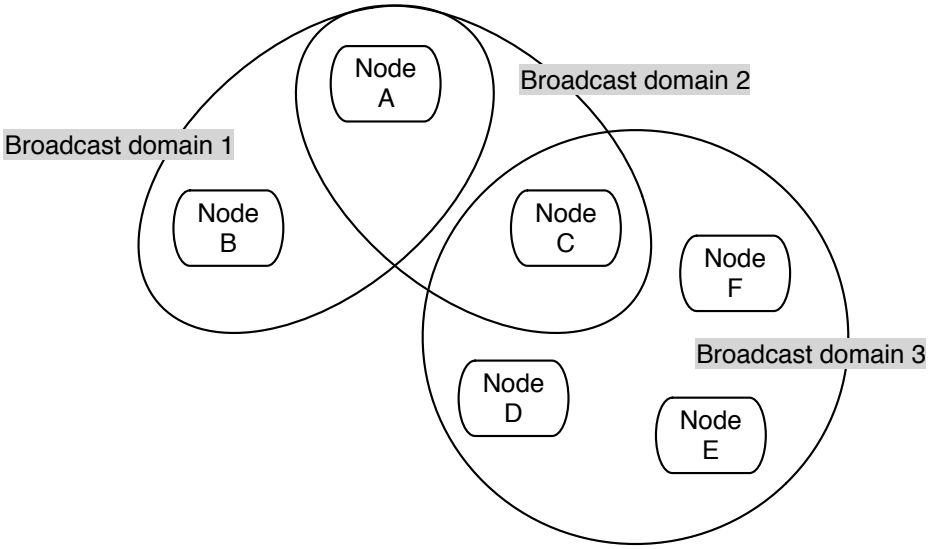

図 4.2: ブロードキャストドメインに属する汎用ノード

の知識を利用者に要求することは、ユーザビリティを低下させる要因となる。そこで、ベンダや機種ごとの設定の差異を吸収する機構を利用者とネットワーク機器との間に設けることで、ユーザビリティの低下を防ぐ。

提案システムの設計層では、トポロジ設計ファイルの記述方式を利用者に提供する。利用者が提供システムを用いる場合は、この記述方式に従ってトポロジ設計ファイルを作成する。このトポロジ記述は、構築したいネットワークトポロジを、図 4.2 のようにブロードキャストドメインとそれに属する汎用ノードの関係を表現する。これにより、図 4.3 のように L2 レベルでのネットワークトポロジが作成でき、L2 以上のネットワーク技術の実装についての検証実験が可能となる。

記述する値に施設固有の識別子を使用すると、利用の時期によって割り当てられるリソースが異なる可能性のあるテストベッドではトポロジ記述を再利用することは困難である。また、設備構成が異なる他のテストベッドで同じトポロジ記述を使用することも難しい。

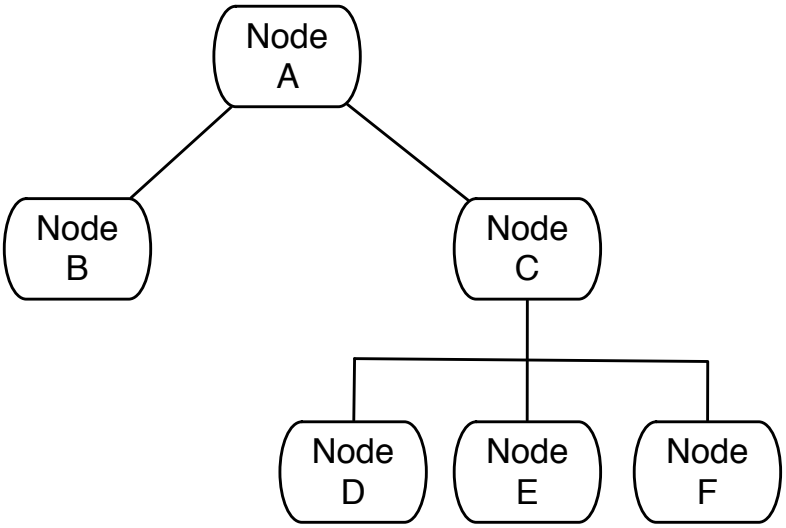

図 4.3: ブロードキャストドメインの集合で実験ネットワークトポロジを構成

4.3 実験リソースの管理

リソース管理機構は、テストベッドの実験用リソースを管理し、利用者が記述した実験ネットワークトポロジ記述に基づき実験用リソースの割り当てを行う。本研究では、ネットワークインスタンスの識別子を固定的な実験用リソースとして扱わず、利用者の要求に応じて動的に定義する。従って、リソース管理機構が管理する実験用リソースは汎用ノードに関する情報のみである。

実験ネットワークトポロジ設計の記述に含まれるコンピュートノードの部分に、その利用者の利用権限がある汎用ノードの情報を適用させ、抽象的な設計からテストベッドの設備と構成に対応した物理トポロジにトポロジ記述にあるノードに、実際の汎用ノードを割り当て、抽象的なトポロジ記述を具体化する。論理的なネットワークを作成するために必要な情報として、割り当てられたノードに関する情報を収集し、ネットワークを定義するための設定を構成する。

汎用ノードは、予めテストベッドの運用者によってどの利用者に対して利用権限を与えるかが定められており、利用権限が与えられている範囲の汎用ノードが利用者の要求に応じて割り当てられる。

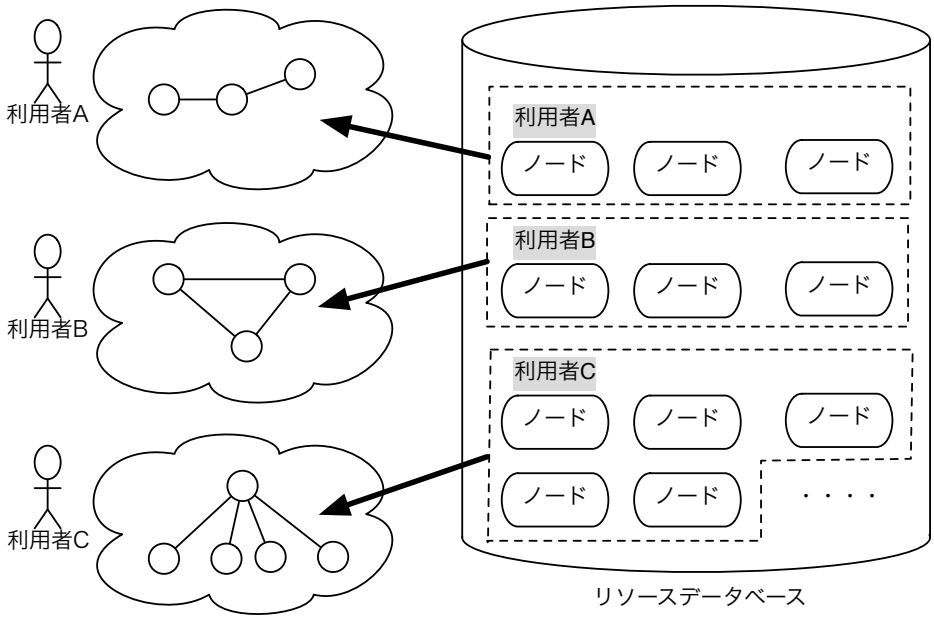

図 4.4: 実験リソースの管理

4.4 仮想ネットワークの作成

既存のイーサネットでは、フレームを End-to-End で転送する際に、以下に示す 5 つの項目が重要となる。

1. 送信元識別子
2. 宛先識別子
3. ネットワークインスタンス識別子
4. 経路・キューの識別
5. サービスタイプ識別子

既存のイーサネットでは、ネットワーク制御技術に VLAN を用いている場合、(1)Destination MAC Address、(2)Source MAC Address、(3)VLAN-ID、(4)VLAN-ID+Priority、(5)Ethernet Type としてイーサネットヘッダ内にエンコードされている。VLAN-ID のようなネットワークインスタンス識別子が End-to-End の識別に用いられているため、ネットワーク制御技術の仕様によるネットワークインスタンス識別子数の制限が発生し、作成可能な論

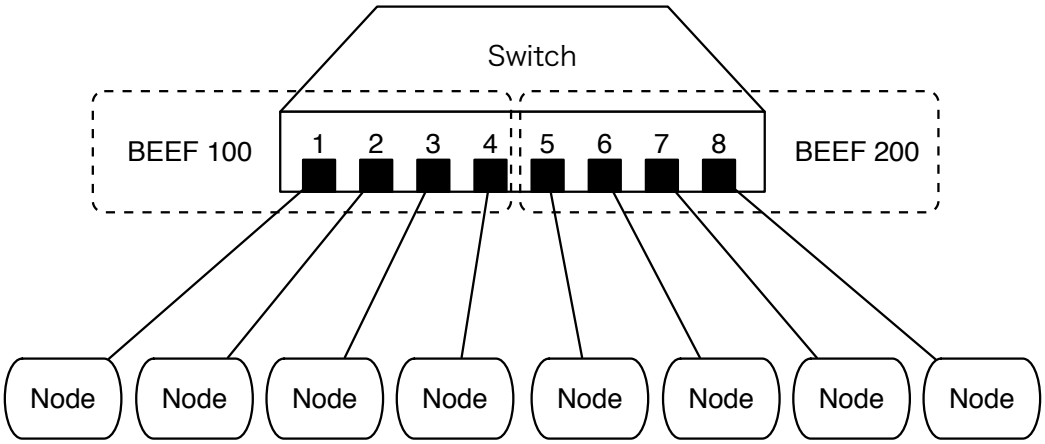

図 4.5: ポートベースのネットワーク仮想化

理ネットワーク数に影響することが、ネットワークの拡張性に関する課題であるとされている。

そこで、本研究では、ネットワークインスタンス識別子に依らない論理ネットワークの作成を行うことで、テストベッドで作成される実験ネットワークトポロジの規模を拡張可能にする。

4.5 物理トポロジに依らないデータ転送

利用者に対して割り当てられるノードは、要求された台数や割り当て状況によっては、異なる複数のネットワーク機器に接続されたノードが割り当てられる場合がある。複数のネットワーク機器に接続されたノードを論理的なネットワークで接続するには、複数のネットワーク機器を跨がる経路制御が必要である。この場合、ネットワーク機器間の物理的な接続にループ構成が存在する可能性がある事を考慮しなくてはいけない。ループ構成は冗長構成を実現するために作られることがあるが、トラフィックのループを発生させる可能性があり、対策が必要である。

4.6 機器の自動設定

ネットワーク機器を設定する場合、一般的にはネットワーク機器に備わっている CLI を用いる。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 入力による方法もとられる。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はあるが、入力される内容は人間が直接入力する CLI コマンドである。CLI による操作は、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用であるが、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るといえる。CLI はあくまでの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 をスクリプト言語から実行するための CLI over Telnet や、Expect などの試みが行われてきた。このことから、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は重要であることがわかる。

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代わり、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、ベンダーや機種が異なった場合、ベンダや機種ごとの 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なっている。

第5章　実装

5.1　構成機構

提案システムは、以下の構成要素からなる。

- Topology Manager
- Experiment Resource Manager
- BEEF-Controller

初めに、実験者が作成したトポロジ設定ファイルを Topology Manager に入力として読み込ませる。Topology Manager はトポロジ設定ファイルの記述内容を解析し、必要な実験リソースの数を Experiment Resource Manager に要求する。Experiment Resource Manager は、自身が持つ実験リソースデータベースの中から、利用が可能な PC ノードの情報を取り出し、Topology Manager に返す。Topology Manager は、提供された PC ノードからトポロジを構成し、PC ノードを結ぶ仮想データリンクを明らかにする。そして、BEEF-Controller に仮想データリンクの作成を指示する。BEEF-Controller は、OpenFlow スイッチのポートを同じドメインに設定するように OpenFlowSwitch を制御する。

5.2　実験ネットワーク記述形式

実験ネットワークトポロジ設計ファイルを作成するためには、何らかの形式で目的のネットワークトポロジを表現する必要がある。実験ネットワークトポロジ設計ファイルは利用者が作成するものであるので、記述性と可読性を満たすために以下の項目が重要である。

- 複雑な構造の記述が可能である

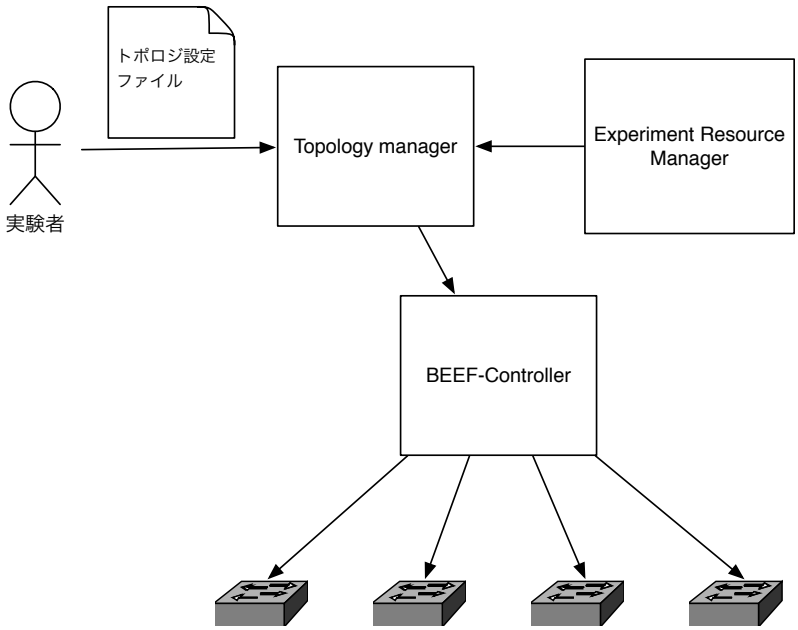

図 5.1: 提案システムの概観

- 容易に記述が可能である
- 記述された内容が理解しやすい

そこで、提案システムにおいては、ファイルフォーマットにデータ記述言語である JSON[20] を用いる。

JSON は、軽量のデータ交換フォーマットであり、人間にとって読み書きが容易で、コンピュートマシンによるパースや生成も可能な形式である。JSON の形式を図 5.2 に示す。オブジェクトは、順序付けされていない名前 (string) と値 (value) のセットであり、” ”で始まり” ”で終わる。名前の後ろには” : ”が付き、名前と値のペアは” , ”で区切られる。名前は、二重引用符で囲われてたゼロ文字以上のユニコード文字の集まりであり、一つの文字も文字列として扱われる。値は、二重引用符で囲まれたオブジェクト・名前・配列などであり、これらの構造はネストできる。配列は、順序付けされた値の集合であり、” [”で始まり”] ”で終わり、値は” , ”で区切られる。

この JSON の形式を基に、利用者が実験ネットワークトポロジ設計を書き下すための記述形式を定義した。実験ネットワークトポロジ設計は experiment 記述部・node 記述部・topology 記述部から構成される。各部分の記述形式を以下に示す。

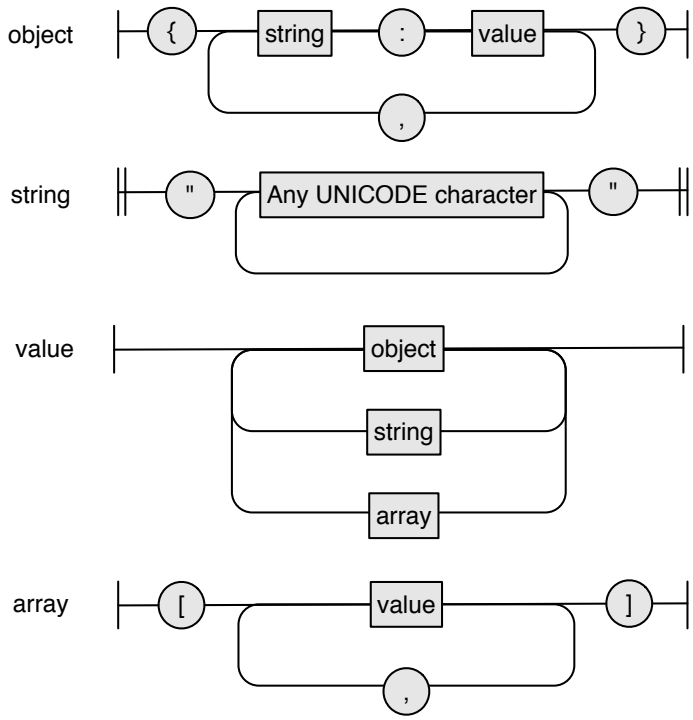

図 5.2: json のオブジェクト表記法

experiment 記述部は、利用者と検証実験を示すためのオブジェクトであり、下記の記述形式である。

```
{"experiment":
    {"user":"利用者名",
     "pass":"パスワード",
     "project":"プロジェクト名"
    }
}
```

”experiment”を名前として持つオブジェクトであり、この値としてに”user” ”pass” ”project”の名前を持つ 3 つのオブジェクトがネストされている。それぞれの値には利用者名・パスワード・プロジェクト名を記述する。

実験環境の構築の際に、これらの項目により利用者と検証実験の識別がされ、適切な実験リソースの割り当てが行われる。各記述項目は、利用者がテストベッドの利用申請を提出する際に任意に決定し、他の利用者と重複がなければ利用可能となり、重複があった場合は変更が要求される。

node 記述部は、割り当てを受け実験ネットワークトポロジに用いられる汎用ノードに対して、利用者が任意の識別子を付けるためのオブジェクトである。

```
{"node":["ノード名 1","ノード名 2","ノード名 3", ...]}
```

”node”を名前として持つオブジェクトであり、この値として汎用ノードに付ける識別子を文字列で記述する。実験ネットワークトポロジに複数の汎用ノードを用いられる場合は配列で記述する。

topology 記述部は、構築したい実験ネットワークトポロジの形状を記述するためのオブジェクトである。

```
{"topology":
    {"ネットワークインスタンス識別子A":["ノード名","ノード名"],
     "ネットワークインスタンス識別子B":["ノード名","ノード名", ... ],
     ...
     }
}
```

”topology”を名前として持つオブジェクトであり、名前にネットワークインスタンス識別子を持ち、値に名前で示されたドメインに所属するノードの名前を記述する。ドメインに所属するノードが複数ある場合は、それらを配列で書き連ねる。

下記に、これらの記述形式から成る実験ネットワークトポロジ設計ファイルの例を示す。

```
{"experiment":
    {"user":"tanabe",
     "pass":"12345",
     "project":"apptest"
    }
}

{"node":["server","switch","client-1","client-2","client-3","client-4"]}

{"topology":
    {"10":["server","client-1"],
     "20":["server","client-2"],
     "30":["server","switch"],
     "40":["switch","client-3"],
     "50":["switch","client-4"]
     }
}
```

experiment 記述部は、利用者名が tanabe・パスワードが 12345・プロジェクト名が apptest であることを示している。node 記述部は、実験ネットワークトポロジの構築に用いる汎

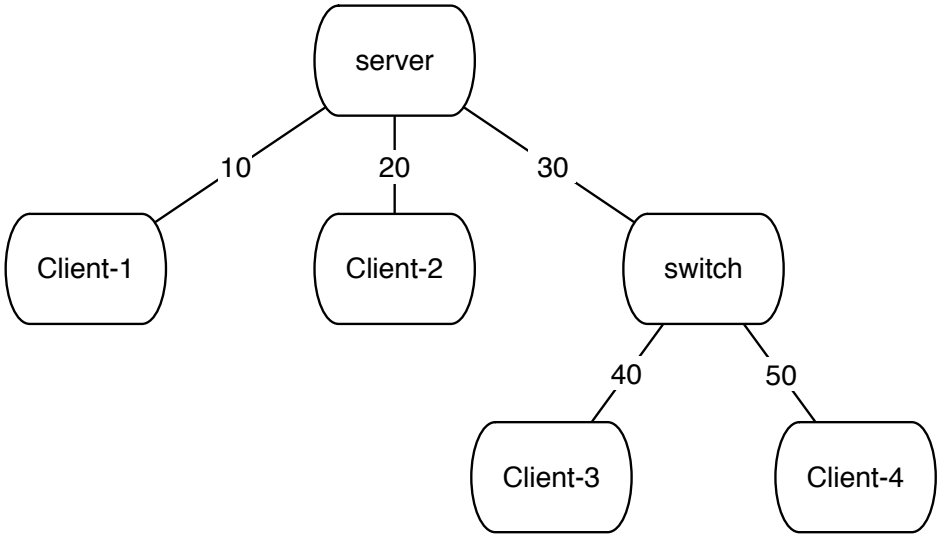

図 5.3: 実験ネットワークトポロジ例

用ノードの数が 6 であり、それぞれの名前が server・switch・client-1・client-2・client-3・client-4 であることを示している。topology 記述部は、server に client-1・client-2・switch が接続され、この switch に client-3・client-4 が接続されることを示している。この実験ネットワークトポロジ設計から構築される実験ネットワークトポロジは図 5.3 となる。

5.3 ExperimentResourceManager

ExperimentResourceManager(ERM) は、テストベッドの検証実験に用いる実験リソースについて管理し、資源の割り当てと設計の具体化を行う際に必要となる情報を与える機能を持つ。ERM が管理する情報は以下の通りである。

1. 利用者情報
2. 利用権限
3. 機器識別子
4. ハードウェア
5. 物理トポロジ

利用者情報は、利用者がテストベッドの利用申請を提出する際に決定される利用者 ID・パスワード・プロジェクト名の情報であり、ERM が持つ User Database(UDB) に保持される。

利用権限は、あるノードがどの利用者に割り当てられているかが示されている情報であり、値として利用者 ID が記述されている。実験リソースの情報が要求された際は、利用権限の値を元に適切な情報が返送される。

機器識別子は、テストベッドがノードを管理するための情報であり、テストベッド運用者が任意に値を定める。

ハードウェアは、各ノードが持つ固有の情報であり、論理ネットワークを作成する際のパラメータとして仕様される。含まれる情報は、実験ネットワークに接続されている NIC の MAC アドレスと管理ネットワーク側のネットワークインタフェースに DHCP から割り当てられた管理用 IP アドレスである。

物理トポロジは、ノードとスイッチの接続状態を表す情報であり、ノードが接続されているスイッチの識別子とそのポート番号である。

5.4 TopologyManager

TopologyManager は、利用者の記述による実験ネットワークトポロジの抽象設計を具体化し、物理トポロジと論理トポロジを明らかにする。初めに、利用者が作成した実験ネットワークトポロジ設計を読み込み、実験リソースの数を算出する。そして、ERM に対して必要な実験リソースの割り当てを要求し、割り当てを受けた実験リソースの各種情報を受け取る。それらの受け取った情報を実験ネットワークトポロジ設計に記述された抽象的な資源の部分に当てはめ、設計の具体化を行う。この具体化によって、テストベッドの物理トポロジと、その上で定義される論理トポロジが明らかになる。

5.5 BEEF-Controller

ネットワークの仮想化および制御を行うために、提案システムの実装に OpenFlow を使用した。OpenFlow[21] は、2008 年にスタンフォード大学など中心に設立された OpenFlow Switch Consortium が提唱し実証実験を行なってきた技術であり、2011 年には規格策定の

ために新しく設立された Open Networking Foundation によって標準化作業が進められている。

OpenFlow の特徴は、これまで単体のネットワーク機器の中に組み込まれていた、

- ネットワークの制御機能
- データ転送機能

を分離したことにある。OpenFlow は、

- 経路制御を行う OpenFlowController(OFC)
- データ転送機能を備える OpenFlowSwitch(OFS)
- OpenFlow コントローラと OpenFlow スイッチがメッセージを交換するための OpenFlowProtocol

の 3 要素から構成される。OFS は OFS からの指示に基づき、条件に適合したフローを処理し、データの転送や処理を行う。この際、OFC と OFS との間の通信は、コントロールプレーンと呼ばれるネットワークを経由して OpenFlow プロトコルによって行われる。OFC は、処理対象となるフローの条件と条件が適合した時の動作であるアクションが含まれるルール群をフローテーブルとして OFS に対して定義する。OFS は、フローキャッシュと呼ばれる領域に、一定時間そのフローテーブルを保持する。ルール条件として利用できるフィールは表 5.1 のようになっており、それに対するアクションとして利用できる処理は表 5.2 のようになっている。

OpenFlow1.0 においては、基本的にはこれらのルールとアクションを利用して、ネットワークを構築する。OpenFlow スイッチで処理する以外にも、コントローラに転送されたパケットはコントローラの機能を利用して自由にパケットを処理し、OpenFlow スイッチに返送することが可能である。つまり、単純なヘッダ書き換えや宛先変更などの処理は OpenFlow スイッチのアクションのみで実現できるが、定義されたフィールド以外の要素で条件判断をしたり、Modify-Field で定義されている以外のヘッダ要素を書き換えたいときにはコントローラで処理をすることになる。この場合は、スイッチからコントローラに対してパケットを転送し、コントローラにて処理を行った後に、どのスイッチに送り返すかをコントローラが決定することができる。

OFC と OFS さんの間にて交換されるメッセージは、OpenFlow プロトコルとして定義されている。メッセージは大きく分類すると、

表 5.1: マッチルール一覧

マッチルール	説明
Ingress Port	パケットの入力があったスイッチの物理ポート番号
Ethernet Source Address	Ethernet の送信元 MAC アドレス
Ethernet Destination Address	Ethernet の宛先 MAC アドレス
Ethernet Type	Ethernet のプロトコル番号
VLAN ID	VLAN 番号
VLAN Priority	VLAN Priority Code Point 番号
IPv4 Source Address	送信元 IPv4 アドレス
IPv4 Destination Address	宛先 IPv4 アドレス
IP Protocol	IP プロトコル番号
IP ToS bits	IP ヘッダの ToS フィールド
TCP/UDP Source Port	TCP/UDP の送信元ポート番号
TCP/UDP Destination Port	TCP/UDP の宛先ポート番号
ICMP Type	ICMP タイプ
ICMP Code	ICMP コード

- OFC から OFS へのメッセージ
- 非同期メッセージ
- 対称メッセージ

の 3 種類に分類され、表 5.3 のメッセージがある。

5.5.1 Beef's Ethernet Equivalent Forwarding(BEEF)

Beef's Ethernet Equivalent Forwarding(BEEF) は、提案システムにおいて作成される仮想ネットワークである。BEEF は、OFS の固有識別子とポート番号の対に BEEF-ID を割り当てることによって定義される。同じ BEEF-ID を持つスイッチポートは同じ BEEF であり、ブロードキャストドメインを形成するグループである。ノードがどのドメインに属するかは、ノードが接続されている OFS の固有識別子とポート番号の対に対応付けられた BEEF-ID によって識別される。図 5.5 は、BEEF によって仮想データリンクを作成した際

表 5.2: アクション一覧

アクション	説明
Forward	パケットを指定したポートに対して転送する 転送先としては ALL : 全てのポート LOCAL : スイッチのスタック CONTROLLER : コントローラ TABLE : フローテブルの Action を実行 IN_PORT : パケットが入ってきたポート も選択できる
Enqueue	QoS を行うために、キューにパケットを入れる
Drop	パケットを破棄する
Mofify Field	パケットの特定のフィールドを書き換える 書き換え処理には Set VLAN ID Set VLAN Priority Strip VLAN Header Modify Ethernet Src/Dst Address Modify IPv4 Src/Dst Address Modify IPv4 ToS bits Modify TCP/UDP Src/Dst Port が利用できる

表 5.3: メッセージ一覧

メッセージ	説明
Feature	スイッチに対して機能の有無を問い合わせる
Configuration	スイッチに対して設定を行う
Modify-State	スイッチのフローテブルを修正する
Read-State	スイッチの統計情報を取得する
Send-Packet	スイッチの特定ポートからパケットを送出する
Barrier	スイッチに対して設定が完了したかを確認する
Packet-in	スイッチがパケットを受信したことをコントローラに通知する
Flow-Removed	スイッチ上のフローの期限切れをコントローラに通知する
Port-status	スイッチのポート状態が変化したことをコントローラに通知する
Hello	コントローラとスイッチ間のネゴシエーションに使用
Echo	コントローラとスイッチ間の死活管理に使用
Vendor	ベンダ独自定義メッセージ

の、BEEF-Controller の動作シーケンスを示している。Ethernet と同等の動作を可能としている。

以下は、BEEF-Controller が受け取る仮想データリンクの作成指示の例である。一行目は、SwitchID-100 という識別子がつけられた OpenFlow スイッチの 1 番ポートを、ドメイン apptest-10 に設定するという指示である。ドメイン名として apptest-10 と記述があるが、これは BEEF-Controller 内部で扱うための名前であり、OpenFlow スイッチで扱う識別子ではないので、仮想データリンクの作成数には影響しない。

```
SwitchID-100, 1, apptest-10
SwitchID-100, 2, apptest-20
SwitchID-100, 3, apptest-30
SwitchID-100, 4, apptest-10
SwitchID-100, 5, apptest-20
SwitchID-100, 6, apptest-30
SwitchID-100, 7, apptest-40
SwitchID-100, 8, apptest-50
SwitchID-200, 1, apptest-40
SwitchID-200, 2, apptest-50
```

OpenFlow コントローラの開発フレームワークである Trema[22] を用い、必要機能の実装を行った。

5.5.2 最短経路探索によるトラフィックループの回避

同じ BEEF に所属するノードが異なるスイッチに接続している場合、フローが通過する経路が明らかになる必要がある。そこで、スイッチ間リンクのメトリックが非負の場合の特定ノード間のフローが通過する経路を求める方法として、提案システムにダイクストラ法 (Dijkstra method)[23] による経路探索の機能を実装した。

始点 s に隣接する接点集合 N(s) を考える。v∈N(s) に対して (s,v)∈E であるので、v には枝 (s,v) を使って到達することができる。そこでまず v の s からの距離を dist(v) とすると、$dist_{temp}$(v):=length(s,v) とおく。v の s からの最短距離を dist(v) とすると、$dist(v) \leq dist_{temp}$ となることは明らかである。ただし、dist(v)>$dist_{temp}$(v) となる可能性も当然ある。言い換えれば、一本のリンクで到達するよりも二本以上のリンクを用いて到達する場合の方がメトリックが少ない可能性がある。

ところが、ここで N(s) のうち、$dist_{temp}$(*) が最小の接点 w に着目すると、dist(w)>$dist_{temp}$(w) とは成り得ないことに気付く。なぜならば、枝 (s,w) 以外を仕様して到達する場合は必ず w 以外の N(s) の接点 v を経由しなければならず、リンクのメトリックが非負であることから、その経路の長さは $dist_{temp}$(v)=length(s,v) より短くなうことはあり得ないからである。よって dits(w)=$dist_{temp}$(w) であることがわかり、w についての最短路を $< s, w >$ に決定することができる。

図 5.6 の例では、まず s に隣接する接点 v_1・v_2・v_3 に対して $dist_{temp}(v_1)$:=length(s,v_1)=2、$dist_{temp}(v_2)$:=length(s,v_2)=7、$dist_{temp}(v_3)$:=length(s,v_3)=3 とするが、$dist_{temp}(v_1)$=2 が最小であるので、s-v_1 間の最短経路の長さは 2 であり、最短路は $< s, v_1 >$ であることがわかる。

w についての最短路が固定されれ b 、次は N(v)∪N(w) について議論を広げることが出来、類似の手順を繰り返していくことで全スイッチへの最短路を求めていくことができる。

これまでの手順で、v_1 までの最短路長が dist(v_1)=2 であることがわかった。すると次に v_1 の隣接点 v_2・v_4・t に対して、直前に v_1 を経由して到達する路の長さを求めることができる。v_4 については、$dist_{temp}(v_4)$:=$dist_{temp}(v_1)$+length(v_1,v_4)=2+1=3 とすればよい。t についても同様に $dist_{temp}$(t):=$dist_{temp}(v_1)$+length(v_1,t)=2+6=8 とする。v_2 については既に $dist_{temp}(v_2)$=7 が設定されているので、新たに求めたリンクの大きさを比較して小さい方

を残す。よって $dist_{temp}(v_2)$:=min$dist_{temp}(v_2)$,dist(v_1)+length(v_1,v_2)=min7,2+4=6 となる。このようにして得られた $dist_{temp}$ の値を比較して、最も小さい端点を選ぶ。この場合 v_3 と v_4 が共に 3 で最小であるが、このような場合にはどちらを選んでも構わない。仮に v_3 を選べば、v_3 までの最短路長が現在の $dist_{temp}(v_3)$ の値である 3 であることが固定される。このような手順を繰り返していくことで最短経路を求める。

提案システムの実装では、隣接する OFS へのリンクのコストを全て 1 としており、ホップ数によって最短経路を導出する。メトリックが同じ経路が複数存在する場合は、各 OFS に設定された固有の Datapath ID の値を使用し、その値が小さい経路を選択する。また、定期的に隣接 OFS とメッセージ交換を行うことで、ある経路の途中の OFS に障害が発生した際、最短経路の再計算を行い、代替経路が存在する場合は切り替えてデータ通信を継続する。

5.5.3 スイッチ間の物理ネットワークトポロジの動的把握

Link Layer Discovery Protocol(LLDP) は、IEEE802.1AB[24] にて規格されたネットワークトポロジとネットワーク上の機器に関する情報を取得する為の隣接検索プロトコル (Neighbor Discovery Protocol) である。LLDP では、イーサネット接続を利用するネットワーク機器が、同じネットワーク上に存在する隣接機器との間で情報を交換し、ネットワークに関する情報を保存する。各ネットワーク機器は、上位レイヤの管理機器に対して、隣接ネットワーク機器間の到達性とネットワーク機器の情報を取得し、伝達する。

1. OpenFlow スイッチに対して Feautures Request メッセージを送信し、そのスイッチが持つポートの一覧を取得する。

2. BEEF-Controler は、OpenFlow スイッチ 1 に対して各ポートから LLDP パケットを送信するように Packet-Out を指示する。

3. OpenFlow スイッチ 1 は、Packet-Out メッセージの指示に従い、LLDP パケットを送信する。

4. OpenFlow スイッチ 1 から LLDP パケットを受信した OpenFlow スイッチ 2 は、Packet-In として BEEF-Controler に LLDP パケットを送信する。

5. BEEF-Controler は、受け取った Packet-In メッセージを解析することで、LLDP パケットが通過したスイッチのどのポート間に接続関係が存在するかを知る。

6. これらの手順を各スイッチに対して行うことで、全ての接続関係を収集し、スイッチが構成するネットワークトポロジを把握する。

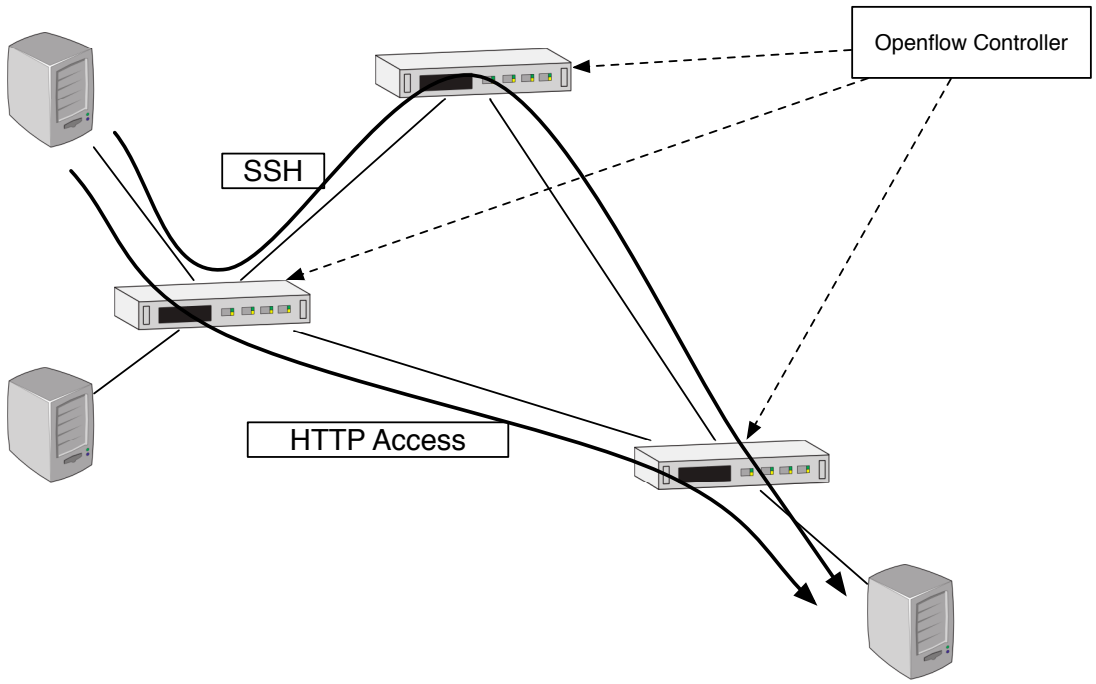

図 5.4: OpenFlow の動作

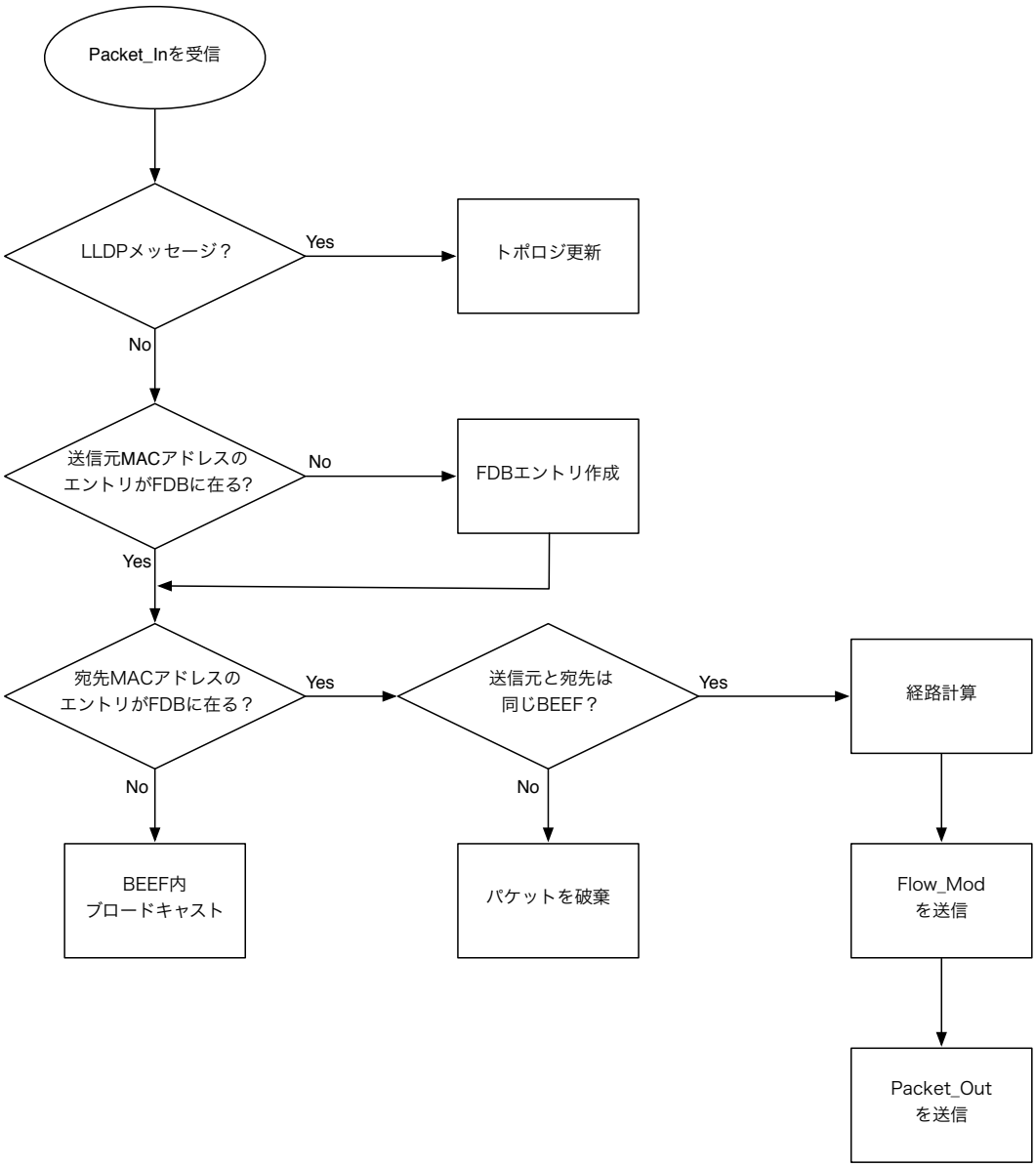

図 5.5: BEEF-Controller の処理手順

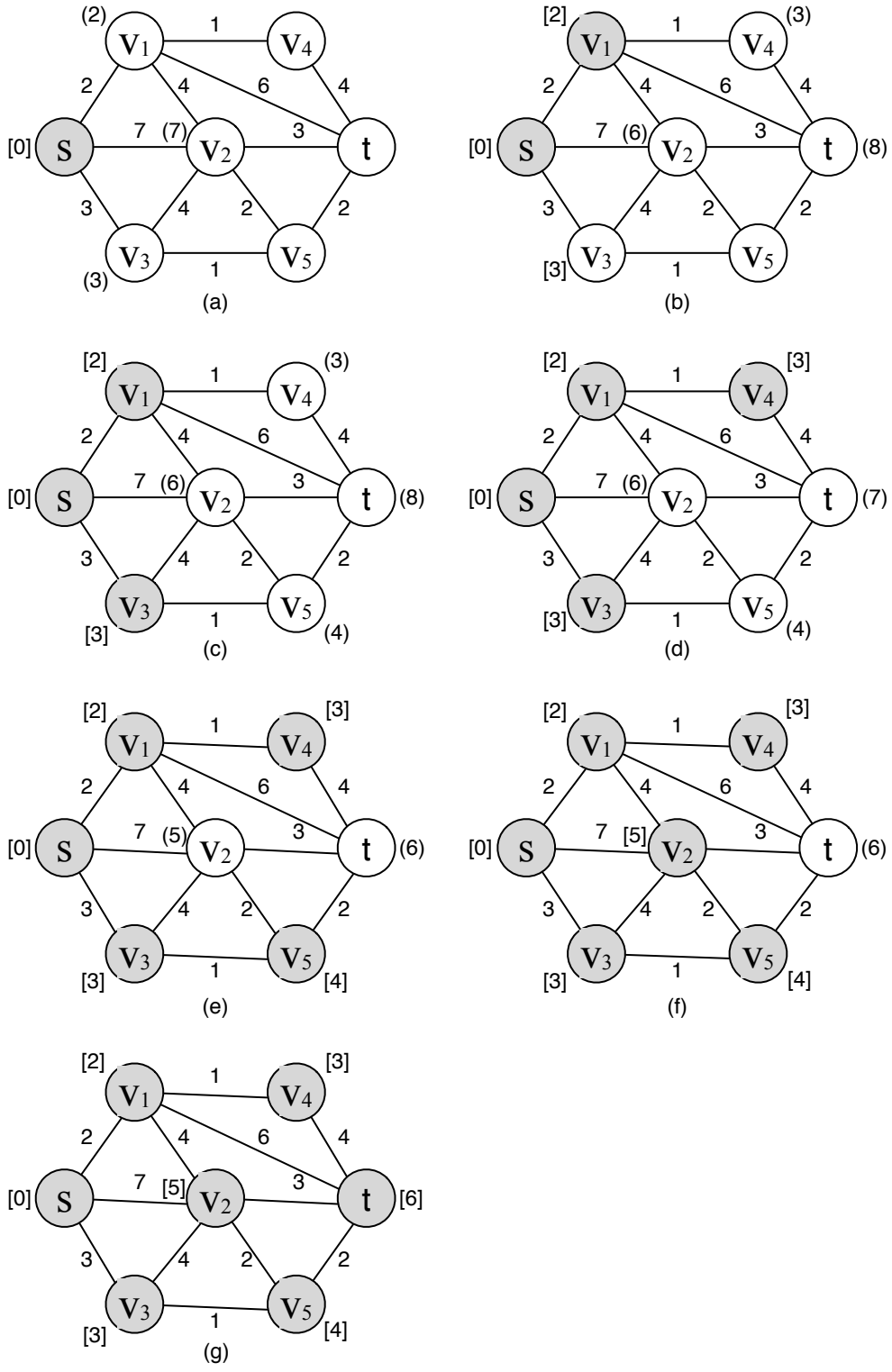
(2)
V1
1
V4
2
4
6
4
[0]
S
7
(7)
V2
3
t
3
4
2
2
V3
1
V5
(3)
(a)
[2]
V1
1
V4
(3)
2
4
6
4
[0]
S
7
(6)
V2
3
t
(8)
3
4
2
2
[3]
V3
1
V5
(b)
[2]
V1
1
V4
(3)
2
4
6
4
[0]
S
7
(6)
V2
3
t
(8)
3
4
2
2
V3
1
V5
[3]
(4)
(c)
[2]
V1
1
V4
[3]
2
4
6
4
[0]
S
7
(6)
V2
3
t
(7)
3
4
2
2
[3]
V3
1
V5
(4)
(d)
[2]
V1
1
V4
[3]
2
4
6
4
[0]
S
7
(5)
V2
3
t
(6)
3
4
2
2
V3
1
V5
[3]
[4]
(e)
[2]
V1
1
V4
[3]
2
4
6
4
[0]
S
7
[5]
V2
3
t
(6)
3
4
2
2
[3]
V3
1
V5
[4]
(f)
[2]
V1
1
V4
[3]
2
4
6
4
[0]
S
7
[5]
V2
3
t
[6]
3
4
2
2
[3]
V3
1
V5
[4]
(g)

図 5.6: ダイクストラ法

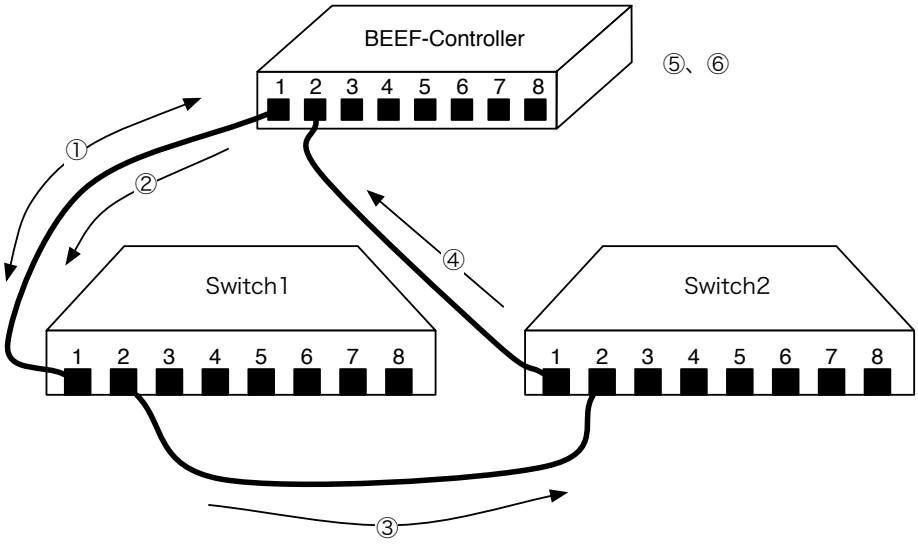

図 5.7: LLDP によるトポロジ把握

第6章　評価と考察

6.1　動作検証

BEEF による実験ネットワークトポロジ構築について、設計通りに動作するかを確認するための動作検証を行った。BEEF を定義する BEEF-Controller には以下の機能を実装した。

- ネットワーク仮想化
- 動的なトポロジ把握
- 最短経路によるパケットの転送

これらの動作を確認するため、動作検証用のテスト構成を作った。構成要素は、5 台の OFS と 5 台のノードである。図 6.1 のように、OFS でループが存在する物理トポロジを構成し、OFS-1 には NodeA・NodeB を、OFS-5 には NodeC・NodeD・NodeE をそれぞれ接続した。ノードのハードウェア情報および物理構成は表 6.1 のようになっている。

このような物理トポロジ構成の上で BEEF を用いた実験ネットワークを作成するために、BEEF-Controller に以下のような BEEF 定義を与えた。

表 6.1: 物理トポロジ構成

ノード名	MAC アドレス	接続 OFS	接続ポート
NodeA	macaddr-A	OFS-1	1
NodeB	macaddr-B	OFS-1	2
NodeC	macaddr-C	OFS-2	1
NodeD	macaddr-D	OFS-2	2
NodeE	macaddr-E	OFS-2	3

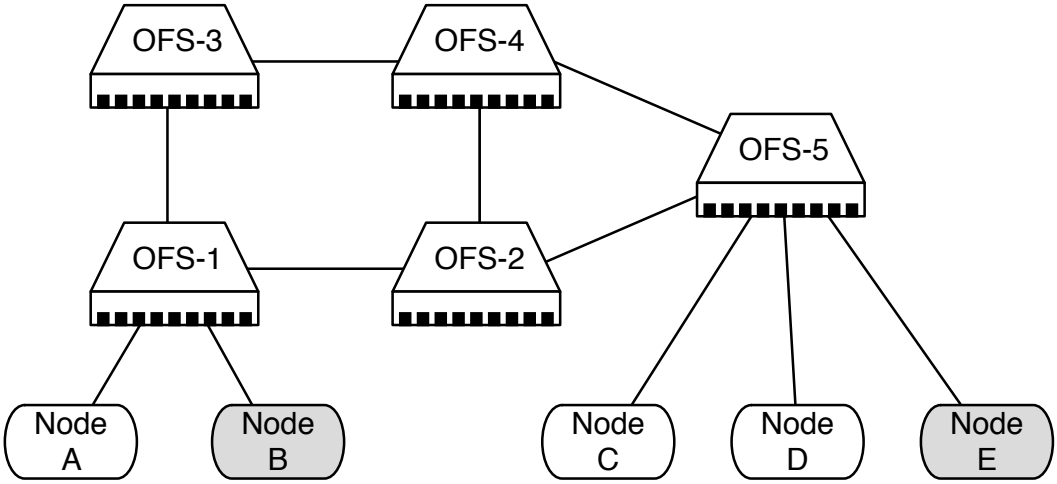

図 6.1: テスト構成の物理トポロジ

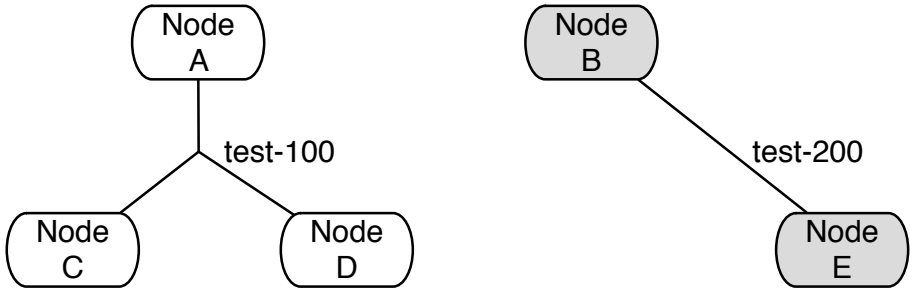

図 6.2: テスト構成の論理トポロジ

```
OFS-1, 1, test-100
OFS-1, 2, test-200
OFS-5, 1, test-100
OFS-5, 2, test-100
OFS-5, 3, test-200
```

BEEF-Controller によって、テスト構成内には 2 つのの BEEF が作成され、それぞれ test-100・test-200 である。test-100 には NodeA・NodeC・NodeD が所属し、test-200 には NodeB・NodeC が所属している。これにより構築された実験ネットワークの論理トポロジは図 6.2 のとおりである。BEEF によって定義されたブロードキャストドメインにそれぞれのノードが所属しており、test-100 と test-200 は論理的に独立した L2 ネットワークとなっている。

表 6.2: 各ノードの IP アドレス

ノード名	IP アドレス	サブネットマスク
NodeA	192.168.0.1	255.255.255.0
NodeB	192.168.0.2	255.255.255.0
NodeC	192.168.0.3	255.255.255.0
NodeD	192.168.0.4	255.255.255.0
NodeE	192.168.0.5	255.255.255.0

次に、各ノードに IP アドレスの設定を施し、L3 レベルの疎通性が確認できる状態にした。各ノードに設定した IP は表 6.3 の通りである。

このようにしてテスト環境を構築し、Address Resolution Protocol(ARP)[25] によるアドレス解決の挙動を観察することで、動作検証を行った。

まず、NodeA から NodeC の IP アドレスを指定して ping を試みる。NodeA は NodeC の 32 ビット IP アドレスを 48 ビットの Ethernet アドレスに変換する必要があり、NodeC の IP アドレスを含んだ ARP Request をブロードキャストする。ブロードキャストなので NodeA が接続されている OFS-1 のポート 1 が所属する BEEF に所属する全てのスイッチポートに ARP Request が転送される。今回の場合は、OFS-1 のポート 1 は test-100 の BEEF に所属しているので、OFS-5 のポート 1 とポート 2 に転送され、その先に接続されている NodeC と NodeD が ARP Request を受信したことを確認した。ちなみに、異なる BEEF である test-200 に所属する OFS-1 のポート 2 に接続された NodeB と、OFS-5 のポート 3 に接続された NodeE には転送されておらず、ARP Request を受信していないことを確認した。

ARP Request を受信した NodeC はユニキャストで ARP Reply を送信する。NodeA が ARP Request をブロードキャストした際に MAC アドレスとスイッチポートとの関連付けが行われ、NodeC からユニキャストで送信される ARP Reply は、NodeA が接続されている OFS-1 のポート 1 に転送される。このようにして、NodeA が ARP Reply を受信したことで、ARP によるアドレス解決が成功していることを確認し、ネットワークの仮想化によって論理的なネットワークが作成できていることを確認した。

また、以上の送受信は、[OFS-1] ー [OFS-2] ー [OFS-5] の経路で行われていることも確認した。これは、BEEF-Controller 起動時から各 OFS が LLDP によって動的に物理トポロジを把握し、その上で転送時に実装したダイクストラ法による最短経路を行った結果であり、正しい動作であるといえる。

表 6.3: 各ノードの通信の疎通性

-	NodeA	NodeB	NodeC	NodeD	NodeE
NodeA	-	×	○	○	×
NodeB	×	-	×	×	○
NodeC	○	×	-	○	×
NodeD	○	×	○	-	×
NodeE	×	○	×	×	-

同様に、全ての End-to-End のノードについて、同様の手順で疎通性を調べた。各ノード間の疎通性は表 6.3 の通りである。同じ BEEF に所属するスイッチポートに接続されているノードは相互に通信することが可能であり、異なる BEEF 間はネットワークが論理的に分離されているので通信は不可能であることが確認できた。

6.2 BEEF のパフォーマンスに関する考察

6.2.1 ハードウェア実装との関係

BEEF のパフォーマンスを考える指標の一つとして、どの程度の数の仮想ネットワークを作成できるかが考えられる。BEEF は End-to-End の接続性を提供するためにフローテーブルの 1 エントリを使用する。すなわち、ある BEEF に所属するノードが N 台の時、N×N のエントリが必要となる。OFS が持つフローテーブルの大きさは、ハードウェアのシステムアーキテクチャと密接な関係がある。OFS の実装方法に関して一般的な方法を下記に挙げる。

- 汎用 CPU と DRAM の組み合わせによるアーキテクチャ
  このアーキテクチャでは、汎用 CPU を用いて、トラフィック処理・パケット処理・ヘッダ情報のマッチング・ヘッダ情報の書き換えが行われる。フローテーブルは、拡張性が高く、低コストな DRAM に記録されるので、何百万ものフローを扱うことができるという長所がある。しかし、フローの検索とマッチングにかかる時間は、他のアーキテクチャと比較して遅く、特にある閾値を越えた場合に急激に性能が低下することが予想される。また、スループットの限界も比較的低く、大規模な構成には不向きであるといえる。

- ネットワークプロセッサーと SRAM の組合せによるアーキテクチャ
  汎用 CPU を使用する場合と異なり、ネットワークプロセッサーは、各種のデータプレーン機能を高速に処理することが可能である。その際の高速のメモリアクセスに対応する為に、ハードウェアには SRAM が搭載されることが多い。SRAM は、DRAM によりも高速にフローの検索やマッチングを行うことができるが、比較的高価で高密度化も困難であるという短所もある。

- ネットワークプロセッサーと TCAM を組合せたアーキテクチャ
  ネットワークプロセッサーが搭載されている点は上記と同じであるが、この実装では Ternary Content Addressable Memory(TCAM) が使用される。TCAM は DRAM や SRAM と比較して非常に高速であり、高いスループットが必要となるハードウェアに搭載される。TCAM は非常に高速であるが、SRAM の数十倍の価格であり、消費電力も大きいという特徴がある。よって提供される TCAM の容量は小さいことが多く、フローテーブルのサイズが著しく制限される。

よって、BEEF が作成できる仮想ネットワークの数は、OFS の実装に大きく依存するといえる。テストベッドにおいて実施される検証実験の規模や内容によって、設備の構成を決定するべきである。フローテーブルの容量と処理の速度はトレードオフの関係にあり、使用する条件にあわせた検討が必要となる。

6.2.2 ノード配置との関係

前小節において、BEEF 自体のパフォーマンスは OFS の実装に大きく依存すると結論付けたが、テストベッド全体で作成できる仮想ネットワークの数は、テストベッドの運用によって改善できると考えた。そこで、実験者に実験リソースとして割り当てられるノードの配置によって、OFS のフローテーブルを効率的に使用することを検討する。

ノードの配置については、2 通りの検討を行い、それぞれ下記の通りである。

- ノードが集中している場合
- ノードが分散している場合

ノードが集中している場合は、図 6.3 のように、1 つの OFS に、実験ネットワークを構成するノードが全て接続されている場合である。このような場合に、全てのノード間の疎

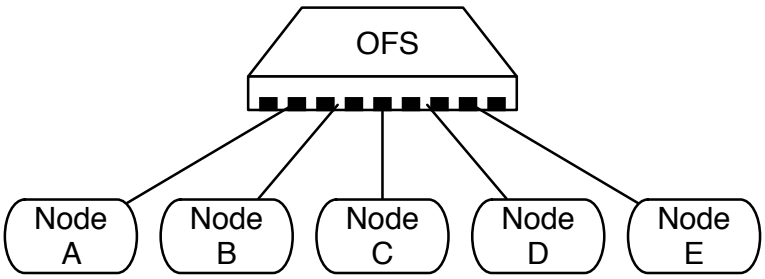

図 6.3: 使用するノードが集中している場合

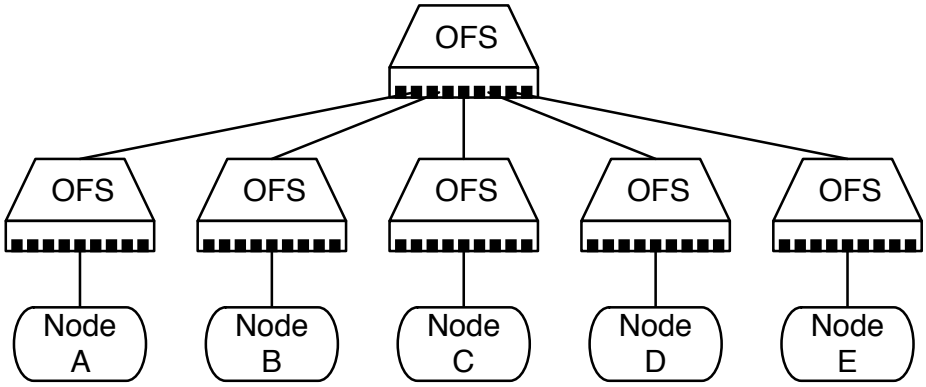

図 6.4: 使用するノードが分散している場合

通性を提供するフローエントリを定義すると、その数はノード数の累乗で 25 である。25 のフローエントリが、この OFS に設定され、実験ネットワークが構築される。

次に、ノードが分散している場合について考える。ノードが分散している例として、図 6.4 のように、複数の OFS にノードを接続した。複数の OFS 間で通信を行うために各 OFS を束ねる上位の OFS を設置し、ツリー型の物理トポロジを構成している。このような場合は、全ての OFS に全てのノード間の疎通性が定義される。すなわち、この構成の全体で、25×6=150 のフローエントリが定義される。

これらのことから、実験者に提供するノードは、分散させるとフローテーブルの使用効率が悪いことがわかる。より多くの仮想ネットワークを作成するためには、効率のよいフローテーブルの運用が必要となるので、可能な限り同じ OFS に接続されているノードを実験リソースとして割り当てる運用が望ましい。

6.3 仮想インターフェースへの対応

テストベッドで、ノードとして用いられている計算機のハードウェア構成によっては、構築したいネットワークトポロジに対して物理的なネットワークインタフェースの数が不足していることが考えられる。このように、物理ネットワークインタフェースが少ない計算機で構成されているテストベッドにおいても、提案システムを使用するためには、仮想インタフェースに対応する必要がある。仮想インタフェースは、OS に機能によって 802.1q タギングを利用して物理インタフェースを仮想的に多重化する。仮想インタフェースに対応した BEEF-V を提案し、BEEF をベースに機能の追加を行った。追加した要素は以下の 2 点である。

- マッチルールの追加
- 802.1q タグの付け替え

BEEF-V では、仮想インタフェースの識別をするために、BEEF-Controler のマッチルールに VLAN-ID を追加した。OpenFlow スイッチにパケットが入った時、スイッチ識別子とスイッチのポート番号とフレームに付加されている VLAN タグの組み合わせによって所属している BOMAIN を識別する。BOMAIN が同じだった時の動作は BEEF と同じである。OpenFlow スイッチがパケットを受け取ると、宛先と送信元の所属 BEEF を確認し、同じ BEEF に所属する場合は転送を行う。転送を行う際に、宛先の仮想ネットワークインタフェースに割り当てられている VLAN-ID に対応するようにフレーム内のタグを付け替える。これによって適切な宛先への転送が実現する。

第7章　おわりに

本研究では、利用者が制限を意識することなく自由に検証実験で用いるネットワークを設計し、その構築を可能とすることを目的とし、トラフィックフロー制御によるネットワーク構築手法を提案した。提案手法では、抽象度の高い記述方式によって、実験施設のファシリティーや機器の設定項目を意識せずに検証実験に用いるネットワークを設計し、その設計を反映してネットワークを構築する。利用者の設計を反映し、ネットワークを構築するネットワーク制御方式として、Beef's Ethernet Equivalent Forwarding(BEEF) を提案した。BEEF は、ネットワーク機器のポートの集合でブロードキャストセグメントを定義し、パケットのヘッダに含まれる情報を用いたトラフィックフロー制御によってその定義を適用する。これによってネットワーク制御技術に定義された固定的な識別子に依らない論理ネットワークの作成が可能となり、従来の識別子の数による制限を無くし、高い拡張性を実現した。提案手法を用いることにより、大規模汎用テストベッドの需要増加による利用多重度と、構築されるネットワークの規模の拡張性が向上し、自由度の高いネットワーク構築が可能となる。

謝辞

本研究を行うにあたり、主指導教官である篠田陽一教授には多大なる御指導を頂きました。深く感謝し、心からお礼を申し上げます。

研究の節目節目において貴重な御助言や指摘を賜った丹康雄教授、宮川晋客員教授、知念賢一特任准教授、宇多仁助教、副テーマを御指導していただいた宮地充子教授に感謝を申し上げます。

北陸 StarBED 技術センターの三輪信介センター長、宮地利幸研究員、太田悟史研究員、中井浩研究員、Razvan Beuran 研究員には StarBED の利用、研究に関する議論・情報共有など多くの御助力を頂きました、感謝を申し上げます。

WIDE プロジェクト DeepSpace One WG および NERDBOX フリークス WG の櫨山寛章助教、宮本大輔助教、榎本真俊氏、には定期的な WG の活動において多くの御助言を頂きました、感謝を申し上げます。

本研究室の安田真悟氏、高野祐輝博士、井上朋哉博士、川瀬拓哉氏、立花一樹氏には日常より熱い御指導と御鞭撻を頂きました、心からお礼を申し上げます。

本研究室の博士後期課程である明石邦夫氏、Muhammad Imran Tariq 氏には研究に関して多大な御助言を頂きました、心からお礼を申し上げます。

本研究室の博士前期課程である鍛治祐希氏、村上正太郎氏、大野夏希氏、向井雄一朗氏、岩橋紘司氏、加藤邦章氏、成田佳介氏、向井康貴氏、岩本裕真氏には、研究に関して活発な議論で貢献していただき、研究生活を送る上でもお世話になりました、心からお礼を申し上げます。

最後に、研究と生活を支えてくれた家族へ心から感謝致します。

参考文献

[1] 鈴木未央, 櫨山寛章, 門林雄基. AnyBed : クラスタ環境に依存しない実験ネットワーク構築支援機構の設計と実装. 情報処理学会研究報告, 2004.

[2] 宮地利幸, 中田潤也, 知念賢一, BEURAN Razvan, 三輪信介, 岡田崇, 三角真, 宇多仁, 芳炭将, 丹康雄, 中川晋一, 篠田陽一. StarBED : 大規模ネットワーク実証環境. 情報処理, 2008.

[3] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 2002.

[4] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 2003.

[5] R. Munoz, C. Pinart, R. Martinez, J. Sorribes, G. Junyent, and A. Amrani. The adrenaline testbed: integrating GMPLS, XML, and SNMP in transparent DWDM networks.

[6] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 2002.

[7] Joseph D. Touch, Yu-Shun Wang, Venkata Pingali, Lars Eggert, Runfang Zhou, and Gregory G. Finn. A global x-bone for network experiments. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, 2005.

[8] T. Lehman, J. Sobieski, and B. Jabbari. Dragon: a framework for service provisioning in heterogeneous grid networks. *Communications Magazine, IEEE*, 2006.

[9] 北陸 StarBED 技術センター. http://starbed.nict.go.jp/.

[10] Linux-VServer.

[11] Toshiyuki Miyachi, Ken-ichi Chinen, and Yoichi Shinoda. StarBED and SpringOS: large-scale general purpose network testbed and supporting software. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, 2006.

[12] Yanyan Wang, Matthew J. Rutherford, Antonio Carzaniga, and Alexander L. Wolf. Automating experimentation on distributed testbeds. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005.

[13] IEEE Draft Standard for Local and Metropolitan Area Networks, Virtual Bridged Local Area Networks, Amendment 4: Provider Bridges. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1638125.

[14] IEEE Draft Standard for Local and Metropolitan Area Networks, Virtual Bridged Local Area Networks, Amendment 6: Provider Backbone Bridges. http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf.

[15] 宮地利幸. 大規模実証環境の実現と実験支援によるネットワークサービスの検証技術. 北陸先端科学技術大学院大学博士論文, 2007.

[16] Quagga Routing Suite. http://www.nongnu.org/quagga/.

[17] Open vSwitch. http://openvswitch.org/.

[18] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, EECS Department, University of California, Berkeley, 2009.

[19] 伊藤未明. 国内データセンターネットワークインフラストラクチャ市場 2008 年の分析と 2009 年〜2012 年の予測. 2009.

[20] Introducing JSON. http://www.json.org/.

[21] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 2008.

[22] Trema:Full-Stack OpenFlow Framework in Ruby and C. http://trema.github.com/trema/.

[23] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959.

[24] Station and Media Access Control Connectivity Discovery, IEEE Std 802.1AB. http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf.

[25] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826 (Standard), November 1982. Updated by RFCs 5227, 5494.