L1Patch 応用ネットワークテストシステム
試験結果レポート

2015/11/13

内容

1 用語

表 1-1 用語一覧

用語	略語・省略表記	分類	意味
沖縄オープンラボラトリ	OOL	一般	(略語定義) 一般社団法人 沖縄オープンラボラトリ https://www.okinawaopenlabs.org/
ネットワーク	NW	一般	(略語定義)
L1 パッチ (L1patch)		プロジェクト	一般的なネットワーク用パッチパネル(特定の物理結線を集中して付け替えるための機構)。本プロジェクトでは、「ネットワーク用パッチパネルと同等のことを行えるシステム」として使う。
OpenFlow	OF	一般	(略語定義)
OpenFlow Switch	OFS	一般	(略語定義)
OpenFlow Controller	OFC	一般	(略語定義)
OF パッチ（OFpatch)		OOL	OOL で実装している、OpenFlow を使った L1 パッチパネル機能を持つプロダクト
Firewall	FW	一般	(略語定義)
Load Balancer	LB	一般	(略語定義)
ディスパッチャ (dispatcher)		プロジェクト	L1patch と同じ。初期モデル検討の中では L1patch という名称ではなく、任意の物理箇所にパケットを送出するものという意味でディスパッチャという語を使っていた。今回の PoC 実装のプログラム中でも使用している。
ジェネレータ (generator)		プロジェクト	「ネットワークに対して行いたいテスト」を実行するソフトウェア。ネットワークのテストで行うことは、一般的に、パケットを吐いてその応答を観測するものなので「ジェネレータ」と記載している。NW テストのためのトラフィック(パケット)生成機能/そのためのアプリケーション。
テスト対象 NW		一般	検証環境で構築する、動作確認を行いたいネットワーク。複数のネットワーク機器から構成される。
テスト対象機器 (Device Under Test)	DUT	一般	テスト対象となる個々の機器。テスト対象 NW を構成するいずれかひとつの機器。
Mininet		一般	OSS のネットワークテスト用ツール
Open vSwitch	OVS	一般	Linux 上で動作する L2 スイッチ/OFS 実装
ワイヤ(wire)		プロジェク	L1patch によって実現されるデバイス間接続(L1patch が論理的に実

		ト	現する結線)
専有モード (exclusive mode)		プロジェクト	ワイヤ種類のひとつ。物理ポート単位で 1 対 1 のマッピングを行うワイヤ。テスト対象 NW 機器(DUT)間ポートを接続し、テスト対象 NW の物理トポロジを構成するために使用する。
共有モード (shared mode)		プロジェクト	ワイヤ種類のひとつ。MAC アドレス単位でマッピングを行うワイヤ。ひとつの物理ポートに複数のテスト用ノードを配置するために使用する。

2 はじめに

2.1 本書の目的

本書の目的は、本プロジェクト(L1patch 応用ネットワークテストシステムプロジェクト: 以降 L1patchPJ)の企画意図、プロジェクトとしての目的、今回の実証実験(PoC)のターゲット、PoC 実行内容を一覧することである。また PoC で得られた結果と PoC およびプロジェクトとしての目的に対する考察を行い、目的に対する今回の取り組みの評価を行う。

本書では、検証内容や PoC で製作した各種ツール類についての技術詳細は扱わない。技術面の詳細(設計情報やツールの使用方法など)については「技術レポート」を参照すること。

2.2 本プロジェクトの概要と目的

情報システムに求められる拡張性、俊敏性の要求から、システム基盤は大規模化・複雑化してきている。こうした複雑なシステムの制御や管理を人力で行うことは非常に難しく、構築や制御などのオペレーションはソフトウェアによって行うようになってきている。こうした考え方を表す用語として、ソフトウェア定義によるネットワークの構築や制御(SDN: Software Defined Networking)だけでなく、インフラ全体(SDI: Software Defined Infrastructure)、情報システムすべて(SDx: Software Defined Anything)などが提唱されている。

本プロジェクトは、情報システムを構成する要素のうち、ネットワークを対象としている。ネットワークを構成する要素がソフトウェアによって定義されることで、ネットワークの構築や制御の自動化システムやネットワークの運用管理システムの開発・実装は、より高度・高速に行えるようになろうとしている。ただし、そうしたシステムの実装やオペレーションにあたっての動作確認(テスト)については、まだ人手によって行われており、本来ソフトウェア定義によってオペレーションすることで期待していた迅速性や拡張性のネックになっていると考えられる。

ネットワークの構築や制御を行うシステムの開発・構築にあたっては、「作る」-「作ったものをテストする」というサイクルが必要になる。作った・操作したネットワークが本当に狙ったとおりに動作しているかどうか、投入した情報によって行いたかった通信が実際に実現されているか、手作業で確認していては開発・構築のスピードは上げられない。ネットワークの構築・制御がソフトウェアによって定義できるようになるのであれば、それによって実現されるネットワークそれ自体のテストもソフトウェアによって定義され、同等のスピードで「作ったもののテスト」が行えるようになる必要がある。

本プロジェクトの目的は、ネットワークの動作テストとして(人力で)行われている作業をソフトウェアによって定義可能にする一手法と実装を検討し、その有効性を調査することである。

- 仮説: ネットワーク(あるいは情報システム基盤全体)はソフトウェアによって定義・制御されるようになる。多数の拠点・多数のノードを持つ大規模なネットワークを高速に制御してゆくシステムに対して、「制御された結果得られたネットワークの動作」をテストするためのシステムが必要になる。
- 目的: 「ネットワークに対する(テストのための)操作」を記述可能にするシステム、これまで「現地・現物」が求められ、手作業に頼らざるを得なかった作業と同等のことが可能になるシステムを構築する。それによって、ネットワークの構築、構成変更、動作確認という一連のテストサイクルから人手による作業を排除する。

2.3 期待される効果

「ネットワークに対するテスト操作」が記述でき、自動実行できるようになると、以下のような効果が期待できる。こ

こにあげていることは、自動化と属人性の排除に伴う一般的な効果であるが、現状ネットワークに求められる物理的な特性(2.4 節参照)からこうしたことが実現できていない。人手による作業を排除することで、より早く、安定した、品質の高いサービスの開発・提供が可能になることを期待できる。

- 物理・論理基盤の動作チェックをイベントごとに自動実行できる。
  - 環境の変化(追加・変更・削除)に対するトラブルの早期検出
  - サービスの追加や変更への追従
    - サービスオーケストレータなどと連動した試験: 基盤 CI
    - ハードウェアや OS の更新に伴う機能変化など、「実体」の検証・テスト作業のコスト削減
- 基盤に対する検証・作業前後チェックで可能なことの拡大
  - より速く、ミスやバグのリスクを減らしながら、操作の実行やサービス機能開発ができるようになる。
  - 迅速に、信頼性の高いサービスを提供できるようになる。
- 複雑で人手ではできないチェック作業の実行可能性
  - 組み合わせ爆発に対処するためのポリシベーステスト (ポリシ定義して個々のテストケースは自動生成)
  - 特に高レイヤ通信のテストで期待
- テスト作業のノウハウ継承・人による作業品質のばらつきの回避(個人の知識やスキルに基づく判断のばらつきの回避・属人性の排除)

2.4 ネットワークのテストにおける課題

2.4.1 ネットワークの役割

ネットワークのひとつの役割として、情報システムにおける計算機ノードの「物理的な位置(配置)の抽象化」がある。計算機は、ネットワークで接続され、IP アドレスによって計算リソースやその上のサービスを利用できるようになることで、利用者からは物理的な位置が隠蔽される。ネットワークで接続されることで上位のシステムや利用者からは物理的にどこに何があるかを気にせずに[1]利用することができるようになる。(図 2-1)

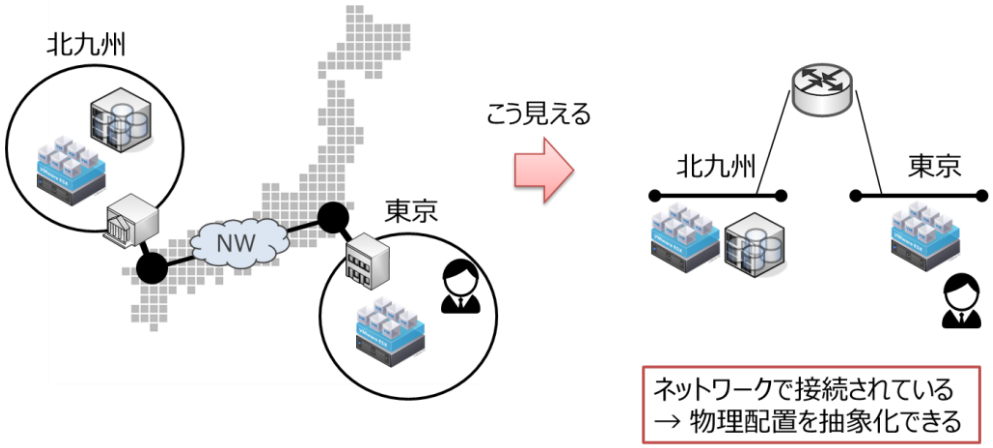


図 2-1 ネットワークの役割

[1] 通信遅延を気にするようなサービスやアプリケーションではこの限りではないかもしれない。ただ、それでもほとんどの計算機リソースはネットワークを介して利用するアーキテクチャになっているだろう。情報システム、それを構成する計算機リソースはネットワークを介してアクセス・オペレーションするアーキテクチャになっており、ネットワーク(がもたらす接続性)はより重要になってきている。

情報システムは、ネットワークによって物理配置が隠蔽される。したがって、ネットワークそれ自体は、物理的にどこに何があるのかを考慮した上で設計・構築・運用されなければならない。ネットワーク自身はシステムを構成する要素の物理配置の制約から逃れられない。そうした「構成要素の実体」を接続し、構成要素間の通信の実現・制御をすることが求められる。

2.4.2 従来のネットワークにおける「テスト」の課題

ネットワークはシステム構成要素の物理的な配置を抽象化する。したがって、ネットワーク自体がシステムの物理的拡張に耐えられるように作られる必要がある。また同様に、特定の単一障害によってシステム全体が影響を受けることがないよう冗長性も要求される。そのため従来のネットワークは、自律分散制御によって構築されてきたが、これはシステム全体の挙動や状態の把握にしにくさ、制御の複雑さを招く要因でもあった。(図 2-2)

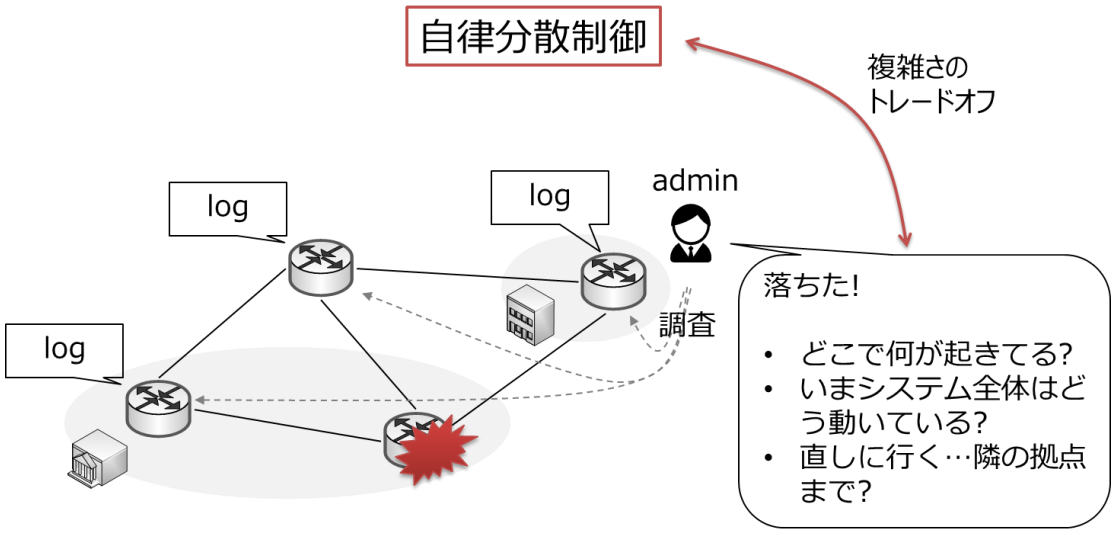


図 2-2 ネットワークの複雑さ

近年ではそれに加え、システム全体に仮想化技術が導入され、計算機リソースを複数の仮想環境として分割して(あるいは逆に、複数の計算機リソースをひとつの仮想環境として統合し)、システム全体の収容効率や利用効率を高めることが一般的になった。ネットワークに関してみると、仮想化技術の導入によって、物理的な構成(ネットワークトポロジ)と論理的な構成との分離がおこり、システムはより複雑になってきている。(図 2-3)

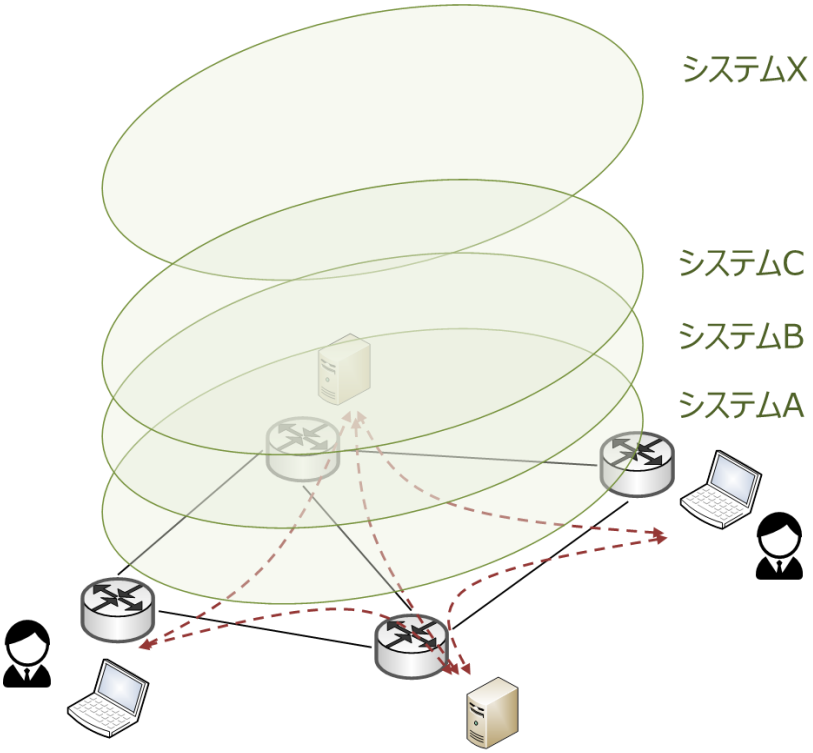


図 2-3 仮想化による複雑さの増大

ネットワークを構築した場合、こうした拡張性や冗長性、そのほかの機能・非機能の要求を満たしているかどうかを確認する必要がある。通常、以下の点を組み合わせた上で、当初想定していたとおりの動作が実現でき、通信要求を満たすかどうかを確認する。

- 物理構成・物理配置の考慮
  - 拠点数の変化
  - 機器数(NW 機器、サーバなど計算機リソース)の変化
  - 機器の種別、OS の種別、バージョン、使う機能などの組み合わせ
  - 物理的な障害に対する通信経路制御
    - ネットワーク上で発生するイベントやそれに対するネットワーク全体の状態変化: 正常時の通信・障害時の通信、イベントに対する状態遷移とそのときのトラフィック影響
- 論理構成、サービス要件の考慮
  - セグメント数の変化
  - サービストラフィック(フローの種別):アプリケーション要件
  - 仮想化された計算機リソースの操作・変化

ネットワークのテスト項目は、これらの項目と物理的な実体配置を考慮した組み合わせになる。たとえば拠点ごとに必要なアプリケーション通信が問題なく動作するかどうか・拠点ごとの障害時の経路切り替えが正しく動くかどうか、など。仮想化技術の導入によってテストで考慮すべき組み合わせ項目はさらに増えている。また、短時間で・動的に・頻繁にリソース配置やシステムの構成が変化するようになってきており、テスト実行の時間的要求もより厳しくなってきている(短時間でたくさんテストをすることが求められる)。

2.5 課題に対する対処・ネットワークのテスト操作とモデルの検討

「いままで現地・現物に対して行われていたテスト操作」を集中制御で行えるようにするために、まずネットワークのテストをどのようにとらえるかを整理する。ネットワークの持つ機能に対して、テストとして行われることを図示すると図 2-4 のようになる。

本プロジェクトは特にデータプレーンに対するテストに着目する。なぜなら、データプレーンの機能は、最終的なサービス利用者(ネットワーク上にのる情報システムの利用者)に「直接見える」ネットワークの機能であり、もっとも物理配置を考慮しなければいけない(特定の箇所や機材を経由する通信フローを考慮しなければいけない)要件だからである。マネジメントプレーンの機能もコントロールプレーンの機能も、最終的に望んだ通信をデータプレーン上で実現するためにある。本プロジェクトでは、データプレーンのテストを行うことで適切な制御や管理が行えているかどうか、ネットワーク利用者に見える機能レベルでのテストを行うことを考える。

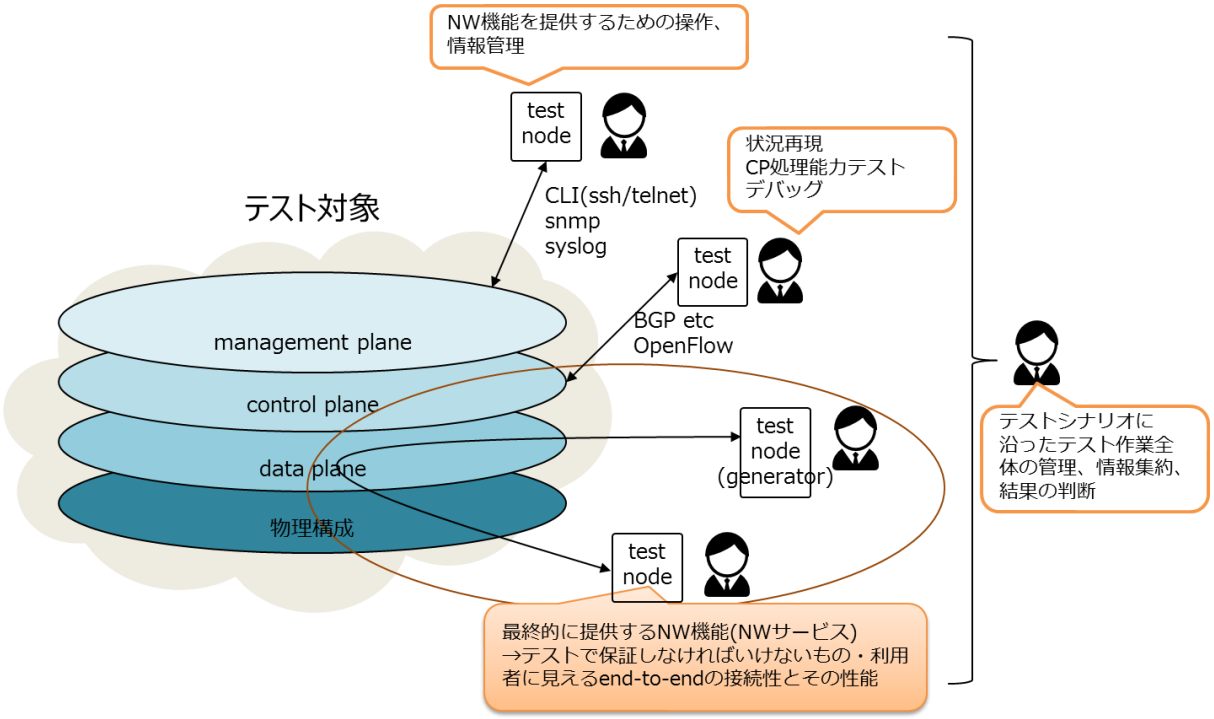


図 2-4 ネットワークのテストの分類

データプレーンのテストに対して、従来行われていた、現地・現物の配置パターンを加味したテストと同等のことを行えるようにする(図 2-5)。そのために、ネットワークテスト操作の抽象化を行う構成要素の機能として図 2-6 のようなモデルを考える。本プロジェクトでは、図 2-5 のように、テスト対象とするネットワークに対しての操作(テスト操作)を抽象化するためのシステムを”L1patch”と呼んでいる。

- Generator: データプレーンのテスト用トラフィック(パケット)を生成するためのツール
- Dispatcher: Generator が生成したトラフィックを、テスト対象の任意の物理箇所へ送り、受信するための機構
- Observer: テスト対象への/からのトラフィックをミラーして観測するための機構

機能としてはこれらの機能が考えられるが、本プロジェクト(今回の実証実験)では主に dispatcher の機能実装とを行うこととし、テストアプリケーションとしては最も基本的な L3 通信(ICMP)テストの自動化を中心に検証と評価を行った。

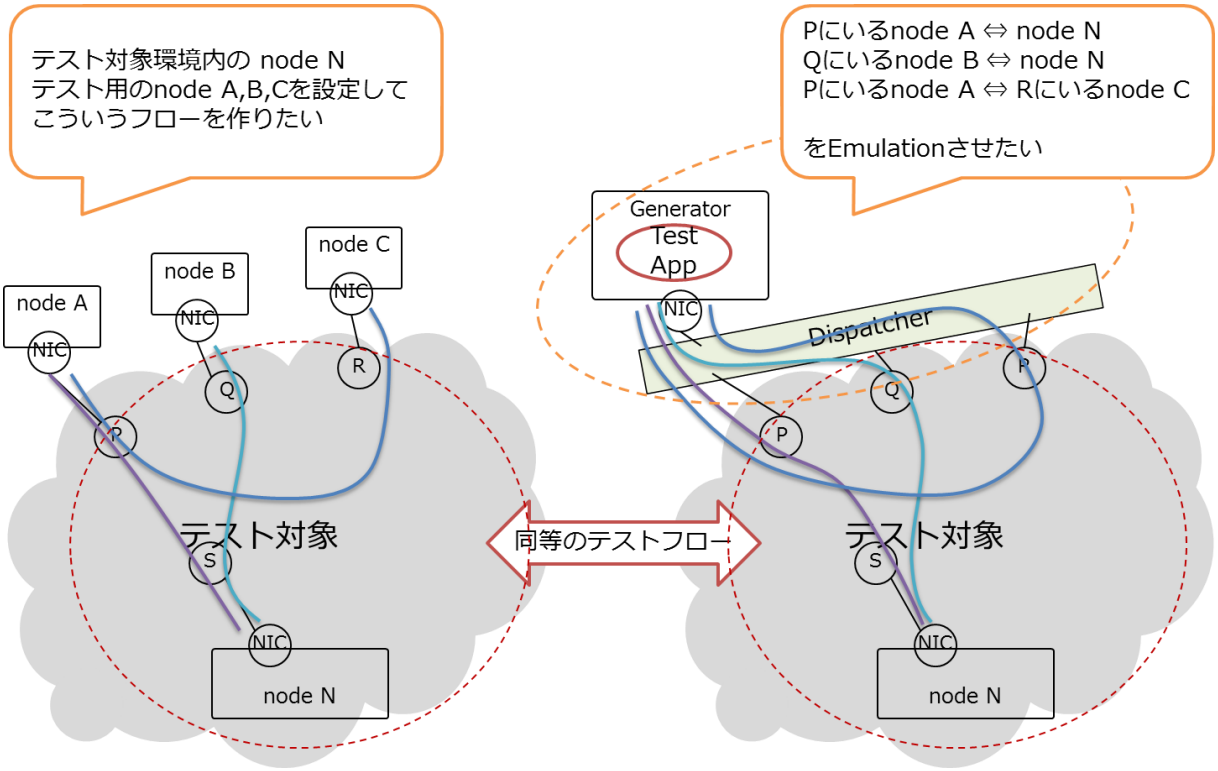


図 2-5 L1patch によるネットワークのテストのイメージ

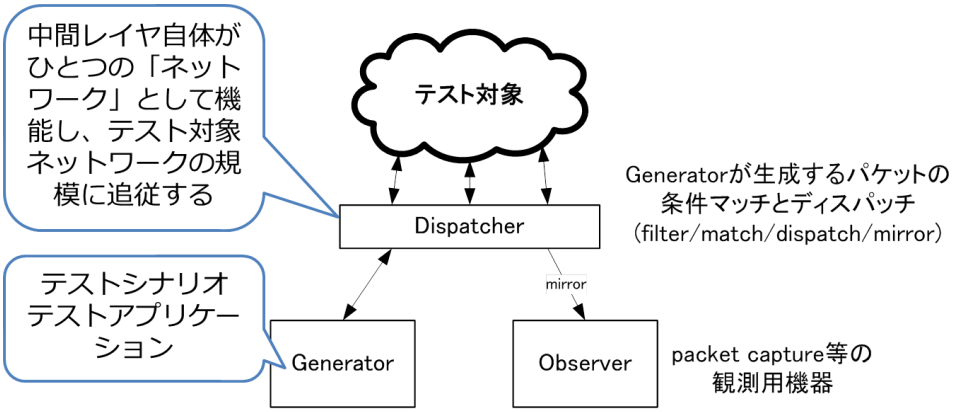


図 2-6 ネットワークオペレーション抽象化のモデル

3 PoC のターゲット検討

3.1 PoC 方針

方針として: テストユースケースのヒアリングやターゲットの絞り込みを行う際に、大きく方針を決めるための観点として以下の点をあげた。(以下の観点によって PoC として見るべきことの方向性が大きく変わると考えた。)

- テストフェーズ
  - 本番前のテスト: NW 機器単体・複合、多数のテストパターンの網羅。本番前に可能な限りたくさんのテストを繰り返して問題点を洗い出し、本番運用後に問題が起きるのを防ぐことを目的とするテスト。
  - 本番後のテスト: 本番環境・運用開始後(運用中)のトラブルシュート、運用中に起きている問題事象の

検出、原因・影響調査を目的とするテスト。

- テスト対象レイヤ[2]
  - 高レイヤ: アプリケーション寄りの通信要件テスト
  - 低レイヤ: ネットワークの物理構成・論理構成に関するテスト

本プロジェクト・今回の PoC としては、その目的・企画意図(2.1 節参照)から以下のように方針を設定した。

- 本番前のテストを対象とする。
  - 繰り返し作業が多い、つまりテストパターンが多い。商用化(サービスイン)前の試験では、行われるテストがある程度単純化・定型化されることが多い。したがって自動化しやすく自動化の効果を出しやすい。
- 低レイヤのテストを対象とする。
  - 物理・論理のネットワーク構成パターン網羅など、従来のネットワークのテストにおける課題設定に対してどのように対処するかという点が当初の目的としてある(2.4 節参照)ため。ネットワークの構成を考慮した上でどのようにテストパターン網羅を行うことができるか、特に物理構成の拡張やパターン網羅をどのように考えるかに注目する。

3.2 テストユースケース分類とターゲット選択

PoC ターゲット設定前におこなったユースケース検討をもとに、「ネットワークのテスト」として行われる作業のうち代表的なものをリストアップし、テストのアプリケーションとして・テストオペレーション(テスト実施時の作業・操作として)何ができているのか(できていないのか)を検討した(図 3-1,図 3-2)。今回の PoC ではオペレーションに着目(3.1)して、効果が高いと思われるユースケース選択をターゲットとして選択している。

- アプリケーションとしては最も基本的、かつテストの最初に実行される疎通試験(L3/ICMP での通信機能試験)を選択する。(図 3-1)
- オペレーションについては、最終的にシナリオベースのテスト(テスト実行・障害発生模擬・再テスト実行・結果比較といったネットワーク構成変更を含むテストシナリオ)を自動化することを PoC のゴールとした。そのために、機能別にいくつかのステップに分けて実験を行うこととした。(ステップについては 4.2 節参照。)

[2] テスト対象ネットワークのデータプレーン上で実際に通信できるかどうか、という観点のテストを考える(2.5 節参照)

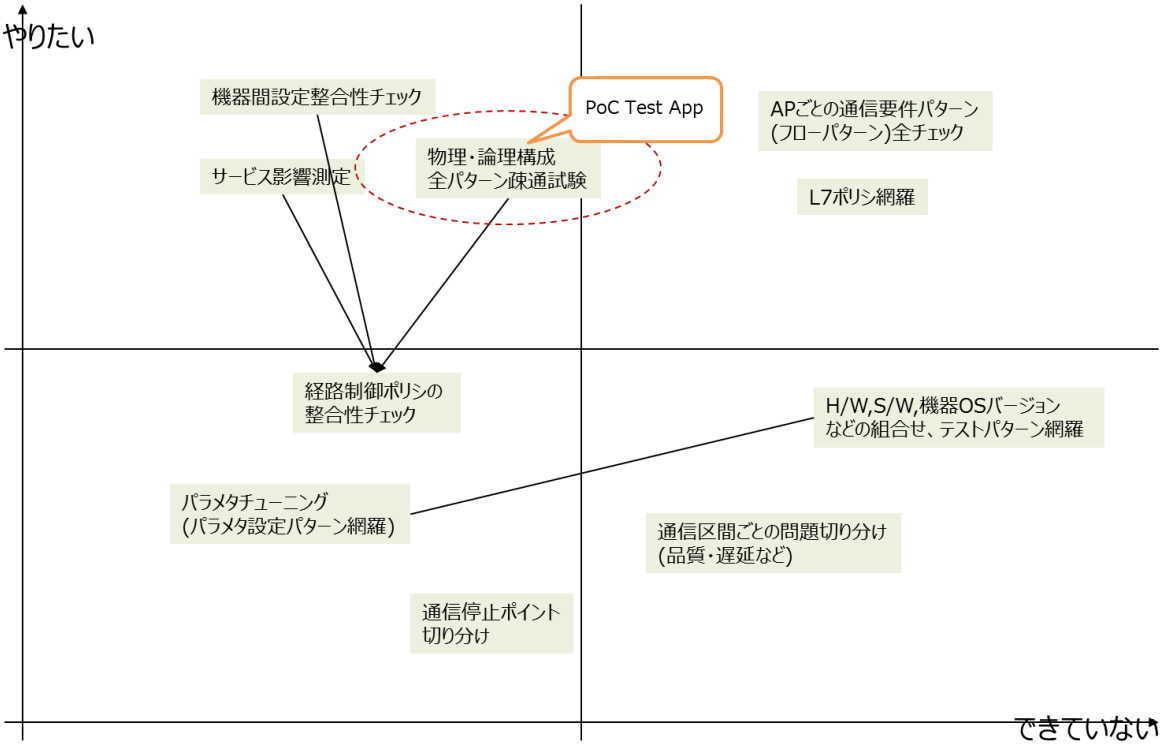


図 3-1 テストアプリケーション視点の PoC ターゲット選択

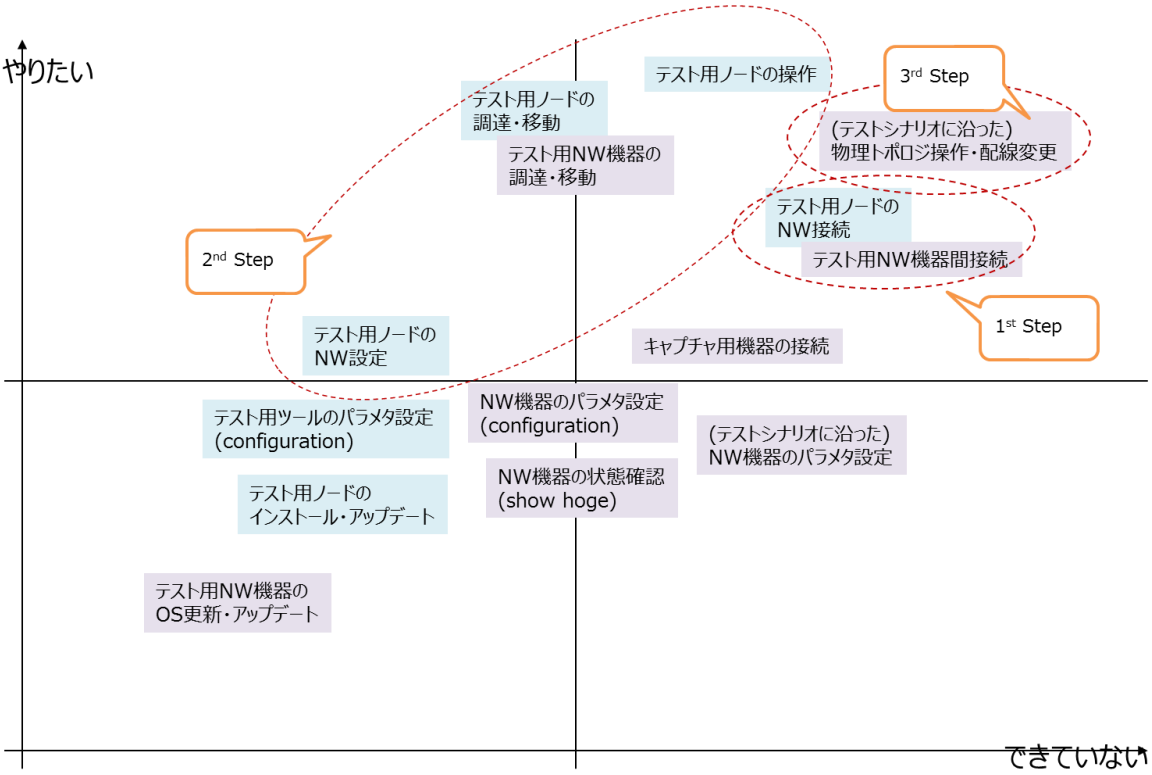


図 3-2 テストオペレーション視点の PoC ターゲット選択

3.3 PoC の目的設定

3.3.1 PoC ターゲット設定

3.1 節～3.3 節の検討をもとに、PoC のターゲット(対象・シチュエーション等)を表 3-1 のように設定した。

表 3-1 PoC ターゲット設定

	PoC ターゲット
いつ	● 本番導入前の NW(機器単体・結合)テスト
誰が	● 本番化対象の機器または複数機器の組合せについて、導入前動作検証を行うエンジニア
どこで	● 検証環境・ラボ環境
何を	● テスト対象: 検証環境で構成する機器単体あるいは小規模環境のテスト用 NW (複数拠点・WAN 接続を含む構成は今回の PoC スコープ外。)
どうする	● テスト対象の機能的な試験(通信機能)の試験実施 ● テストアプリケーションとしては、データプレーンのテスト(NW 利用者の実際の通信)を対象とする。(マネジメントプレーン・コントロールプレーンの操作の結果、データプレーンで実際に狙った通信ができているかどうかのテスト)
期待すること	● 試験環境の物理構成変更作業の自動化: ➢ テスト対象に対するテスト用トラフィックのエントリポイント増減、付け替え、テスト対象のトポロジ構成・トポロジ変化などテストシナリオを、自動操作可能にすることで作業の省力化を図る[3]。 ● 正常性の確認の自動化: 複数のトラフィックを生成して、テスト対象のデータプレーンを通し、テスト対象が想定した動作(通信)をしていることの確認を行う。 ➢ 作業者がマネジメントプレーン・コントロールプレーンの設定(どうあるべきかの定義)に集中できるようにする。最終的に実現させたい通信のチェックを、自動的に・たくさん・早く実行する。 ➢ 簡単なテスト用アプリケーションの作成、人的作業の集中ができればテスト用アプリケーションのつくり方・使い方・考え方も変わるのではないか。

3.3.2 PoC 目的設定

3.3.2.1 実証したいことの概要

今回の PoC におけるターゲット設定に対して、L1patch によって「ネットワークのテスト」の以下の課題がどのように変えられるかを検証する。PoC では特に、現状人力作業のものや、単純だがパターンが多く、リソース(人・物・時間)の制約により、一部のテスト実行をあきらめざるを得ないような作業をターゲットにする。

- テストすべきこと・テストとして実施したいことは、すべてテストできるようにする
  - テストパターン網羅問題への対処
  - リソース制約によるテストの縮退実行

[3] ネットワーク構成変更を含むテストシナリオ実行をゴールと設定しているのでこうした機能についても考慮を入れるが、この観点ではすでに OOL のテストベッド PJ での取り組みがあるため、本プロジェクトではここの点についての優先度は高くない。

3.3.2.2 PoC ターゲットに対する課題感

ネットワークのテストを検討する際、以下に挙げる理由から、検討したテストパターンがそのまますべて実行可能であることはほとんどない。

- テストパターン自体が非常に多い(図 3-3)
  - 小規模なネットワーク(数台の NW 機器・数個のネットワークセグメント)だけでもテスト対象となるトラフィックパターンは多数になる。実際にはその上にアプリケーション(高レイヤのネットワークテスト)が乗ってくるためさらにパターンが増える。
  - 複数人で並行して実行可能なテストもある。複数人のテストでは、テスト基準や結果の判断基準がテスト実行者によって異なってしまい、最終的なテストの品質保証が難しくなってしまうことを避けるために、判断基準などを厳密に設定しておくことが求められる。また、複数の人が分散して行ったテスト結果(個々のテスト作業用ノードのログなど分散して取得されるエビデンス)を集約し、全体としての成否を判断する作業が別途発生する。
- リソース制約によるテストの縮退
  - ネットワーク構成パターンに沿った通信テストでは、いくつかの場所や環境(ネットワークパラメータ)などに応じてテスト用の機材を用意しておくことが望ましいが、準備可能なテスト用ノードなどの機材リソースの都合で理想的なテスト構成を取れないことがある。
  - 特に物理構成パターンの網羅については、「現地・現物」の操作が必要になることが多く、規模が大きい・複数拠点に渡るようなシステムでは時間的・人的なリソース制約から想定されるパターンをそのまますべて実行することができないことが多い。(人が移動して・人手で操作して・人の目で見て行うオペレーションは、作業範囲に含められるテスト対象のサイズに限界がある。)

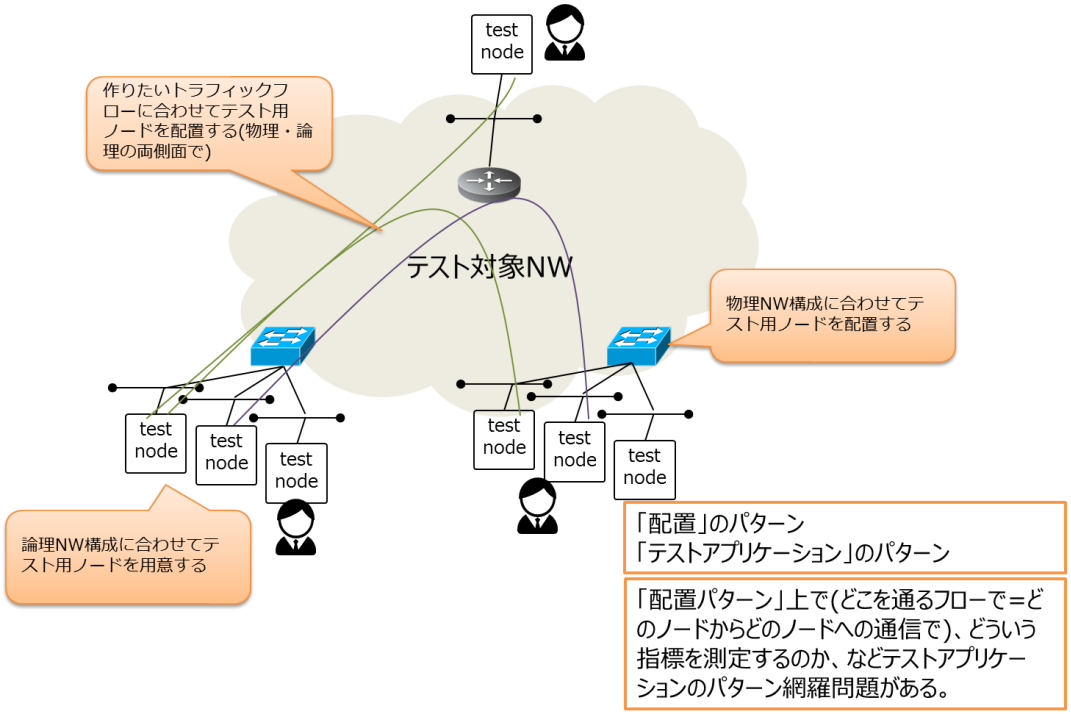


図 3-3 テストシナリオ検討の際の「パターン」

これらの点が重なって、機械的に想定されるテストパターンのうち特に重要な通信要件や、特定の構成パター

ンをグループ化してその中の代表例などを選択し、一部のテストパターンのみを行うことが実際には多い。パターンを定義するだけであればごく単純な作業だが、そこから利用可能な人的・時間的・機材的なリソースの範囲内でのテストケースを選択し、実現性のあるスケジューリングを行う、という非常に複雑な作業を行っている。

そのため、テストケース選択やテスト成否判定の基準ずれ(案件やテスト設計者など人や状況に応じて異なる基準でテストが行われる)、特定のテストケースの選択漏れ、再テスト実行などが往々にして発生し、トラブルの火種となる。また、網羅できていないテストパターンや条件の中から障害が発生することがある。

3.3.2.3 実証するために用意する機能

3.3.2.2 項に示した課題感を踏まえて、PoC ではこれらの点が実現可能かどうかを検討・実証する。

- 物理構成に対するパターン網羅のための仕組み
  - テスト対象ネットワークの環境(物理トポロジ)セットアップの自動化
    - ✧ NW 機器間の接続
    - ✧ 作業用ノードの準備と NW への接続、作業用ノードのパラメタ設定・セットアップ
  - 作業用ノードの操作(作業用ノードを使ったテスト操作)とテスト結果取得
    - ✧ シンプルなノードが、テストパターンに応じてたくさん用意できること。それらの操作(テスト用アプリケーションの実行とデータの取得)が自動化できること。
      - 用意できるノード・作業者に制限されない
      - 自動的に NW 機器間を接続する
      - 多数のテスト用ノードをテスト対象 NW に接続する
      - 複数のテスト用ノードでの操作を自動実行し、結果を集約する
- ポリシベースのテストパターン自動生成と実行
  - 物理構成やテスト用ノード配置などの制約を解決できるシステムとして、簡単なテストパターン(テストポリシ)からパターン網羅するように個々のテストケースを生成し、自動実行・結果取得・レポーティングなどの一連の操作を実行する。

PoC では、上記にあげた点を実現するための機能を試作し、パターン爆発問題・リソース制約によるテスト縮退問題をどの程度解決できるかを検証・検討する。テスト対象ネットワークの物理トポロジ(ネットワーク機器間の接続)についての操作が可能になることから、物理リンク障害などを想定したテストも実施するが、テストとしては静的なテストを想定する。[4]

4 PoC 実施内容

4.1 L1patch 仕様策定

4.1.1 L1patch 機能概要

3 章で定義したターゲットに対して、図 4-1 および下記のように L1patch の機能を定義した。

- 2 種類のワイヤの定義
  - 専有モードワイヤ(exclusive mode wire)

[4] 本書では、定常状態にあるネットワークに対するテストを「静的なテスト」、状態変化(状態遷移)やそのときの影響などを含むテストを「動的なテスト」としている。たとえば、ping を連続して実行しておいて障害を発生させ、障害発生時の経路切り替わり・その際の通信(ping)への影響時間などを調べるといったテストは、本 PoC では想定していない。

- ✧ 接続したいポート間を物理ポート単位で設定する。専有モードワイヤを構成するための中間経路では物理ポートを専有する。1 物理ポートがひとつのワイヤに対応する。
- ✧ PoC ではテスト対象 NW 機器間の接続(物理トポロジ)を構成するために使用している。

➢ 共有モードワイヤ(shared mode wire)

- ✧ 接続したいポート間を MAC アドレス単位で設定する。共有モードワイヤを構成するための中間経路では物理ポートは共有される。1 物理ポートが複数のワイヤに対応する。
- ✧ PoC ではテスト対象 NW 機器とテスト用ノード(Mininet host)間の接続(テスト用ノード配置)を構成するために使用している。

- VLAN 対応
  - ➢ 共有モードではひとつの物理ポートに複数のワイヤ(複数のテスト用ノード)を接続する。そのため、接続先のテスト対象機器(DUT)ポート側で VLAN を使用するオプションが必要になる。
  - ➢ PoC 実装では、テスト用ノード(Mininet host)は VLAN を使用しない(Mininet host が接続する OVS とは”vlan access” port でつながっている)とした。DUT 側ポートは”vlan trunk” port を設定できるものとし、vlan tag の付け外しは L1patch(OFS, OpenFlow ルール)設定で行う。
- Mininet サーバのプラットフォーム想定
  - ➢ 図 4-1 では Mininet サーバをハイパーバイザ上の仮想マシン(VM)として構築する想定があり、間に仮想スイッチ(vSwitch/非 OpenFlow スイッチ)が入ることを想定していた。今回の PoC では Mininet サーバはベアメタル機器で構築しているため、間に vSwitch を挟む構成は実際には検証できていない。(ワイヤ実装方式からプロミスキャスモードで動作させる必要があると想定される。)

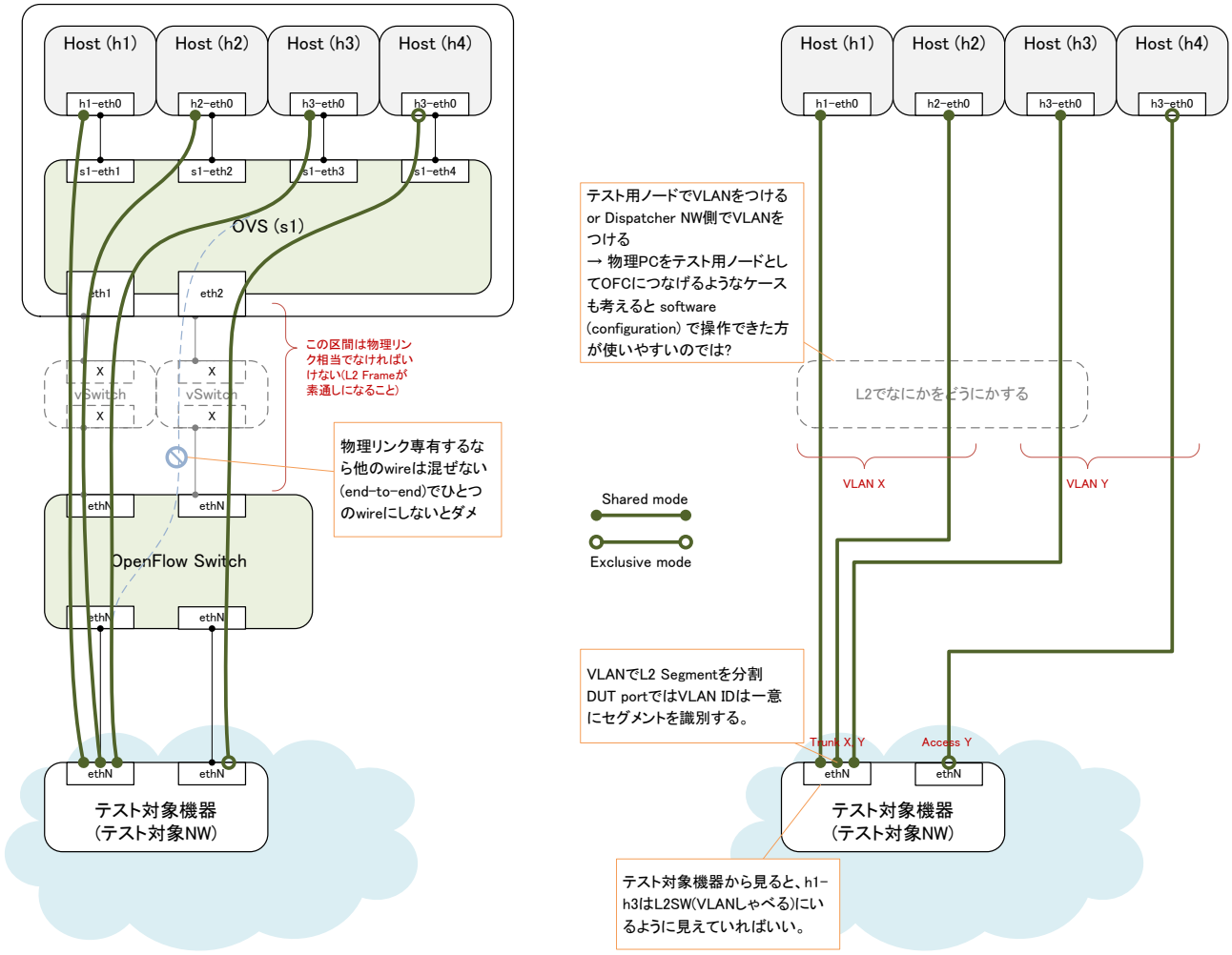


図 4-1 L1patch 機能要求定義

4.1.2 OpenFlow 制御方式

PoC では 4.1.1 項に示した 2 種類のワイヤを、OpenFlow をつかって実現している。

OpenFlow で L2 の制御を行う方式のひとつとして、ARP リクエストを packet-in させて、コントローラ(OFC)側で MAC Learning を行い、ノードの接続位置を把握する方式がある(リアクティブ方式)。今回の場合、(L1patch を経由して)テスト対象ネットワークに接続するテスト用ノードは、あらかじめ物理配置も固定(L1/L2 mobility を考えなくて良い)であり、そのネットワークパラメータ(IP/MAC アドレス)も任意に設定可能である。よって、ARP packet-in によるリアクティブ動作ではなく、MAC アドレスを既知としたプロアクティブ方式で OpenFlow による L1patch 制御ルールを生成している。

フロールール設計や実装上の詳細については「技術レポート」を参照すること。

4.2 PoC で実行した内容

最終的にテストの自動化を実現するために、3 段階の実行ステップ(マイルストーン)を設定し、各段階で既存あるいは新しく構築するツールの実現性を確認した(表 4-1)。試験結果の詳細としては最終的に行った Step3 の内容について記載することとし、中間ステップ(Step1, Step2)についてはその概要のみ記載する。

表 4-1 PoC 実行時マイルストーン設定

Step	概要
Step1	L1patch によるテスト環境の構成 ● 検証環境のネットワーク機器間接続(物理トポロジ)を、L1patch(専有モードワイヤ)で構築する。(従来手作業で行っていたテスト対象ネットワークの物理トポロジ構築を L1patch で実現する) ● 狙い: OFC の API(REST API)ベース操作の基礎機能実装と動作確認 ➢ REST API で OFC(OFS に対して設定するフロールール)を設定する。 ➢ 機器間接続と OFS に対して設定するフロールールを分離する。本来テストシナリオ作成者が定義すべきテストの情報(what)と、それをどのように実現するか(how)を分離する。 ● 期待: テスト実施時のテスト対象ネットワーク構築に関する物理作業の解決 ➢ テストシナリオやテストパターンに合わせて、毎回テスト対象機器間の接続を作り直す/テスト用のノードの再配置を行うための手作業の排除: 物理構成設定の自動化・時間短縮、それによる検証環境で構成できる物理構成パターンの自由度の拡大。
Step2	Mininet 応用[5]によるテスト用ノードの準備と操作 ● テスト用ノードの準備(生成)・配置・接続・ノードのネットワークパラメタ設定・ノード内での作業(テスト用ノード操作)方法の確立 ● 狙い: 仮想化技術を利用したテスト用ノードの生成と配置、テスト作業を行うための基本機能の確認 ➢ Mininet を利用したテスト用ノードの自動生成、ノードのネットワークパラメタ設定(IP 等)の自動化 ➢ 自動生成されたテスト用ノードを L1patch(共有モードワイヤ)で任意の箇所に配置し、テストが可能であること。テスト用ノード操作(コマンド実行・テスト結果取得)の動作確認。 ● 期待: リソースネックによるテストシナリオやテストパターン縮退といった、「本来行いたいテスト」を阻害する要因の排除。 ➢ テスト用のノード準備(機材の数や環境内への配置など物理的リソース問題・セットアップや操作するための人手などの時間的・人的リソースの問題)への対処として Mininet を利用する。 ➢ 多数のテスト用ノードにテストシナリオに応じたネットワークパラメタを設定し、適切な箇所(物理的な接続位置)に配置すること、また多数のテスト用ノードを操作し、テストアプリケーションを実行して結果を取得することにかかる人的・時間的リソースの解決。
Step3	テスト用ノード操作の自動化/(簡易)オーケストレーション ● Step1/2 で行った、テスト対象ネットワークの物理トポロジ操作とテストの自動実行(多数のテスト用ノードの生成・配置・操作の自動化)の組み合わせによるシナリオテストの実行。

4.3 PoC における前提・制約の設定

4.1 節・4.2 節でも一部挙げているが、今回の PoC においては以下のような前提や制約を設定している。

[5] Mininet が利用可能であることがわかったため省略しているが、実際には Linux namespace を応用したアプリケーションなども検討している。Mininet を使うことにした理由については 9.1 節を参照すること。

- テスト用のノードや物理構成の情報はあらかじめ与えられる(given)とする。
  - テスト対象ネットワークの構造(トポロジ)
  - テスト用のノードをどのように配置するか
- テスト対象の NW 機器の設定(CLI 操作の自動化)は含めない。
  - CLI 操作の自動化(マネジメントプレーンの操作)は既存のツールなどがあるので PoC の主要なターゲットにしない。(PoC 上、テスト対象ネットワークの設定は手動で実施。)
- PoC におけるテスト環境の前提・制約
  - 一度構築し、状態が安定したネットワークに対して、そのうえで同じテストを繰り返す状況、すなわち「静的なテスト対象ネットワークで、テストアプリケーションを複数パターン自動実行する」ことを PoC のターゲットにする。
  - シングルユーザ・シングルテスト:「検証環境をひとつのテストで占有して実行する状況」を最初のターゲットにする。

4.4 PoC 評価項目設定

PoC の評価として、ルータ・スイッチから構成されたテスト対象ネットワークに対する L3 通信テスト(ICMP)を行い、発生する作業にかかる人・時間・物理リソースをもとに評価を行うこととする。テスト方針やシナリオの検討から、テスト環境の構築、テストの実行、レポーティングという、テストを行う際に発生する一連の作業について、下記の 3 パターンの実装レベルでどの程度変化があるのかを確認する。

- すべて手動(従来のテスト)
- Step2 相当: L1patch+Mininet を利用して環境構築を行う、通信テストの実行自体は手作業で行うが、作業はすべて Mininet 上で集中実行する。
- Step3 相当: 環境構築、通信テスト実行のすべてを自動化する。

4.5 テスト対象ネットワーク

L1patch を使ったネットワーク構成には 2 種類の物理構成がある。それぞれについて今回の PoC の構成図を示す。PoC 使用した機材の詳細は 9.1 節に記載する。

- テスト対象機器と L1patch 構成要素との接続を含む物理構成(図 4-2)
  - 個別のテストによって変化しない、テスト環境としての物理構成
    - ✧ テスト対象の NW 機器はすべて OFS に対してのみ接続され、NW 機器間は直接接続されない。
    - ✧ テスト対象機器および OFS はすべて独立した管理アクセスのためのポートを持つ(out-of-band な管理ネットワーク接続を持つ)。図 4-2 中の"TestBed NW"がこの管理ネットワークにあたる。
- L1patch の設定を行うことによって構成される、テスト対象機器間の物理構成(図 4-3)
  - 個別のテストごとに定義される NW 機器間接続
    - ✧ L1patch に設定を行うことによって、テスト対象機器間を図 4-3 のようにする。
    - ✧ その上でテスト用ノードの配置・ノード間のトラフィックパターン(テストパターン)などの検討を行う。テスト方針・テストパターンについては 5.2 節参照。
- L1patch
  - OpenFlow スイッチによって実現される。

- OpenFlow1.0 を使用する。
  - L1patch 実装上は OpenFlow1.3 にも対応するようにしたが、PoC は OpenFlow1.0 を使用して実行している。(OpenFlow1.3 を使用した検証は行っていない。)
  - 今回実装した L1patch に求められる OFS 要件については 9.1.2 項参照。

L1patch の設定をおこなうことで構成する(テスト対象 NW 機器間の接続やテスト用ノード配置を行って実現する)テスト対象ネットワークを図 4-3 に示す。テスト対象 NW は以下の考え方のもとで構成されている。

- VLAN/VRF によって構成されるマルチテナントネットワークを想定している。
  - 今回の PoC では 1 テナントのみのテストを行っている。
- ひとつのテナントは 3 つのゾーンを持つ。
  - 外部(ext)/内部(int)/DMZ(dmz)
    - IP や VLAN ID の払い出しルールなども設定しているが詳細はここでは記載しない。
  - 検討上は内部/DMZ-外部間の接続で NAT を行う想定があったが、今回の PoC では 1 テナントのテスト単純化し、NAT やゾーン間接続制御(Firewall)設定は設定していない。(すべてのゾーン間・セグメント間で通信が可能な状態としている。)
  - 内部ゾーンは 2 つ、DMZ はひとつ、外部ゾーンはひとつのネットワークセグメントを持つ。(内部・DMZ については複数のセグメントを持つことができ、テナントごとに必要な数を選択するという設定。)
    - テスト用ノードは 5.2 節に示すテスト方針に必要なノード数を定義する。
    - 各セグメントにおいて、ルータは VRRP により first hop を冗長化する。

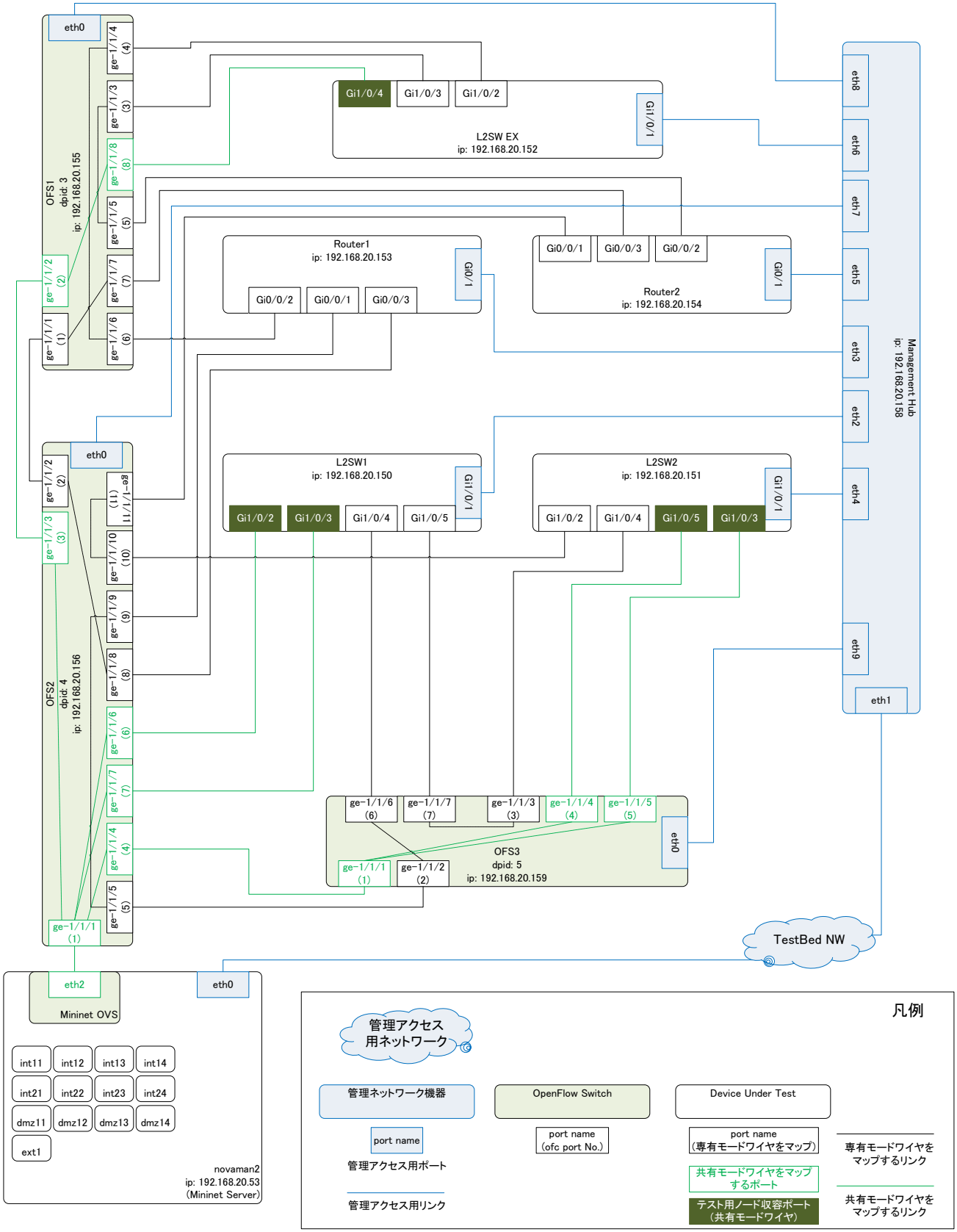


図 4-2 PoC 環境物理ネットワーク図

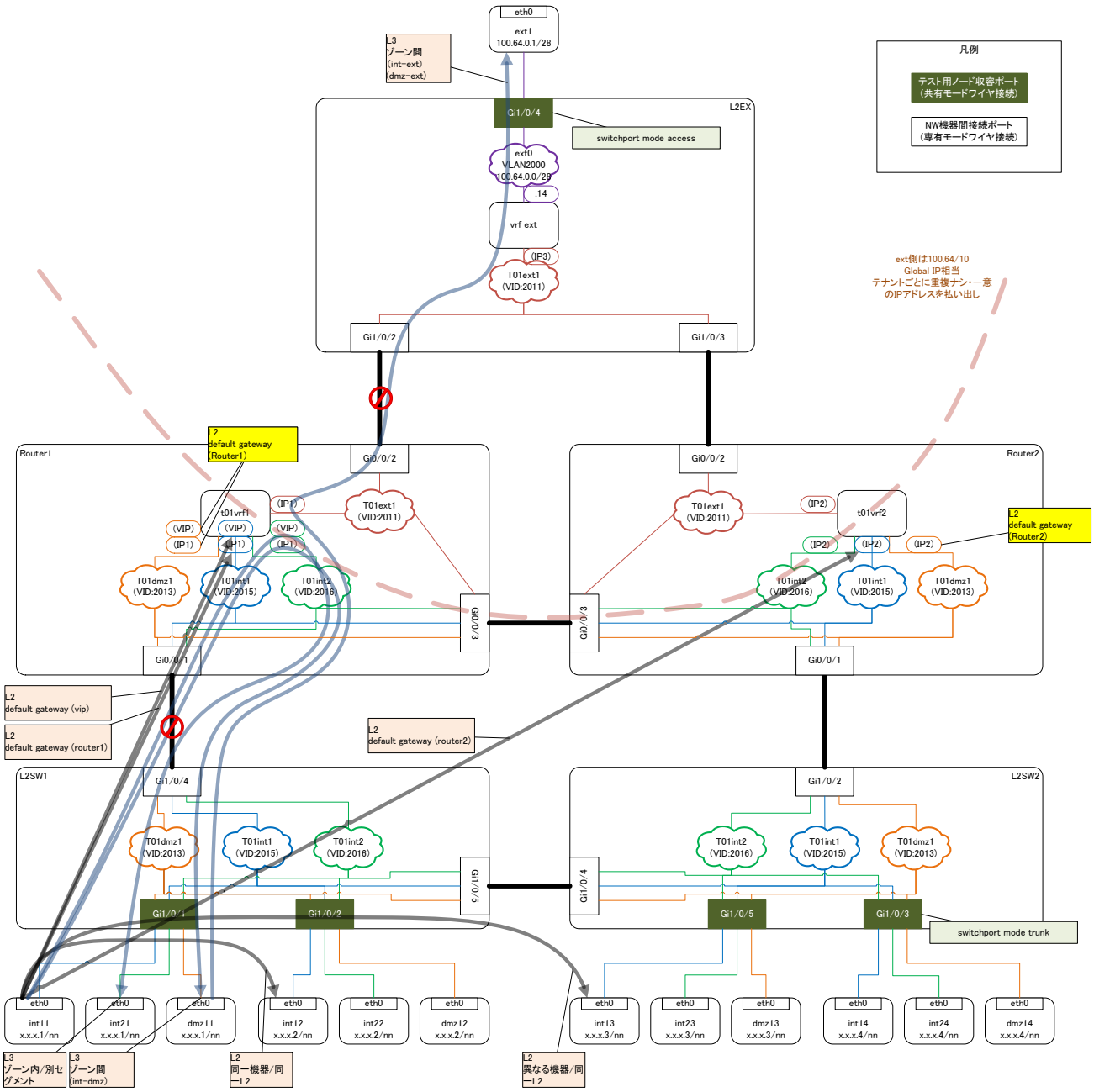


図 4-3 L1patch によって構成するテスト対象ネットワークとテストトラフィック

5 PoC 結果

5.1 テスト内容および結果の概要

テストを行う際にどういった作業が行われているか、L1patch の利用によってそれらの作業がどのように変化するのかの概要を表 5-1 に示す。また、各項目における作業項目および実行テストケース数を表 5-2 に示す。各タスクにおける作業の詳細、L1patch 利用による変化、結果の評価については以降の各節で考察する。

なお、表 5-1・表 5-2 の「従来(すべて手作業)」は今回実際には実施していない。OOL テストベッドでの検証構築環境の都合、一部に OFPatch(OOL テストベッドの物理環境構築システム)が入ることと、実機設置場所(IT 診療パーク)とオフィス(兼箇段 DC)で場所が離れていて、物理作業そのものに拠点間移動が必要になるためである。

表 5-1 シナリオテストで行う作業概要

<table>
<tr><th colspan="2">タスク</th><th>従来(すべて手作業)</th><th>L1patch+手作業テスト</th><th>L1patch+テスト自動実行</th></tr>
<tr><td rowspan="2">[A]
テスト計画・設計</td><td>[A1] テスト対象 NW の要件定義・設計、テスト方針の決定</td><td colspan="3">変化なし。</td></tr>
<tr><td>[A2] テスト方針に基づいたテストシナリオの作成。</td><td colspan="2">方針に基づいたテストパターンの検討、利用可能な物理リソースや実行時間を考慮した冗長なテストパターンの削除。</td><td>テストパターンの定義(定義ファイルの作成)</td></tr>
<tr><td rowspan="2">[B]
テスト
スクリプト
実装</td><td>[B1] テストパターンからテストシナリオを生成するためのスクリプト実装・テスト</td><td>(不要)</td><td>(不要)</td><td>テストパターン定義から個々のテストシナリオを生成するためのスクリプト実装(正しくパターン生成されているかどうかのテスト)</td></tr>
<tr><td>[B2] テストシナリオを自動実行するための素栗と実装・テスト</td><td>(不要)</td><td>(不要)</td><td>テストシナリオを読んでテスト用ノードを操作し、テストを自動実行するスクリプトの実装と動作テスト</td></tr>
<tr><td rowspan="3">[C]
テスト対象
NW 構築</td><td>[C1] テスト対象 NW を構築するための NW 機器の調達</td><td colspan="3">変化なし。</td></tr>
<tr><td rowspan="2">[C2] NW 機器間接続(物理トポロジの構築)</td><td rowspan="2">今回実施なし。
構築したい NW 機器の構成図をもとに手作業で直接 NW 機器間に配線。</td><td>NW 機器から必要な本数の配線を OFS に配線。OFS 間も機器間接続を実現できるだけのリンクを配線。この時点では構成したい NW の物理構成を参照する必要はない。後から NW 機器間を接続するために必要な配線(本数)が確保できれば良い。</td><td></td></tr>
<tr><td>テスト実行に必要なテスト対象 NW 機器間の接続を定義(L1patch 専有モードワイヤの定義作成)、定義からフロールールを生成し L1patch OFC へ設定。</td><td></td></tr>
</table>

	[C3] 物理構成の確認(CDP など)	今回実施なし。 目視あるいは CDP や LLDP などの物理接続確認用プロトコルを利用した物理接続確認	CDP/LLDP による確認を行うという点では変化がない。 ただし、直接テスト対象 NW 機器間が接続されているわけではないので、いくつかの制約が発生する点に注意が必要となる。	
	[C4] NW 機器への設定投入(configuration)	変化なし。		
	[C5] テスト用ノードの調達	今回実施なし。 テストパターン・テストシナリオに沿って必要な台数を準備	Mininet 動作可能な機材(Mininet サーバ)1 台を準備	
[D] テスト実行	[D1] シナリオに沿って NW 機器の設定変更、状態確認	テスト対象 NW のテスト前状態(事前状態)確認(ルーティングテーブルや VRRP Active/Standby 状態など)。 変化なし。		
	[D2] テスト用ノードを NW 機器へ接続	今回実施なし。 調達した台数をテスト対象 NW にて作業で配線	テスト実行に必要なノードの定義とテスト対象への接続位置を定義(L1patch 共有モードワイヤの定義作成)、定義からフロールールを生成し L1pach OFC へ設定。	
	[D3] テスト用ノードの NW パラメタ設定	今回実施なし。 個々のノードを手作業で設定	テスト用ノード定義の中に含まれる。	
	[D4] シナリオに沿って複数のテスト用ノードを操作、結果の取得	個々のノードを手作業で操作(コマンド実行・結果取得)		スクリプト中で自動操作・結果取得
		今回実施なし。 物理的に異なるノードを操作(可能であればテスト用ノードにリモートアクセスして操作端末を切り替え)	Mininet 上ですべての作業を実行(ホスト/namespace 指定で捜査対象のノードを切り替え)	
	[D6] 複数台のノードで取得したテスト結果の集約、加工、テスト結果	今回実施なし。 個々のテスト用ノードで取得したデータをリモートコピー(あるいは USB 等の	Mininet サーバ上に保存した結果を集約。	スクリプト中で結果の自動集約・レポート生成。

	評価	物理メディアでコピー)して集約。		
[E] 障害試験模擬(テスト対象NWでのトポロジ変更作業)	[E1] リンクダウンの発生	今回実施なし。 直接ダウンさせたいリンクを操作(手作業による物理的なリンクダウン操作実行)	OFS 操作によりダウンさせたいリンクを構成させているポートを停止。	
	[E2] 再テスト実行	[D]を再実行		

表 5-2 テストパターン数・作業時間的な変化のサマリ

タスク		従来(すべて手作業)	L1patch+手作業テスト	L1patch+テスト自動実行
[A] テスト計画・設計・方針	[A1] テスト対象 NW の要件定義・設計	変化なし。		
	[A2] テスト方針に基づいたテストシナリオの作成。	項目の検討と書き起こし 2 時間程度/90 ケース		テストパターン定義してテストケースを自動生成。 1 時間弱/194 ケース
[B] テストスクリプト実装	[B1] テストパターンからテストシナリオを生成するためのスクリプト実装・テスト	(不要)	(不要)	4 時間前後 (0.5 人日程度)
	[B2] テストシナリオを自動実行するための素栗と実装・テスト	(不要)	(不要)	40-45 時間前後 (5 人日程度)
[C] テスト対象 NW 構築	[C1] テスト対象 NW を構築するための NW 機器の調達	変化なし。		

	[C2] NW 機器間接続(物理トポロジの構築)	今回実施なし。(物理配線に 2 時間前後と想定。)	物理配線に 2 時間前後	
			L1patch の設定作成に 1.5 時間	
	[C3] 物理構成の確認(CDP など)	確認時間は同等		
	[C4] NW 機器への設定投入(configuration)	変化なし。		
	[C5] テスト用ノードの調達	今回実施なし。(4 台のノード調達と仮定。営業日換算で半日前後?)	物理機器としては Mininet サーバ 1 台のみ。OOL テストベッドの借用時間＋OS 更新・Mininet 等セットアップ含めて 1-2 時間程度。	
[D] テスト実行	[D1] シナリオに沿って NW 機器の設定変更、状態確認	変化なし。		
	[D2] テスト用ノードを NW 機器へ接続	今回実施なし(テスト用ノード 4 台に対して 0.5～1 時間程度と想定。)	L1patch 設定作成に 0.5 時間(テスト用ノード 13 台分の定義作成、フロールール生成・OFC 設定)	
	[D3] テスト用ノードの NW パラメタ設定	今回実施なし(テスト用ノード 4 台に対して 0.5～1 時間程度と想定。)		
	[D4] シナリオに沿って複数のテスト用ノードを操作、結果の取得	今回実施なし(テストノード 4 台・90 ケース実行、テスト手順書への結果の記入で 1～2 時間前後かかると想定。)	Mininet 上でテスト実行、テスト手順書への結果の記入。30 分程度/90 ケース(テストに問題がない場合)	Mininet 上で自動実行。194 ケース/3～4 分(テストに問題がない場合)
	[D6] 複数台のノードで取得したテスト結果の集約、加工、テスト結果評価			

[E] 障害試験模擬(テスト対象NWでのトポロジ変更作業)	[E1] リンクダウンの発生	今回実施なし。 (現地作業が発生。移動時間により変動)	リモートでの OFS 設定変更(数分)
	[E2] 再テスト実行	[D]再実行	

5.2 [A] テスト方針

テストとして何を行うべきか、というテスト要件の定義・方針策定においては特に変化する点はない。今回のテストでは以下のテスト方針を設定している。

- すべての物理構成(すべてのデバイス)において通信が正しく行われることを確認する。
- すべての論理構成において通信が正しく行われることを確認する。

そのために表 5-3 のパターンでテストを行うことにした。特に「同一機器・同一 L2」パターンは、機材の数に応じてテスト項目が増えること、かつ、障害が起きる可能性が低いところなので省略されることが多い。しかし、特定機器でのトラブル切り分けで必要になるテストパターンである。今回は、試験として確認しなければいけないパターンをすべて網羅するという目的があるので、ネットワーク的に隣接(同一機器/同一 L2)から順に接続性の確認を行う方針としている。(4.5 節参照)

表 5-3 テスト方針設定・テストパターン検討

● テナント内通信試験 　➢ L2/隣接ノード間通信試験 　　✧ 同一機器/同一 L2 　　✧ 異なる機器/同一 L2 　　✧ Default gateway 　➢ L3 　　✧ ゾーン内通信試験 　　✧ ゾーン間通信試験 　　● 内部-内部(int-int, int-dmz) 　　● 内部-外部(int-dmz, dmz-ext, int-ext)
● 冗長系(障害試験) 　➢ VRRP Active 側通信経路の障害に対指定通信が切り替わること 　　✧ Router1-L2SW1 間リンクダウン 　　✧ Router1 ノードダウン

5.3 [B] テストスクリプトの実装

テストの自動化を行う際には、テスト用ノード上でのテストアプリケーション実行とその結果の取得など、テストシナリオ実行手順の自動化実装が必要になる。今回の PoC では手動テスト・自動テストの両方を実行できるようにしている。表 5-2 のように実装自体にはかなり時間がかかっているが、これは Mininet 操作のための基本機能の実装と動作確認などを含めて行っているためである。

5.3.1 手動テスト

手動テスト実行にあたっては、5.2 節に定義した方針に基づいて個々のテストケース(90 ケース)を設定し、実行している。テスト項目一覧を 9.1 節/表 9-4 に示す。これは従来通り、特定の端末から特定の宛先に対しての通信を行うテストを手動で実行するものである。(操作するテスト用ノードが Mininet host になる。)

5.3.2 自動テスト

5.2 節に定義した方針に基づいて、テストパターンを定義する(表 5-4)。定義ファイルには、通信を行うべきホストの集合を複数定義する。(4.5 節・図 4-2・図 4-3 参照)

- ここでは単純に、通信を行うべきテスト用ノードの集合と通信可否の想定(SUCCESS/FAIL)のみを定義する。表 5-4 ではテスト方針(5.2 節参照)に基づいて 16 個のパターンを定義している。定義されたテストパターンにから、個々のテストシナリオを生成する。
  - 表 5-5 のようにパターンごとに送信元・送信先集合を選択し、集合内の要素(テスト用ノード)の組み合わせを計算する。
  - 16 パターンをもとに 194 個のテストケースが生成される。
- テストシナリオを読み込んで、Mininet 上に生成したテスト用ノードで ping コマンドを実行する。

表 5-4 テストパターン定義(抜粋)

scenario_test_topo5.json	

{
(省略)
"scenarios": {
"L2-return-by-device": {
"L2SW1-dmz1(SUCCESS)": [
[
"dmz11"
],
[
"dmz12"
]
],
(省略)
},
"L2-via-neighbor-device": {
"dmz1(SUCCESS)": [
[
"dmz11",
"dmz12"
],
[
"dmz13",
"dmz14"
],
[
	同一機器・同一 L2
	● 同一機器・同一セグメント間のノード間相互通信を定義している
	
	異なる機器・同一 L2
	● 同一セグメント・異なる機器にあるノードについて、相互にノード間相互通信を定義している。

"@router1-dmz1-ip@",
"@router2-dmz1-ip@",
"@dmz1-vip@"
]
],
(省略)
},
"L3-within-zone": {
"int(SUCCESS)": [
[
"int11",
"int12",
"int13",
"int14"
],
[
"int21",
"int22",
"int23",
"int24"
]
]
},
(省略)
}
}
	ゾーン間(セグメント間)
	● セグメント語に、ノード間相互通信を定義している。

表 5-5 テストケースの生成

scenario_generator.py	

def generate_task(self, task, task_data):
task_list = []
for set1, set2 in itertools.permutations(task_data, 2):
for prod1, prod2 in itertools.product(set1, set2):
prod1_is_param = self._is_param(prod1)
if not prod1_is_param:
task_list.append(self._generate_ping_task(task, prod1, prod2))
return task_list
	送信元・送信先集合の選択
	
	組み合わせの生成

5.4 [C] テスト対象ネットワークの構築

5.4.1 L1patch による物理トポロジの構築

物理的な配線作業量そのものについてはそれほど大きな変化はないが、物理配線作業時に求められる考え方は下記のように変化する。

- L1patch を使う場合、テスト対象の NW 機器間のリンク 1 本を構成するために、最低でも[NW 機器-OFS-NW 機器]という構成が必要になるため[6]、本数としては L1patch を使用した場合の方が全体の配線量は増大する。
- ただし、L1patch を使用した場合、NW 機器～OFS 間の配線については特定のテストの要件に依存しない。つまり、実行したいテストに関係なく、テスト対象機器～OFS 間の物理配線(ポート数)が足りているかどうかが問題となる。そのため、テスト対象機器の初期設置の際にある程度物理配線を OFS に対して行っておくことで、その後の任意のテストが実行可能になる。(配線数が足りている限り、テストのたびに物理配線を都度変更する必要がなくなる。)

5.4.2 L1patch の設定による NW 機器間の接続

L1patch で実現する NW 機器間接続情報を定義し、それをもとに OFS のフロールールを生成・設定する。

- L1patch を使用する場合、テスト対象ネットワークの構成図をもとにした作業は、物理機器・物理配線の時にではなく、L1patch 設定の作成を行うときに必要になる。
- フロールールを設定することで初めて NW 機器間(ポート間)は接続された状態になる。そのため、直接 NW 機器間を物理結線する場合と比較して、以下のような制約が発生する。
  - L1patch(専有モードワイヤ)のフロールール設定を OFC に行うまでは NW 機器間の接続は実現されない。テスト対象 NW の物理トポロジを構築した後で、機器間接続を確認する必要がある。
  - テスト対象機器間が直接結線されているわけではないため、目視による物理構成確認はできない。CDP などのプロトコルベースの物理トポロジ確認を行う必要がある。

5.5 [D] テスト実行

5.5.1 物理構成自動化の効果

L1patch を使うことで、テスト用ノードの設定(ネットワークパラメタ設定)とテスト環境への配置で物理作業を行う必要がない状態になっている。

- テスト用ノードは Mininet 上のホストとして生成しているため、作業用 PC(物理テストノード)などを使用する従来のケースと比べて、利用可能なノード数が増大している。
  - Mininet 上のホストは Linux namespace を利用して構成されている。OS/Mininet の許容可能な範囲内で任意の数のホストを生成できる。
  - 多数のノード(テスト用ノード)があったとしても、人が操作可能なノードの数には上限がある。また、テスト対象に直接結線されている場合はその位置まで移動しなければならない。リモートアクセスでテスト用ノードを操作可能なときも、作業に応じてアクセス先(ウィンドウ)の切り替えを行う必要がある。
    - ✧ Mininet 上のテスト操作は、「操作主体となる namespace(ホスト名)+コマンド」の形で行うことができる

[6]今回の専有モードワイヤの機能としては。

ため、複数のホストで行う操作、操作主体となるホストの切り替えなどを柔軟に行うことができる。(ただし同時並行作業などは別途考慮が必要になる。)

- 利用可能なテストノード数の上限が上がることから、テスト対象 NW の規模(物理機器数)それ自体の増大・環境拡張に対してもテスト用ノードのリソースがネックになり得る可能性はかなり低くなっている。
  - テストパターン・テストシナリオ検討時に、利用可能なノード、担当者数、操作可能な範囲など、人的・物理的な制約からテストケースを縮退させなければいけないという追加考慮・リスク検討はこうした仕組みによりかなり軽減できる(当初想定したテストパターンをそのまま実行できるようになっている)と考えられる。

5.5.2 手動作業の検討

今回、テストの実行(テストアプリケーションの実行:今回は ping 疎通試験)は手動による実行と自動実行の 2 種類を試せるように PoC 実装を行った。自動実行にあたっては、以下のようなケースで ad-hoc な手動テストを行う状況が発生した。

- L1patch によるテスト対象 NW のトポロジ構成・テスト用ノード配置を行った後、テスト用ノードのパラメタ設定や接続先の状態確認を行う際に、個別の確認作業を手動で行う必要がある。
  - 特に、何らかの形で通信できない事象が発生した場合: 何がどこまでつながっているのか、Mininet サーバ/OFS/テスト対象機器間で原因を切り替えなければならない。ひとつの事例として、L1patch ワイヤとそれを接続するテスト対象機器側ポートの設定で VLAN Tag の取り扱い設定に関する不整合があったことがある。Mininet ホストからのパケットの送出と、テスト対象機器/OFS/Mininet サーバそれぞれでのパケットダンプを行い、どこからどこまでの通信が行えているのかを切り分けることになった[7]。
- テスト自動実行スクリプトの実装にあたって、「テスト自動実行スクリプト」それ自体の動作テストが必要になる。

これらの理由から、今回はテスト対象 NW のトポロジ構成、テスト用ノードの生成と配置のみをおこなって Mininet CLI に入る機能を用意した。また、テスト自動実行スクリプトでも、自動実行中に強制中断して Mininet CLI に入る機能を実装している。テスト中に不審な動作をしている場合に中断した上で、テスト対象 NW の状態確認や特定テストの手動再実行を行い、各種デバッグ作業を行えるようにした。

5.5.3 自動実行処理の検討

テストアプリケーション実行の自動実行にあたっては以下の点が考慮ポイントとなる。

- テストアプリケーションの設定定義
  - 今回は ping を使っているが、パケット送信間隔、タイムアウト間隔、パケット送信時間(何秒間パケット送信を試みるか)などのオプションがある。
  - オプション設定によってテストの結果やテスト実行時間に変動がある。たとえば、環境構成直後や一部動作の不安定な機材がある場合は、最初の数パケットには応答がないが一定時間待つ・一定数パケットを打つと通信が復帰するような状況がある。

[7] NW 機器間を直接結線する通常のテスト対象 NW 構築を行っている場合、NW 機器間の配線を疑うだけでよいが、L1patch ではそれに相当する部分を(複数の)OFS を経由して実現しているため、中間 OFS やそれらの持つフロールールなどに問題がないかどうかをチェックしていかなければならない。つまり L1patch が間に入る分、障害ポイントは増大している。L1patch を使うことによって構成の柔軟さを得られるが、物理配線のトラブルやポートの物理呼称などによるトラブル長期化のリスクは高まる。

- 実行結果の判断
  - 今回の場合は ping を行い、送信した ICMP パケットの半数以上に応答があった場合に「成功」と判断している。
  - 「成功」をどのように考えるかを定義する必要がある。MAC learning の時間があるため初期の数パケットについて応答がないことは一般的なネットワークでも発生する。今回 PoC 実行中に発生した事例では、手作業テストで ICMP パケットを送信し始めてから数秒(5-10 秒)間パケット応答がない状態が続き、その後は通常通り応答が返る事象が見られた[8]。こうした事象をどのように判断するかを定義する必要がある(手作業テストでは、作業者によってはこうした状況を「異常なし」と判断する可能性がある)。
- テストリトライ
  - テストを実行した際、失敗した(想定と実際の結果が異なった)ときに一定時間待って、再度同じテストを繰り返し実行するようにした。表 5-2 でテスト自動実行を 3〜4 分としたが、これはすべてのテストが問題なく実行され、リトライが発生しなかったケースの実行時間である。
    - 今回はすべてのノード間で通信が成立するようになっているため、通信が成功する場合はこの時間ですべてのテストが終了する。ゾーン間 Firewall などを設定し、通信できないことが正しいテストケースがある場合、タイムアウト待ちなどが発生してテスト実行時間が長くなることが想定される。
- フロー設定順序
  - L1patch 動作を実現するための OFS 設定(フロールール設定)は、Mininet 上の OVS を含めて検討する必要がある。Mininet を起動し、OVS が動作可能になった時点でフロールール設定をする必要があるため、今回実装したスクリプトでは、Mininet の起動・テスト用ノードの生成と設定を行った後でフロールールの設定と OFC への投入を行うように実装されている。
  - 障害試験などで、全体の NW 構成を維持したまま一部の構成だけ変えたい、といった状況があるため、どの作業で何のフロールールを設定するかは、ルール設定が必要となる。(後述: 6.2.1 項参照)

5.6 [E] 障害試験模擬

5.6.1 リンクダウン障害模擬

リンクダウンによる障害発生は L1patch OFS のポートシャットダウン(ダウンさせたいポートの対向側 OFS ポートのリンクダウン)によって実行した。リンクダウン状況を模擬する方法として表 5-6 の方式が考えられるが、今回は物理障害もぎなのでリンクダウンを使っている。(フロールールを操作する論理的な障害模擬は今回の PoC では試せていない。)

表 5-6 L1patch によるリンクダウン障害模擬方法

方法	DUT 側リンクダウン検出有無	状況・制約
OFS ポートダウン(物理的な障害模擬)	DUT 側でリンクダウン検出可能	● リンク両端でのダウン(直結されている物理リンクの停止)、リンク片側でのダウン(中間に L1/L2 機器がある場合のリンクダウン)などが実現可能 ● 今回の専有モードワイヤのように、DUT 間接続が物理ポート

[8] ある OFS の再起動により復旧したため OFS の不調と考えられる。詳細原因不明(未調査)。

		単位で設定されている必要がある。
フロールール操作(論理的な障害模擬)	リンクダウン検出不能	● フロールール操作により特定方向のパケットドロップを発生させる方法なので、フィルタリングに近い。リンクダウンは発生しない。 ● 特定方向のみの停止(Tx/Rx いずれかのみの停止)を模擬できる(ファイバ片芯の障害)。 ● 物理ポート単位ではなくワイヤ単位の操作になるので、専有モードワイヤの通信に対しても設定できる。

5.6.2 テストの繰り返し実行

L1patch を使ったテストでは、NW 機器間の接続の実現(物理トポロジの構成)を行った後にテスト用ノードを配置し、実際のテストアプリケーション実行(ping)を行う。

何らかのイベントを発生させ(リンクダウン等)、再度同様のテストを再実行することはネットワークのテストにおいてよく行われる。当初、テスト自動実行スクリプトでは、テスト対象 NW の物理トポロジ構成～テスト用ノードの配置までをすべて一括で行い、テスト実行終了後にそれらのフロールールをすべてクリアする実装にしていた。この場合、1 回のテスト実行が終わるたびにすべてのフロールールがクリアされ、テスト対象 NW 機器間の接続(物理トポロジ)もすべて切断される状態になっていた。

ネットワークに対するイベント発生とテスト再実行を行うためには、テスト対象 NW の物理トポロジを維持したまま、その中の一部のフロールール修正やイベント発生ができなければならない。テスト自動実行スクリプトの実装としては、物理トポロジ操作(専有モードワイヤ)の設定とテスト用ノード配置(共有モードワイヤ)のルールを個別に操作できるように修正した。(詳細について 6.2.1 項で後述)

6 考察

6.1 PoC 実行結果の評価

6.1.1 リソース制約によるテスト縮退への対処

3.3.2 項で挙げたように、今回の PoC の目的のひとつにリソース制約によるテスト縮退に対処することがある。

- リソース制約について
  - テスト用ノードの生成・設定・操作に Mininet を利用することで、想定されるテストを行うために必要なテスト用のノードが不足しているという問題は解決できるようになった。ノードの生成と操作は Mininet の機能で、生成したノードの配置については L1patch の機能を利用することで、テスト用ノードの確保と物理配置のスケーラビリティを獲得することができている。
  - ただし、今回のようなシングルタスクのテストだけでなく、複数のトラフィックを同時に実行しなければいけないテスト(複数のテストアプリケーションを同時に実行したいテスト)では Mininet 上でのタスク実行がネックになる可能性がある。
    - ✧ Mininet 機能上、複数並列でタスクを実行できるかどうか。テスト用ノード操作と結果のハンドリングをどのように行えるか。(未検証/今後の検討点のひとつ)
    - ✧ ひとりの人が作業可能な範囲の制限: たとえば Step2 相当/Mininet 上で特定ホストのターミナルを

複数開き(特定ホストのネームスペースで複数のターミナルを実行し)マルチタスク作業を行うことは可能である。これは通常のテスト用ノードをリモートホストで使っている状況とほぼ同等になる。作業用ノード数問題が解決できても、人が操作可能な範囲(操作可能なノード数)には限界がある。

- 作業工数について
  - 表 5-2 から「[B]テストスクリプト実装」にかなり時間がかかっていることがわかる。
  - 今回の PoC ターゲットは検証環境と設定している(表 3-1)。以下の点を考慮して、今回作成したようなテスト自動化スクリプトがどの程度の頻度で実行するかどうかがコスト的な評価ポイントとなる。テスト自動実行のための基本機能・一般的なテストを抽出し、再利用性の高いテスト用ツールを用意することで自動テストそのものの実装コストは回収できる。
    - 通常行われる基本的なテストの「共通ツールボックス」化: ICMP による通信試験(今回の PoC ターゲット)、TCP/UDP や Firewall を含めた通信機能(疎通)テストなど、よく利用される通信テスト機能をプールし、再利用することで、テスト実行にかかるコストを削減することができる。
    - テストの再利用性の向上: L1patch 上でテスト対象 NW を構成する場合、テスト対象 NW のトポロジ、DUT の設定(コンフィグ)、L1patch の設定など環境を構成するためのすべてのデータがファイルの形で保存可能になる。自社サービスなどで特定の環境を定期的に構築してテストするようなケースではテストの再利用性が向上する。
    - 属人性の排除: 自動化を行うことで、あるテストを実行下の判断結果や作業の行い方など、個人のスキルや経験などが影響する要素が減る。
    - テストの漸進的な機能拡張: テスト自動化そのものはコストのかかる作業だが、確実に作業が記録され改善可能になるというメリットがある。
      - たとえば、NW 機器のバグ再現試験など、何度も同じようなテストを実行しなければいかに場合には、ログの取り方などを都度実装していくことでテストの制度そのものを向上させてゆくことができる。
      - テスト環境(テスト対象 NW 構成)などの環境情報を含めて記述可能になる。それらがバージョンコントロールされたうえで改良・改善できるようになる。
- 複雑なテストアプリケーションの実行、性能テストの実行について
  - L1patch の特性上、通信そのものが行えるかどうか(データプレーンの機能テスト)は可能であるが、通信性能のテストには向かない。テスト用ノード～テスト対象 NW の通信の間に L1patch(OFS)が入るために純粋にテスト対象 NW の性能だけを抽出することが難しいためである。(今回は機能テストに特化するものとし、性能面の考慮はスコープ外としている。)
  - Mininet host 上で実行可能なアプリケーションはベースとなる OS 上にあるアプリケーションとなる。OS 上のリソースを大量に消費するテスト用ツールや、そもそも非 Linux のテストツール等がある場合には「テスト用ノード」に相当するものを別途 L1patch に接続し、ワイヤを設定する必要がある。それらの独立したツールの操作やオーケストレーションなどは別途検討しなければならない。
    - たとえば、(IXIA 等の)専用測定器でネットワーク帯域測定を行う場合、L1patch 上十分な帯域を確保するように専有モードワイヤのパスを選択して IXIA をテスト対象機器へ接続する、といったテストは実行可能である。その場合遅延時間測定には L1patch を経由する分の遅延時間が加算される。事前に L1patch 部分だけの性能や特性を測定しておくことで、ある程度、テスト対象 NW の性能を

推定することは可能と考えられる。

6.1.2 パターン爆発問題への対処

3.3.2 項であげた PoC のもうひとつの目的は、パターン爆発問題への対処である。

- テスト用リソースの配置パターンと操作について
  - 6.1.1 項で述べた。Mininet と L1patch の組み合わせによって、テスト対象の構成(物理構成・論理構成)パターンに対して、必要なテスト用ノードの配置を解決できる。
- テストケースの自動生成・テストパターン網羅について
  - 今回は 5.2 節に挙げた、テストのあるべき論として設定されるテストパターンの定義とパターンからのテストケース自動生成を行った(5.3.2 項参照)。
  - テスト用リソース配置の制限がなくなることで、単純なパターンにしたがって個々のテストをそのまま定義・実行することが可能になる。
    - ネットワークテスト、特に通信試験では、特定グループ間・特定経路を経由するトラフィックを生成するというのが主要なタスクになるため、ごく単純なパターン定義だけでほとんどのテストケースが生成できる。実際、表 5-5 に示したようにテストケース生成では単純な集合演算しかしていない。
    - 手動実行する場合(9.3 節参照)、同等と見なせるテストケースを集約して、いくつかの代表例のみを実行することになる。今回のような ping(ICMP)による通信テストでは、応答がある(パケットが往復できている)のであれば 2 点間の片側からだけでも問題ないと判断できるが、ステートフルファイアウォールなどが入る場合には、送信元・送信先を含めてパターンを考慮する必要がある。テスト作業量などを見込んで必要なテストケースを検討するには、テスト優先度を判断可能なネットワークの知識が必要になる。(仕様や設計をもとに必要な確認項目を考えられる(単価の高い)エンジニアを使って、単純な作業をしなければいけない。)
    - 本 PoC のように、個々のテストケースの抽出を考えず、テストとして本来必要なパターン定義は何か、という点に注力させることでテストの網羅性を高め、本来必要なテストそのものの定義や結果の分析へ人的リソースを配分することが可能になる。
      - リソース問題を含めて自動化することで、より短時間で多数のテストケース実行を行うことはできるようになったとはいえ、テスト対象ネットワークの環境規模などで、組み合わせ爆発が起こるという点は変化がない。生成されるテストケース量と実行時間が許容範囲を超える場合に、自動生成されるテストケースの最適化(往復ではなく片方向だけテストケースを生成するなどのルール設定)が必要になる。
      - パターン爆発に対するスケーラビリティや最適なテストケース選定などは今後の課題としては残るだろう。ただし、本 PoC のような仕組みを使うことで、そうした対処が、プログラムによる形式的なアプローチでおこなえるようになるという点ではメリットがある。

6.1.3 テストの記述と人的作業の排除

プロジェクトの目的としては、ネットワークのテストを記述可能なものにすること、ネットワークの構築・操作・テストのサイクルから人的作業を排除すること、がある。

- テストの記述

- 6.1.1 項・6.1.2 項ですでに取り上げているが、L1patch を使うことによってテスト対象ネットワークの構成情報の定義、テストアプリケーション(テストパターンとテストシナリオ)の定義を行えるようになっている。
  - 環境情報の定義: ネットワーク機器の多くはテキストの設定ファイル(コンフィグレーションファイル)を持つ。これらとテスト対象機器間の接続情報(L1patch の物理構成設定)とを合わせて持つことで、設定ファイルのリストアだけでテスト環境を交換・再現可能になる。
  - テストアプリケーションとテスト用ノード配置の定義: テストアプリケーションは、テストパターン・テストシナリオ及びテスト自動実行スクリプトによって決定される。今回はいずれも json データおよび python スクリプトである。
- これらが、容易にバージョン管理できる形(テキストデータ)で「定義・記述」される。L1patch の設定ファイルの生成やバリデーションなどはテキストデータを扱うシンプルなツールで実装可能である。また、テストの繰り返し実行やオペレーション上の修正なども、設定ファイル・スクリプトの修正などによって行えるようになるため、バージョンコントロールされたもとで(ソフトウェア開発で一般的なマナーで)でコントロールできるようになる。

- 人的作業の排除
  - 人がテスト作業を行う場合は必ずミスが発生する。特定のチェック箇所や記録漏れ、見落とし、操作ミスなどを考えなければいけない。また、同じ・単純な作業の繰り返しそれ自体が手作業のテストでは負担になる。
  - 自動テスト実装を行って、テストの繰り返し自体が負担ではなくなることに意味がある。
    - 小さな修正や変更点に対しても、今回の場合すべての通信パターンが数分で実行できる。検証環境が、特定の本番環境を模擬したもので、本番環境での作業を検査するものだとすれば、より短い時間でたくさんの試行をチェックできるようになる。
    - 自動テスト実装には確かに実装コストがかかるが、単純作業の繰り返しで人的リソースを消耗させる必要がなくなるという点では大きなメリットがある。ネットワークエンジニアはテスト実行そのものではなくテストパターンの定義やテスト結果の分析に注力し、テスト実行作業そのものはスキルの低いエンジニアへ委託することも可能になる。

6.2 L1patch を応用したテスト実行の検討点

6.2.1 L1patch のテスト実行フェーズの考え方

5.6.2 項であげたように、テスト対象 NW の操作(テスト対象 NW でのイベント発生)とテストの再実行という操作を考えた際に、NW トポロジと繰り返し実行されるテストで必要な処理を都分けて考える必要がある。今回の PoC では単純に物理構成操作(専有モードワイヤ操作)とテストノード配置(共有モードワイヤ操作)で分けて操作するようにした。実際には以下の観点でワイヤ操作を分けて考える必要がある(図 6-1)。

- 操作の分類
  - すべてのテストで共通となる環境それ自体に対する操作
    - 複数のイベント発生とテスト実行をおこなう一連の流れ(シナリオ)において共通する・長時間維持すべきネットワーク構造の構築
    - ベースとなるネットワーク構造に対する差分変更操作
  - 個々のテストで必要なリンクやテスト用リソースの配置に対する操作

- ✧ テスト用リソースの配置: テストごとに必要なリソース(ノード数やテストアプリケーション)は変化する。異なるテストを同一の環境で切り替えながら実行する際に、テスト前後関係でのリソース干渉がないようにする必要がある。

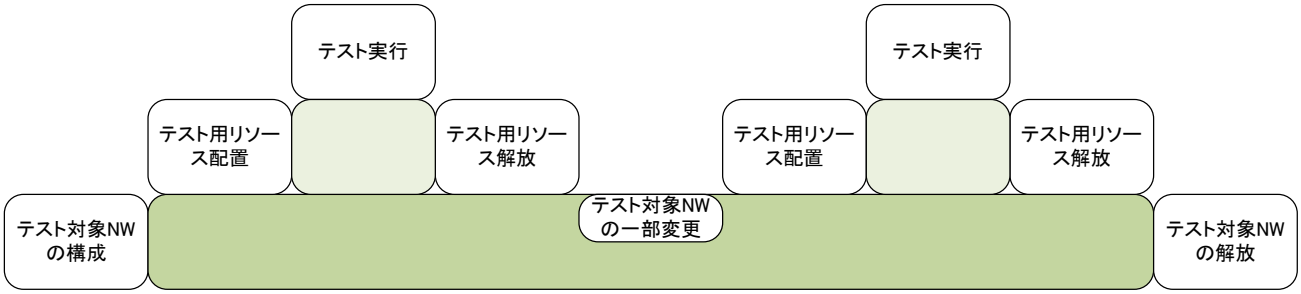


図 6-1 L1patch 応用テスト実行フェーズ

今回の PoC では Mininet によるテスト用リソース制約への対処を考えているため、Mininet だけを使ってテストを実装している。しかし、実際の利用状況を考えると、テストのために専用アプライアンスを特定の機材に接続したいなど、個々のテスト実行のために物理構成(専有モードワイヤ)操作が必要になるケースが考えられる。したがって、ワイヤ種別だけで単純に考えるのは本来正しくない。

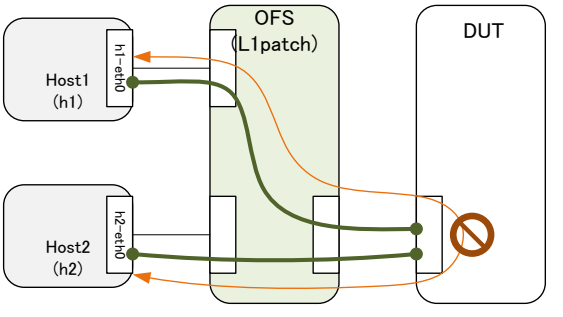
6.2.2 L1patch それ自体の配置方法の検討

6.2.2.1 同一デバイス内折り返し

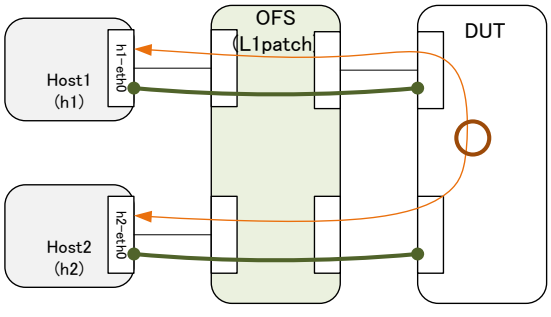
今回実装した L1patch(共有モードワイヤ)の通信制約に、同一機器での折り返し通信がある(図 6-2)。共有モードワイヤを使うと、図 6-2(a)のように DUT(L2 機器)のひとつのポートに対して複数のホスト(テスト用ノード)が接続されているように見せかけることが可能となる。これは通常の物理結線では想定されない接続パターンとなるため注意が必要である。

図 6-2(a)では、DUT はワイヤが紐付けられている物理ポートでホスト(h1,h2)の MAC アドレスを学習するが、同一のポートに接続されているために相互のフラッディングが行われない。また、L2 スイッチでは通常、同一物理ポートでトラフィックが折り返されることを想定しないため、この状況では h1-h2 間の通信は成立しない。

5.2 節ではテスト方針として、同一 L2・同一 NW 機器での折り返し(NW 的に隣接しているノード間の通信テスト)をパターンとして含めた。これを実現するためには図 6-2(b)のように物理ポートを分けてテスト用ノードを配置する必要がある。



(a)同一ポートでの折り返し



(b)別ポートでの折り返し

図 6-2 共有モードワイヤ接続の制約

6.2.2.2 L1patch の構成

5.5.2 項で PoC 実行中の問題切り分けについて取り上げた。テスト用ノードやテスト対象機器間が直接接続されない L1patch ベースの環境では、「どこまで何が通信できているか」を確実にテストするための仕組みがより重要になる。

- ミラーリング、パケットダンプ
  - 切り分けとして以下の箇所がある
    - Mininet サーバやテスト用ノード(Mininet host)上
    - 中間経路の OFS 上
    - DUT 上
  - L1patch のコンセプトとしては、場所(物理配置)の制約に依存しないことをあげている。そのため、リモートで(現地・現物に依存しない形で)ミラーリング・パケットダンプ取得が行えなければならない。いずれの機器に対してもミラー自体の設定は可能だが、ミラーリングされたトラフィックをどのようにテスト実行者のところへ引き込むかを検討する必要がある。
- トラフィック折り返しパターン
  - 特定のノードの障害、物理リンクや物理ポートの障害、特定ポートの設定ミスや不整合などの可能性を排除するために、DUT に対して複数のパスがあり、同じデバイスに対してこれらの条件を変えて切り分け作業が可能になっていることが望ましい。
  - DUT 同一物理ポートでの共有モードワイヤ通信制約(図 6-2 参照)から、DUT 側ポートの問題切り分けのために、DUT には 2 ポート以上で L1patch(共有モードワイヤ)が接続可能であることが望ましい。同様に、中間経路の OFS や Mininet サーバのトラブルを回避・問題切り分けを行うために、テスト用ノードと DUT 間に物理的に異なる 2 経路が確保されるのが望ましい。(図 6-3)

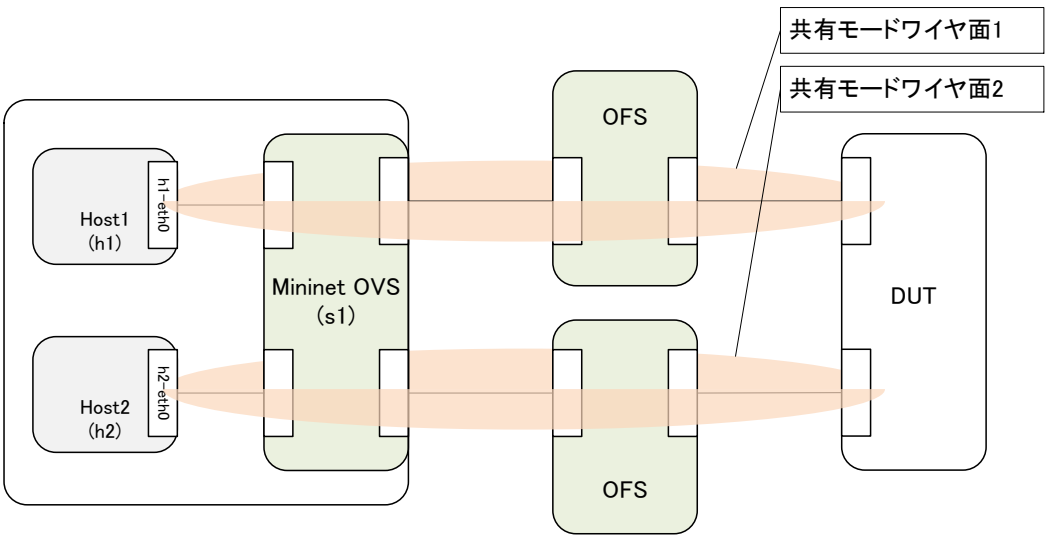


図 6-3 冗長性を考慮した L1patch(共有モードワイヤ)構成案

今回の PoC で構築した環境(図 4-2)では Mininet サーバ～OFS2 が単一障害点となっている。実際に OFS2

の動作不調による障害が発生しているが(5.5.3 節参照)、現在の構成ではそれらの障害対応(問題切り分け)のために現地移動しなければいけない状況となっている。図 6-3 の用に、テストノード配置を物理 2 面で行えるとこうした状況でも問題箇所の切り分けや、ワークアラウンド(L1patch 上の設定変更による共有モードワイヤ経路の片寄せ)による試験継続などが可能になる。

7 おわりに

沖縄オープンラボラトリで実施した、「L1patch 応用ネットワークテスト」の概要、目的、実証実験の実施内容と結果、それらの評価について記載した。

本書ではまず、プロジェクトの概要と目的を説明し(2 章)、それにそって今回の概念実証(PoC)のターゲットおよび目標の設定をどのように行ったのかを示した(3 章)。PoC のターゲットおよび目標に対して、”L1patch”と既存のツール(Mininet)を組み合わせたシステムの構成を検討し(4 章)、実際の試験環境(テスト対象のネットワーク)に対するテストを行った。実環境での検証では、人手によるテストと自動テストでのテストパターン数や作業時間などを計測し(5 章)、PoC の目的として設定した、リソースによるテスト縮退・テストのパターン爆発という問題への対処が可能になることを実証した(6 章)。

8 謝辞

本プロジェクトの実行にあたって、ネットワークテストのユースケースヒアリングや実証実験ターゲットの検討など、特に計画フェーズで沖縄オープンラボラトリ賛助会員各社から議論やヒアリングに参加いただきました。改めて本プロジェクトに参加してくださった皆様に感謝します。

- 協力いただいた OOL 賛助会員各社(順不同・敬称略)
  - 株式会社アドックインターナショナル
  - イクシアコミュニケーションズ株式会社
  - エヌ・ティ・ティ・コミュニケーションズ株式会社
  - 株式会社オキット
  - 日本電気株式会社
  - ブロケード コミュニケーションズ システムズ株式会社
- プロジェクトオーナー
  - 新日鉄住金ソリューションズ株式会社

9 補足

9.1 PoC 環境で使用した機器・ソフトウェア情報

9.1.1 ハードウェア・ソフトウェア

今回、本プロジェクト実行・検証にあたっては沖縄オープンラボラトリのテストベッドを使用した。テスト対象機器(Cisco スイッチ・ルータ)については新日鉄住金ソリューションズの機材を持ち込んで使用した。サーバおよびOpenFlow スイッチは沖縄オープンラボの機材を借用している。(スイッチについては一部 NTT コミュニケーションズからも借用している。)

今回の PoC 環境で使用したハードウェアの一覧を表 9-1、ソフトウェアの一覧を表 9-2 に示す。

表 9-1 PoC 環境ハードウェア一覧

機器分類	メーカー・型番	
Mininet/OFC サーバ (novaman2)	NEC Express5800/R120d-1E	● CPU: Intel Xeon E5-2470 @ 2.30GHz (16thread) ● Memory: 16GB ● HDD: 854GB
テスト対象機器 ルータ (Router1, Router2)	Cisco Cisco1921/K9	● IOS 15.2(4)M7
テスト対象機器 スイッチ (SW_EX, L2SW1, L2SW2)	Cisco Catalyst 3750G-24TS-1U	● IOS 12.2(50)SE5 (ipservices)
OpenFlow Switch (OFS1)	Quanta T1048-LB9	● PicOS 2.5.2 / Revision 19975 ● Type:1GE/Fuature:BaseProduct,Layer3,OpenFlow
OpenFlow Switch (OFS2)	Pica8 P-3290	● PicOS 2.2.1S3 / Revision 14775 ● Type:1GE/Fuature:BaseProduct,Layer3,OpenFlow
OpenFlow Switch (OFS3)	Quanta T1048-LB9	● PicOS 2.5.2 / Revision 19975 ● Type:1GE/Fuature:BaseProduct,Layer3,OpenFlow

表 9-2 PoC 環境(Mininet/OFC サーバ)ソフトウェア一覧

ソフトウェア	バージョン
OS	Ubuntu 14.04.3 LTS
Python	2.7.6
Pip	1.5.4
Mininet	2.2.22
Ryu	3.24
Open vSwitch	2.0.2

9.1.2 OpenFlow スイッチ要件/OpenFlow バージョン指定

PoC で実装した L1patch フロールール・OpenFlow コントローラに求められる要件は、OpenFlow1.0/1.3 に対応していて、表 9-3 に示す機能を持つことである。なお、L1patch を構成する OpenFlow スイッチは、すべてが単一の OpenFlow バージョンを使うことを前提としている(OpenFlow1.0/1.3 の混在環境は想定していない)。

表 9-3 OpenFlow スイッチ要件(OpenFlow1.0/1.3)

分類	項目
Match	In-Port
	VLAN ID
	Source/Destination MAC Address
Action	Out-Port
	Push/Pop VLAN
Property	Priority

9.2 Mininet の採用理由

テスト用ノードの生成・操作を行うために今回は Mininet を利用した。物理的なリソースネックを排除するという観点では何らかの仮想化技術であればよく、他の選択肢も考え得るが、今回特に Mininet を利用することを考えた理由として以下の点が上げられる。

- 必要なテスト用ノードの機能として複雑なテストアプリケーション実行を想定していない
  - 静的な環境におけるネットワークテストでは、何らかのテスト用コマンドの実行・ファイル操作ができれば良い、程度の要求になる。VM を起動するほどのリソースは必要なく、多数の(テストパターンに応じた)NW 設定(IP etc)のもとコマンド実行できればよい。
  - 今回の L1patch であれば、そうしたテスト用ノードが OFS(Open vSwitch)に接続されていれば良い。
- 複数のインスタンスを操作できる。
  - 上記の条件であれば Linux container 系も選択肢として考えられる。Mininet の場合さらに、ノード操作を行うための API が用意されており、複数のノードに対する操作やコマンド実行結果の取得などを行うテスト実行のための機能がそろっているという利点がある。
- Mininet 上で利用するために作成された既存の資産が再利用可能になる
  - Mininet は SDN コントローラ(OFC あるいはその上で動作するアプリケーション)のテストのために利用される一般的なツールである(図 9-1)。そこでは Mininet 上に複数の OFS(OVS Bridge)を生成して実施のネットワークトポロジを模擬し、実際のテスト(ノード間通信テスト)を行う。
  - 今回作成した L1patch を使うと、仮想環境上に構築したネットワークではなく、実際の物理トポロジに対してテスト用のノードを配置できる。テスト対象ネットワークとネットワーク機器へのノード接続の方法が異なっているだけで、仮想環境上で作成したテストシナリオ(テスト用のスクリプト等)が実ネットワーク(物理ネットワーク)にも流用可能になる(図 9-2)。
    - たとえば図 9-1 のように(従来の Mininet の使い方で)ネットワークコントローラをテストし、そのコントローラで実機(実際の物理ネットワーク)を構成するような際に、同じテストスクリプトを流用することが可能になる。

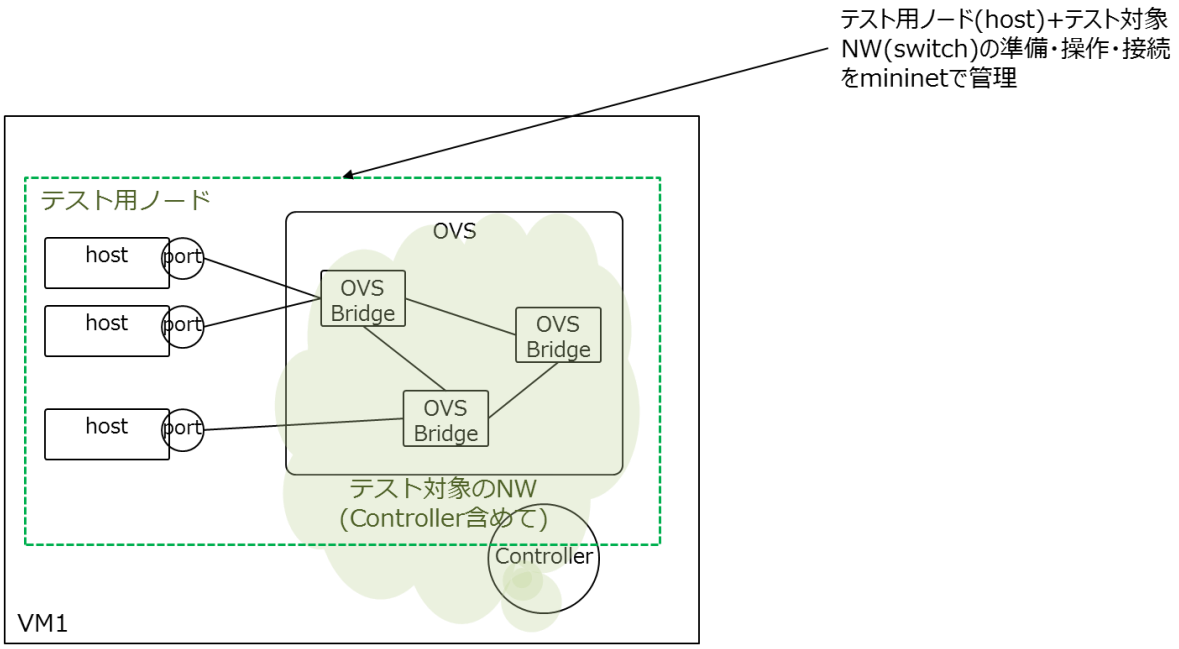


図 9-1 Mininet による従来のネットワーク機能テスト

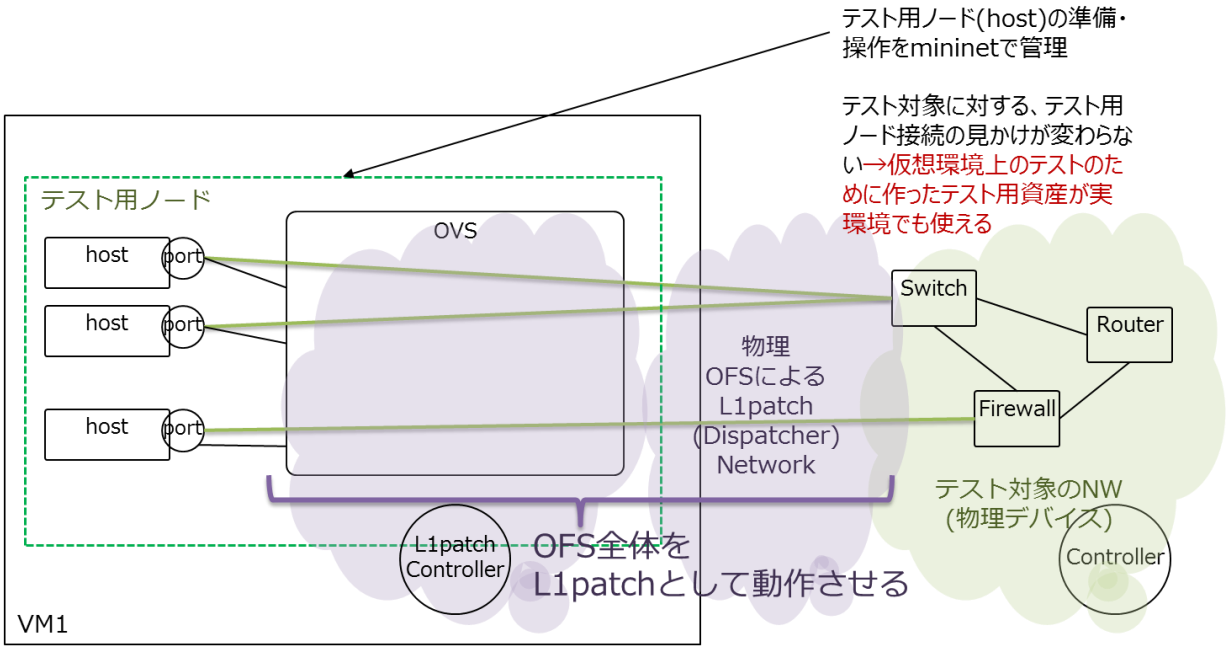


図 9-2 Mininet と L1patch の組み合わせによる物理ネットワークの機能テスト

9.3 手動テスト試験項目

表 9-4 手動テストの試験項目

大項目	中項目	小項目	試験手順
事前確認	テナント内通信試験	dmz1 セグメント (private/アドレス持ち込みあり)	1. 作業用端末(試験環境 MGMT NW)より SSH 接続 2. 下記コマンドを実行 3. 実行結果が OK であることを確認する
		internal 2 セグメント (private/アドレス持ち込みあり)	
		external 1 セグメント (global 相当/払い出し/unique)	
	疎通確認	Mininet host dmz11 ⇔ Mininet host dmz12 の疎通確認	dmz11 ping -c 5 dmz12
		Mininet host int11 ⇔ Mininet host int12 の疎通確認	int11 ping -c 5 int12
		Mininet host int21 ⇔ Mininet host int22 の疎通確認	int21 ping -c 5 int22
		Mininet host dmz13 ⇔ Mininet host dmz14 の疎通確認	dmz13 ping -c 5 dmz14
		Mininet host int23 ⇔ Mininet host int24 の疎通確認	int23 ping -c 5 int24
		Mininet host int13 ⇔ Mininet host int14 の疎通確認	int13 ping -c 5 int14
		Mininet host int11 ⇔ vip(T01int1)の疎通確認	int11 ping -c 5 10.15.1.254
		Mininet host int11 ⇔ IP1(T01int1)の疎通確認	int11 ping -c 5 10.15.1.253
		Mininet host int11 ⇔ IP2(T01int1)の疎通確認	int11 ping -c 5 10.15.1.252
		Mininet host int11 ⇔ IP3(vrf ext)の疎通確認	int11 ping -c 5 10.11.1.252
		Mininet host int11 ⇔ default gateway の疎通確認	
		Mininet host int11 ⇔ Mininet host int13 の疎通確認	int11 ping -c 5 int13
		Mininet host int11 ⇔ Mininet host int14 の疎通確認	int11 ping -c 5 int14

		Mininet host int21 ⇔ vip(T01int2)の疎通確認	int21 ping -c 5 10.16.1.254
		Mininet host int21 ⇔ IP1(T01int2)の疎通確認	int21 ping -c 5 10.16.1.253
		Mininet host int21 ⇔ IP2(T01int2)の疎通確認	int21 ping -c 5 10.16.1.252
		Mininet host int21 ⇔ IP3(vrf ext)の疎通確認	int21 ping -c 5 10.11.1.252
		Mininet host int21 ⇔ default gateway の疎通確認	
		Mininet host int21 ⇔ Mininet host int23 の疎通確認	int21 ping -c 5 int23
		Mininet host int21 ⇔ Mininet host int24 の疎通確認	int21 ping -c 5 int24
		Mininet host dmz11 ⇔ vip(T01dmz1)の疎通確認	dmz11 ping -c 5 10.13.1.254
		Mininet host dmz11 ⇔ IP1(T01dmz1)の疎通確認	dmz11 ping -c 5 10.13.1.253
		Mininet host dmz11 ⇔ IP2(T01dmz1)の疎通確認	dmz11 ping -c 5 10.13.1.252
		Mininet host dmz11 ⇔ IP3(vrf ext)の疎通確認	dmz11 ping -c 5 10.11.1.252
		Mininet host dmz11 ⇔ default gateway の疎通確認	
		Mininet host dmz11 ⇔ Mininet host dmz13 の疎通確認	dmz11 ping -c 5 dmz13
		Mininet host dmz11 ⇔ Mininet host dmz14 の疎通確認	dmz11 ping -c 5 dmz14
		Mininet host int12 ⇔ vip(T01int1)の疎通確認	int12 ping -c 5 10.15.1.254
		Mininet host int12 ⇔ IP1(T01int1)の疎通確認	int12 ping -c 5 10.15.1.253
		Mininet host int12 ⇔ IP2(T01int1)の疎通確認	int12 ping -c 5 10.15.1.252
		Mininet host int12 ⇔ IP3(vrf ext)の疎通確認	int12 ping -c 5 10.11.1.252
		Mininet host int12 ⇔ default gateway の疎通確認	
		Mininet host int12 ⇔ Mininet host int13 の疎通確認	int12 ping -c 5 int13
		Mininet host int12 ⇔ Mininet host int14 の疎通確認	int12 ping -c 5 int14

		Mininet host int22 ⇔ vip(T01int2)の疎通確認	int22 ping -c 5 10.16.1.254
		Mininet host int22 ⇔ IP1(T01int2)の疎通確認	int22 ping -c 5 10.16.1.253
		Mininet host int22 ⇔ IP2(T01int2)の疎通確認	int22 ping -c 5 10.16.1.252
		Mininet host int22 ⇔ IP3(vrf ext)の疎通確認	int22 ping -c 5 10.11.1.252
		Mininet host int22 ⇔ default gateway の疎通確認	
		Mininet host int22 ⇔ Mininet host int23 の疎通確認	int22 ping -c 5 int23
		Mininet host int22 ⇔ Mininet host int24 の疎通確認	int22 ping -c 5 int24
		Mininet host dmz12 ⇔ vip(T01dmz1)の疎通確認	dmz12 ping -c 5 10.13.1.254
		Mininet host dmz12 ⇔ IP1(T01dmz1)の疎通確認	dmz12 ping -c 5 10.13.1.253
		Mininet host dmz12 ⇔ IP2(T01dmz1)の疎通確認	dmz12 ping -c 5 10.13.1.252
		Mininet host dmz12 ⇔ IP3(vrf ext)の疎通確認	dmz12 ping -c 5 10.11.1.252
		Mininet host dmz12 ⇔ default gateway の疎通確認	
		Mininet host dmz12 ⇔ Mininet host dmz13 の疎通確認	dmz12 ping -c 5 dmz13
		Mininet host dmz12 ⇔ Mininet host dmz14 の疎通確認	dmz12 ping -c 5 dmz14
		Mininet host int13 ⇔ vip(T01int1)の疎通確認	int13 ping -c 5 10.15.1.254
		Mininet host int13 ⇔ IP1(T01int1)の疎通確認	int13 ping -c 5 10.15.1.253
		Mininet host int13 ⇔ IP2(T01int1)の疎通確認	int13 ping -c 5 10.15.1.252
		Mininet host int13 ⇔ IP3(vrf ext)の疎通確認	int13 ping -c 5 10.11.1.252
		Mininet host int13 ⇔ default gateway の疎通確認	
		Mininet host int13 ⇔ Mininet host int11 の疎通確認	int13 ping -c 5 int11
		Mininet host int13 ⇔ Mininet host int12 の疎通確認	int13 ping -c 5 int12

		Mininet host int23 ⇔ vip(T01int2)の疎通確認	int23 ping -c 5 10.16.1.254
		Mininet host int23 ⇔ IP1(T01int2)の疎通確認	int23 ping -c 5 10.16.1.253
		Mininet host int23 ⇔ IP2(T01int2)の疎通確認	int23 ping -c 5 10.16.1.252
		Mininet host int23 ⇔ IP3(vrf ext)の疎通確認	int23 ping -c 5 10.11.1.252
		Mininet host int23 ⇔ default gateway の疎通確認	
		Mininet host int23 ⇔ Mininet host int21 の疎通確認	int23 ping -c 5 int21
		Mininet host int23 ⇔ Mininet host int22 の疎通確認	int23 ping -c 5 int22
		Mininet host dmz13 ⇔ vip(T01dmz1)の疎通確認	dmz13 ping -c 5 10.13.1.254
		Mininet host dmz13 ⇔ IP1(T01dmz1)の疎通確認	dmz13 ping -c 5 10.13.1.253
		Mininet host dmz13 ⇔ IP2(T01dmz1)の疎通確認	dmz13 ping -c 5 10.13.1.252
		Mininet host dmz13 ⇔ IP3(vrf ext)の疎通確認	dmz13 ping -c 5 10.11.1.252
		Mininet host dmz13 ⇔ default gateway の疎通確認	
		Mininet host dmz13 ⇔ Mininet host dmz11 の疎通確認	dmz13 ping -c 5 dmz11
		Mininet host dmz13 ⇔ Mininet host dmz12 の疎通確認	dmz13 ping -c 5 dmz12
		Mininet host int14 ⇔ vip(T01int1)の疎通確認	int14 ping -c 5 10.15.1.254
		Mininet host int14 ⇔ IP1(T01int1)の疎通確認	int14 ping -c 5 10.15.1.253
		Mininet host int14 ⇔ IP2(T01int1)の疎通確認	int14 ping -c 5 10.15.1.252
		Mininet host int14 ⇔ IP3(vrf ext)の疎通確認	int14 ping -c 5 10.11.1.252
		Mininet host int14 ⇔ default gateway の疎通確認	
		Mininet host int14 ⇔ Mininet host int11 の疎通確認	int14 ping -c 5 int11
		Mininet host int14 ⇔ Mininet host int12 の疎通確認	int14 ping -c 5 int12

		Mininet host int24 ⇔ vip(T01int2)の疎通確認	int24 ping -c 5 10.16.1.254
		Mininet host int24 ⇔ IP1(T01int2)の疎通確認	int24 ping -c 5 10.16.1.253
		Mininet host int24 ⇔ IP2(T01int2)の疎通確認	int24 ping -c 5 10.16.1.252
		Mininet host int24 ⇔ IP3(vrf ext)の疎通確認	int24 ping -c 5 10.11.1.252
		Mininet host int24 ⇔ default gateway の疎通確認	
		Mininet host int24 ⇔ Mininet host int21 の疎通確認	int24 ping -c 5 int21
		Mininet host int24 ⇔ Mininet host int22 の疎通確認	int24 ping -c 5 int22
		Mininet host dmz14 ⇔ vip(T01dmz1)の疎通確認	dmz14 ping -c 5 10.13.1.254
		Mininet host dmz14 ⇔ IP1(T01dmz1)の疎通確認	dmz14 ping -c 5 10.13.1.253
		Mininet host dmz14 ⇔ IP2(T01dmz1)の疎通確認	dmz14 ping -c 5 10.13.1.252
		Mininet host dmz14 ⇔ IP3(vrf ext)の疎通確認	dmz14 ping -c 5 10.11.1.252
		Mininet host dmz14 ⇔ default gateway の疎通確認	
		Mininet host dmz14 ⇔ Mininet host dmz11 の疎通確認	dmz14 ping -c 5 dmz11
		Mininet host dmz14 ⇔ Mininet host dmz12 の疎通確認	dmz14 ping -c 5 dmz12
		Mininet host dmz11 ⇔ Mininet host ext1 の疎通確認	dmz11 ping -c 5 ext1
		Mininet host dmz12 ⇔ Mininet host ext1 の疎通確認	dmz12 ping -c 5 ext1
		Mininet host dmz13 ⇔ Mininet host ext1 の疎通確認	dmz13 ping -c 5 ext1
		Mininet host dmz14 ⇔ Mininet host ext1 の疎通確認	dmz14 ping -c 5 ext1
		Mininet host int11 ⇔ Mininet host ext1 の疎通確認	int11 ping -c 5 ext1
		Mininet host int12 ⇔ Mininet host ext1 の疎通確認	int12 ping -c 5 ext1
		Mininet host int13 ⇔ Mininet host ext1 の疎通確認	int13 ping -c 5 ext1

		Mininet host int14 ⇔ Mininet host ext1 の疎通確認	int14 ping -c 5 ext1
		Mininet host int21 ⇔ Mininet host ext1 の疎通確認	int21 ping -c 5 ext1
		Mininet host int22 ⇔ Mininet host ext1 の疎通確認	int22 ping -c 5 ext1
		Mininet host int23 ⇔ Mininet host ext1 の疎通確認	int23 ping -c 5 ext1
		Mininet host int24 ⇔ Mininet host ext1 の疎通確認	int24 ping -c 5 ext1
冗長系通信試験(トポロジ切り替え・切り戻しと書く状態での通信試験)	リンクダウン: L2 経路の切り替え/切り戻し	OFSW3 ge-1/1/6 のリンクダウン	
		OFSW2 ge-1/1/9 のリンクダウン	
	ノードダウン: VRRP の切り替え+L2 経路の切り替え/切り戻し	Router1 の停止(電源操作が難しいのであれば Router1 のすべてのリンクダウン)・起動(すべてのリンクアップ)	