修 士 論 文

ネットワーク構築と運用における設定情報の抽象化に関する研究

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

廣瀬 真人

2017 年 3 月

修 士 論 文

ネットワーク構築と運用における設定情報の抽象化に関する研究

指導教員　篠田陽一

審査委員主査　篠田陽一
審査委員　丹康雄
審査委員　知念賢一

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

1510046 廣瀬 真人

提出年月: 2017 年 2 月

概 要

インターネット初期におけるネットワークは、現在と比べて単純な構造であった。そのため、ネットワークオペレータがネットワークの構築または運用を行う際にネットワーク機器一つ一つに対して誤りなく設定作業を行うことが可能であった。しかし、ネットワークが大規模かつ複雑になり、ネットワークオペレータがネットワーク機器に対して誤りなく設定作業を行うことは困難になりつつある。

その要因として、ネットワーク機器の設定情報が機器単体で閉じており、他のネットワーク機器の設定情報に関与していないことが挙げられる。つまり、ネットワークオペレータがあるネットワーク機器の設定情報を変更した場合、その設定情報が他の隣接するネットワーク機器の設定情報と矛盾する場合でも許容されてしまう。したがって、ネットワーク機器の設定情報に誤りがあった場合でも、ネットワークオペレータはネットワークが稼働するまでその設定情報の誤りを認識することが出来ない。

本研究は、ネットワークが稼働する前に設定情報の検査を行うことで、ネットワークオペレータによる設定作業の誤りを予防することを目的とする。OSI 参照モデルで表現されているように、コンピュータネットワークにおいて上位レイヤのプロトコルは下位レイヤのプロトコルが正しく動作しない限り動作は保証されない。よって提案手法では、複数のネットワーク機器の設定情報をレイヤ毎にモデル化して検査を行う。これにより、設定情報が変更された箇所とそれに依存する箇所が異なるレイヤであっても検査を行うことが可能となる。また、提案手法におけるモデルは、ネットワーク機器の設定情報を、各レイヤ毎で異なるルールを適用することで構築が行われる。そのため、利用者はシステムに問い合わせることで、モデル上の論理的な接続関係について検査を行うことができる。従って、利用者が設定情報をネットワーク機器に反映する前に、本システムに問い合わせることで、その設定情報に矛盾が含まれていないか確認することが可能となる。本システムの実装には、ネットワーク機器の設定情報を抽象度の高いまま論理式として扱うことが出来る論理プログラミング言語を使用した。

本研究では、複数のトポロジにおけるネットワーク機器の設定情報を擬似的に用意し、実装したシステムにどのような問い合わせが有効であるか実験を行った。また、設定情報を実装したシステムに与えてモデル化を行うまでにどのくらい時間を要するか性能評価を行った。最後に、既存のネットワーク記述言語と提案手法との比較を行った。

ネットワークの理論的なシミュレーションは、複雑なネットワークに対して検査を行うことが可能である。提案手法における設定情報の検査は、利用者が誤った設定情報をネットワーク機器に投入してしまう事を未然に防ぐことに貢献する。

目 次

第1章　はじめに

本章では、本研究の背景、目的、本論文の構成を述べる。

1.1　背景

インターネットは人々の生活と密接に関わっている。インターネットを利用したサービスは多く存在し、コミュニケーションやエンターテイメント、情報検索など様々な形で使用されている。近年では、証券会社の株取引や医療機関の情報伝達にもインターネットが使われており、今後も適用範囲が広がることが予想される。そのためインターネットに障害が発生した際には、社会的、経済的に大きな被害をもたらす。つまり、インターネットは重要な社会基盤であり、インターネットの障害は深刻な問題である。

インターネット上で障害が起こる原因に、インターネットを構成するネットワーク機器に対して、ネットワークオペレータが誤った設定作業を行うことが挙げられる。これは、ネットワークオペレータが設定作業におけるプロジェクト毎に決められたフローを守っていても、設計書を誤認識したまま設定作業を行ってしまうことが原因である。設計書に記述された項目を見間違えたり勘違いすることによって起こるヒューマンエラーであるため、完全に防ぐことが困難である。加えて、現在ではネットワーク機器の増加と仮想化技術の進歩によってネットワークは大規模かつ複雑になっている。そのため、設定作業が煩雑になりネットワークオペレータが誤りのない設定作業を行うことはより困難であるといえる。一方で、インターネット上で障害が起こる原因は人間だけでなくネットワーク機器の性質にも挙げられる。ネットワーク機器は、隣接するネットワーク機器の設定情報と矛盾する設定情報を許容してしまい、ネットワークオペレータの意図と反する設定情報でも反映される。

これらの問題によって起こる障害を減らす試みの一つに、設計書をコンピュータが読み込み可能な形にすることで、コンピュータがその情報に基づいて設定作業を行う手法がある。

しかし、設定作業のフローが必ずしも、いつも同じやり方とは限らない場合がある。ネットワークオペレータがその場で状況を判断し設定を加える場合も容易に考えられる。そのため、ネットワーク機器に設定を行う前に矛盾がないか検証を行うことが可能である必要がある。

1.2 目的

あるネットワーク機器に対して他のネットワーク機器の設定情報と矛盾がないよう設定作業を行うには、機器に投入する未反映の設定情報と、ネットワークに属する全ての機器の反映済みの設定情報との間に矛盾がないか検査を行う必要がある。この設定情報に矛盾がないか検査を行うことを本論文では設定情報の一貫性検査と呼ぶ。

本論文では、対象とするネットワークに属する機器の設定情報をネットワークレイヤ毎に記述し、一貫性検査を行う手法を提案する。

ネットワークレイヤ毎に記述された複数のネットワーク機器の設定情報を検査することで、レイヤを跨った設定情報であっても一貫性検査を行うことができ、機器間で矛盾する設定情報を許容してしまう問題を解決することが可能であると考える。したがって提案手法により、ネットワークオペレータの誤認識による誤った設定作業を防止することに貢献できる。

1.3 本論文の構成

本論文は、本章を含めて 7 章から構成される。2 章では、ネットワークの構築と運用における環境とその課題について述べる。また、それらの課題における要求の変化について述べる。3 章では、ネットワーク検査とネットワーク記述言語に関連する研究を説明し、2 章で述べた課題に対して不足している機能を明らかにする。4 章では、マルチレイヤネットワークに対応した設定情報の矛盾検査手法について提案し、その適用例を示す。5 章では、提案した手法のシステムに基づくシステムの設計と実装について述べる。6 章では、提案手法の評価実験、性能評価、関連研究との比較について述べる。7 章では、本論文のまとめと今後の課題について述べる。

第2章　ネットワークの構築と運用における環境と課題

本章では、今日のネットワークに対するオペレーションの環境と、その環境における要求の変化について述べる。さらにその変化によって伴う問題について触れる。

2.1　構築と運用におけるヒューマンエラー

ネットワークの誤作動によって起こる web サイトへのアクセス障害がメディアで報じられることは少なくない。2012 年 2 月には、アメリカで Amazon 社におけるクラウドサービス内のロードバランサの設定作業中に、ロードバランシングのためのステートデータを誤って削除してしまい、約 12 時間の間、動画配信等のサービスが利用不可能になっている。また、国内では 2013 年に KDDI 社で、手順書の記載ミスからユーザ認証サーバの設定を誤ってしまい、約二日間 E メールのサービスが利用不可能になっている。

これらの原因に、ネットワーク機器への設定作業時におけるヒューマンエラーが挙げられる。インターネット初期には、ネットワークは比較的単純な構造であったため、ネットワークオペレータがネットワークの構築または運用を行う際にネットワーク機器一つ一つに対して手作業で設定を行ってきた。しかし、ネットワークの仮想化技術やトンネル技術の進歩によって、ネットワーク機器の設定情報は複雑かつ大きくなっている。したがって、ネットワークオペレータが手作業で誤りなく設定作業を行うことは困難を極める。同時に、ネットワークで障害が発生する可能性が高まり、品質の低下または最悪の場合だとネットワークが停止してしまう場合がある。 誤りなく設定作業を行うことが困難であることは、人間の誤認識やキーボードの誤入力に起因する。したがってヒューマンエラーが根本にある原因だといえる。

ヒューマンエラーは人間工学の分野では「システムによって定義された許容範囲をこえる人間行動の集合」と定義されている。ヒューマンエラーは人間が全く意図していないに

も関わらず発生するものと、人間が意図的に行動したために発生するものの二つに分類される。ネットワークの構築と運用において、前者はネットワークオペレータの設計書に対する誤認識やキーボードの誤入力が挙げられる。また後者はネットワークオペレータの自身の経験からくる誤った判断や、時間に追われてより楽な方法を取ることが挙げられる。どちらもネットワークオペレータの行動と様々な環境の要因から発生するため、ヒューマンエラーを予防しネットワークの障害を完全になくすのは難しい。

しかし、人間が全く意図していないにも関わらず発生してしまうヒューマンエラーは人間とコンピュータの両方が確認作業を行うことで減らすことが可能である。

2.2 ネットワーク機器における設定情報の構造

ネットワーク機器に設定情報を投入する前にコンピュータが確認作業を行うための前提として、ネットワーク機器の設定情報がどのような形式で記述されているのか述べる。

ネットワーク機器の設定情報は一般的にプロトコル毎に一つのファイルに記述されている。ネットワーク機器の OS は FreeBSD や Linux 等の汎用的な OS をベースにして作られた物が大半なため、プロトコル毎に設定された設定情報を参照し、それぞれのプロトコルがアプリケーションとして実行される。そのため、ユーザが複数のアプリケーションの設定を一つのファイルで管理できるというメリットがある一方、それぞれのアプリケーションにおける記述の依存関係を管理しにくいというデメリットがある。依存関係を壊すような設定を行った場合、設定時にエラーとして返される場合もあるが、稼働した後にログを確認してからでないと正しく動作していないことが判明しない場合もある。

また、様々なネットワーク仮想化技術が開発されているため、ネットワーク機器の設定情報が複雑化する傾向にある。その背景にサーバの仮想化技術が進歩し、ネットワーク側もそれらの機能に対応せざるを得なくなったことが挙げられる。サーバ仮想化による一つのサーバ上で論理的に複数のサーバが存在するように見せるマルチテナント技術が導入された結果、一つのサーバに複数のネットワークインターフェースが必要となり、一本のケーブルを論理的に複数のケーブルに見せるような技術が開発された。そのため、新たな技術が開発される度に、ネットワーク機器に機能追加され設定情報が大きくなり、より複雑になっている現状がある。

2.3　設定情報の変更によるネットワークへの影響

前節では、ネットワーク機器単体における設定情報の構造とプロトコルの依存関係について述べた。本節では、複数のネットワーク機器間における設定情報の変更による影響について述べる。

ネットワーク機器には、隣接するネットワーク機器の設定情報と矛盾する設定情報を許容してしまい、ネットワークオペレータの意図と反する設定情報でも反映されてしまう問題がある。この問題はコンピュータネットワークが分散システムである以上、自明ではあるが、日々進歩するそれらをどう扱うか定められていないため、ネットワークオペレータにとって設置情報を管理するのは難しい問題である。

コンピュータネットワークにおいて通信プロトコルは OSI 参照モデルに定義されているように七つのレイヤに分離されている。それぞれのレイヤは異なる役割を持ち、同様に積み上げられたネットワークの機能を実現するソフトウェア群をプロトコルスタックという。上位のレイヤに属するプロトコルは下位レイヤに属するプロトコルが正しく動作しない限り動作が保証されない。そのため、ネットワーク機器の設定情報が変更された場合の影響範囲は、機器自身と隣接するネットワーク機器の同じプロトコルとその上位レイヤに属するプロトコルの動作である。

ネットワーク上で二台のコンピュータがそれぞれ通信を行う場合の OSI 参照モデルに従ったモデルを図 2.1 に示す。片方のコンピュータのあるレイヤにおける設定を誤ってしまった場合、正しくプロトコルに乗っ取って通信を行えないため、影響はもう一方のコンピュータにも及ぶ。加えて、全ての機器において誤って設定した設定情報のレイヤより上位のレイヤに含まれる設定情報通りに動作する保証はない。

また、ネットワーク上で二台のコンピュータと一台のルータがそれぞれ通信を行う場合の OSI 参照モデルに従ったモデルを図 2.2。中央のデータを中継するルータの設定を誤ってしまった場合の影響は残りの二台にも及ぶ。また二台の場合と同様にすべての機器において、誤って設定した設定情報のレイヤより上位のレイヤに含まれる設定情報通りに正しく動作する保証はない。

そのため、ネットワーク機器に誤った設定情報を反映してしまった場合の影響範囲は、反映した設定情報に該当するプロトコルのレイヤとその上位レイヤにあたるプロトコルの設定情報を含む論理的に隣接する全てのネットワーク機器である。

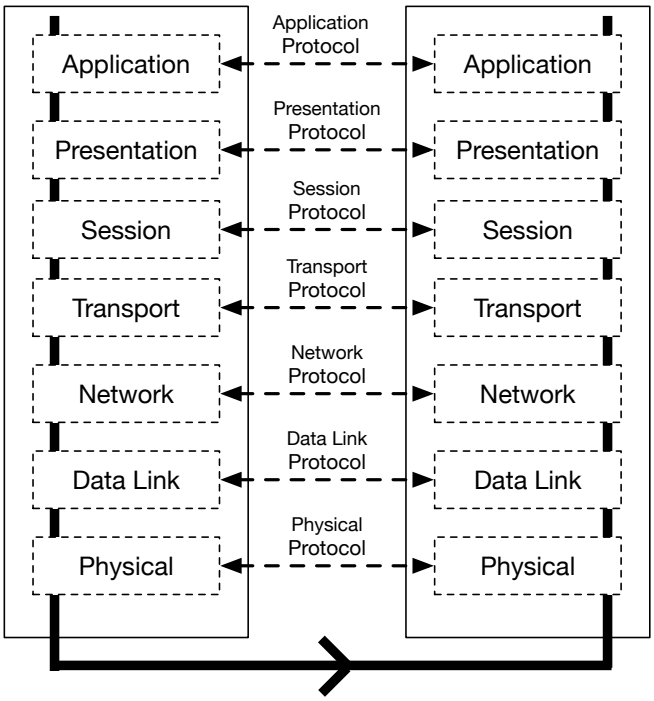

図 2.1: OSI 参照モデル 二台

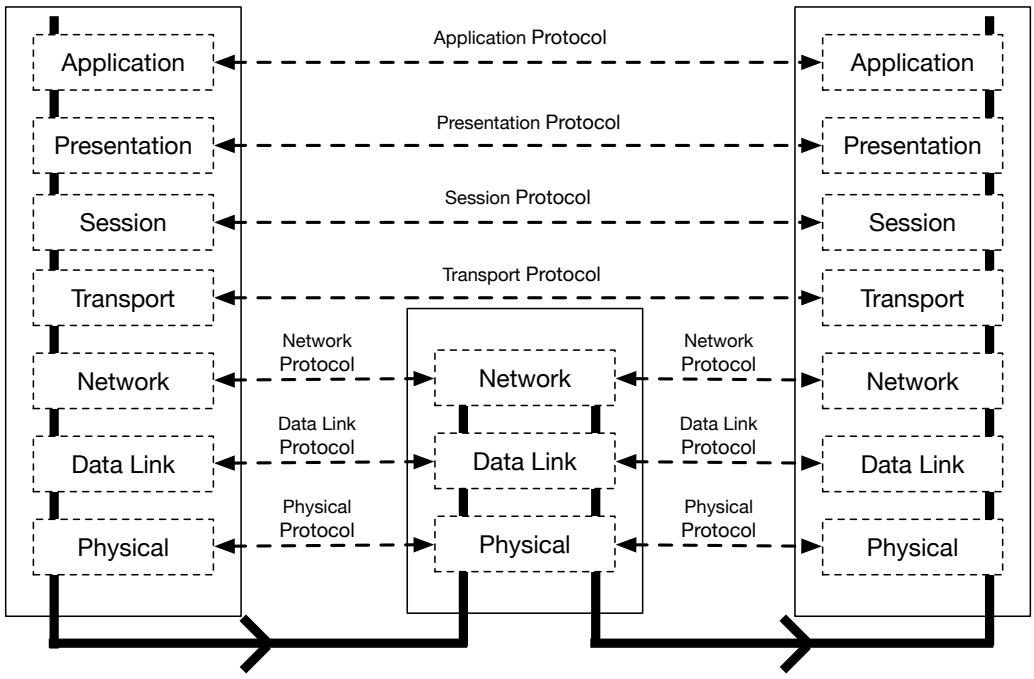

図 2.2: OSI 参照モデル 三台

2.4 ヒューマンエラーによる設定情報の誤りにおける対策と要求事項

本節では、2.2、2.3 節で述べた問題における要求事項について述べる。

一般的なネットワーク機器において設定情報はプロトコル毎に一つのファイルに記述されるため、ネットワークオペレータがそれぞれのプロトコルの設定の依存関係の管理が難しいという問題がある。また、ネットワーク機器には、隣接するネットワーク機器の設定情報と矛盾する設定情報を許容してしまい、ネットワークオペレータの意図と反する設定情報でも反映されてしまう問題がある。

これらの問題を解決する一つの方法として、ネットワーク機器単体に対して設定作業を行うという従来の作業方法ではなく、単一のネットワークスライスに対して設定を行う作業方法が考えられる。ネットワークスライスとは、ネットワークを機能毎に切り分ける概念である。スライスとスライスは互いに機能が干渉することはない。そのため、ネットワークの構築や運用の際の問題の切り分けが容易になるという利点がある。一方で、スライス間で機能が干渉することはないが、あるスライスが稼働することはその下のスライスが稼働していることが前提である。つまり下位のスライスが稼働しない限り、上位のスライスは稼働することが出来ない。

NETCONF は、単一のネットワークスライスに対して設定を行うことを目的としたプロトコルである。そのため、複数のネットワーク機器に対して並列に設定を行うことが可能である。よって、ネットワークオペレータが隣接するネットワーク機器の設定情報における依存関係を把握することが容易である。しかし、機能として設定の投入をすることが目的であるため、下位のレイヤが稼働しているかどうか確認することには関与しない。

したがって、設定をネットワーク機器に反映させる前に投入する予定の設定を、隣接するネットワーク機器の設定情報と照合を行い、ネットワークレイヤを横断して矛盾が発生しないか確認することが必要である。以下に設定情報の誤りにおける要求事項を示す。

- ネットワーク機器に設定を投入する前に矛盾がないか検査が可能である
- ネットワークスライス毎に設定を記述可能である
- 一度に複数の機器に対して設定を行うことが可能である

第3章　関連研究

ネットワーク機器に対して、ネットワークオペレータによって CLI を介した直接的な制御、またはソフトウェアによる制御を行った場合、小さな設定情報の変更であったとしてもネットワーク全体に影響する場合があり、その結果としてネットワークに障害を引き起こす可能性がある。本章では、ネットワーク機器における設定情報の誤った変更を発見するための手段について述べる。

3.1　ネットワークの検査

ネットワークの挙動を検査するツールは多数ある。伝統的なネットワークの動作を確認するソフトウェアに ping [6] や traceroute [7]、tcpdump [8] があり、現在もネットワークの構築または運用の際には頻繁に使用されている。しかし、これらのソフトウェアはネットワークの部分的な動作を検証するには非常に有用であるが、ネットワーク全体の動作を検証するには不向きである。

ネットワーク全体における動作の「正しさ」を検査するには、構成するネットワーク機器群の情報を一箇所に収集した後、収集した情報を元に検査を行う必要がある。本論文におけるネットワークの動作の「正しさ」とは、ネットワークオペレータの意図通りに動作するということではなく、複数のネットワークでプロトコルが正常に動作し、ネットワーク障害が起きないことである。

ネットワーク障害を未然に防ぐ方法、または動作中のネットワークに対して障害を発見する手法は複数挙げられる。しかし、それらはそれぞれが着目するネットワーク機器内の機能が異なる。図 3.1 はネットワーク機器に設定情報が与えられてから、ネットワーク上での動作に反映されるまでの流れを示す。本節では、ネットワーク機器の各レベルでどのような検査が行われているか述べる。

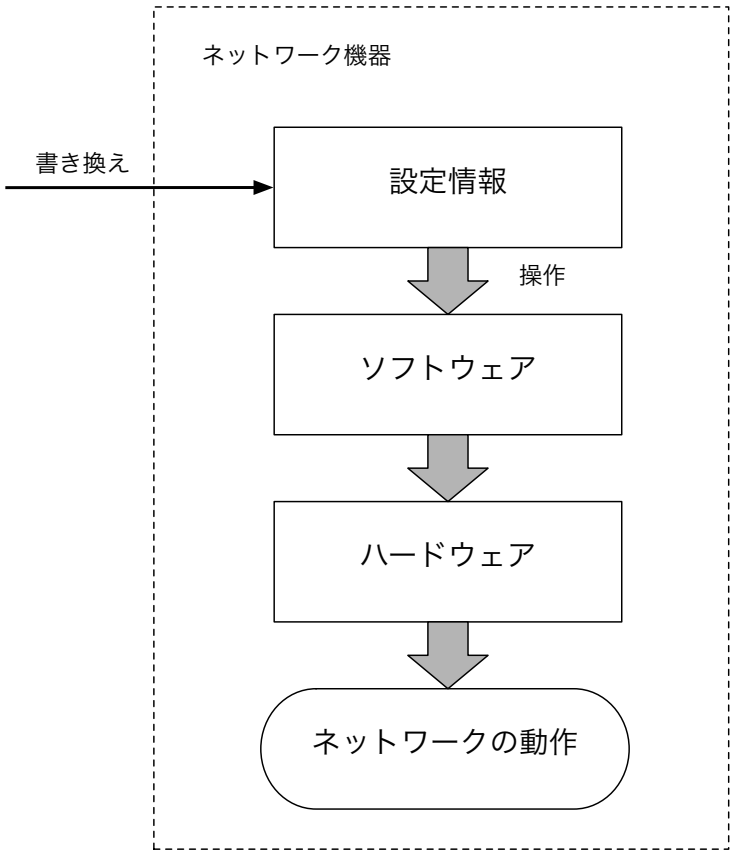

図 3.1: ネットワーク機器の動作の流れ

3.1.1 設定情報に関する検査手法

ネットワーク機器の設定情報を検査する手法は、同じ環境を別途用意する以外でネットワークを稼働させる前にネットワークの動作を予測できる唯一の方法である。ネットワーク機器において設定情報はネットワークオペレータが直接操作するので、ネットワークオペレータの誤りに一番に影響する箇所である。

ベンダー毎に様々な設定の記述方式が存在するため、検査することが困難であることが予想されるが、Ari らは、ベンダー毎の設定情報を統一したフォーマットにしてネットワークの安全性を解析する Batfish という設定情報の解析ツールを提案している [2]。 また、Sanjay らは、ネッワーク機器の VPN の設定情報からインターフェースに最適な MTU の値を算出する手法を提案している [3]。

しかし、設定情報の検査を行ってもネットワーク機器に搭載する機器にバグがあればネットワークは正常に動作しないため、他の検査手法と併用して用いる必要がある。

設定情報の検査は、主に設定情報を形式的にモデル化して、プロトコル毎の記述を検査するものがある。また、論理推論を行うためにネットワーク機器の設定情報を論理式に変換しソルバで解くものが挙げられる。

3.1.2　ソフトウェア・ハードウェアに関する検査手法

ネットワーク機器に搭載されるソフトウェアまたはハードウェアにバグがある場合は、ネットワークオペレータがその情報を知っていない限り、回避することは出来ない。そのため、ソフトウェアとハードウェアのバグは、ベンダがテストを行うだけでは十分ではない。実際に稼働させるネットワークではなく、投入する予定の設定情報をネットワーク機器に反映させて動作させてみる事や、実際に起こりうる事象に対して網羅的にテストを行う必要がある。

3.1.3　ネットワーク上での動作に関する検査手法

ネットワーク機器の動作を検査する手法は、ネットワーク稼働時の振る舞いから検査を行う。その振る舞いを図る尺度としてのパラメータにはネットワークインターフェースの I/O や、経路テーブル、MAC アドレステーブルが使われている。その中でも代表的なネットワークの動作の検査を行う手法である Anteater [4] について解説を行う。

Anteater

Anteater はネットワークに属するネットワーク機器の IPv4 の経路テーブルを SNMP を用いて収集し、SAT(充足可能性問題) に変換し、ネットワークの検査を行う。複数の経路テーブルを用いて検査を行う項目を以下に示す。これらの検査項目を SAT ソルバで解くことで、障害の特定を行うことが可能となる。

- ループが起きていないか
- フレームのロスが起きていないか
- 異なる冗長化されたネットワーク機器間で同じ経路テーブルを保持しているか

オペレータが SAT を記述する必要がないため、SNMP が有効であれば利用可能である。しかし、経路テーブルが増えた場合、SAT が大きくなり、解くのに膨大な時間がかかる。

3.2 ネットワーク記述モデル

設定情報の検査を行う際にネットワークのモデル化を行うことは不可欠である。本章ではネットワーク記述モデルに対する要求と NDL、VRD、INDL、YANG Data Model for Network Topology ついて説明する。

ネットワーク記述モデルには Information Model (IM) と Data Model (DM) がある。IM は、人と人との間の意思疎通のためのデータの文書化、組織化としてどのようにデータを格納するかを表現するものである。DM は、データの意味を記述するものである。データの意味とは、関係性やルールのことである。本論文ではネットワーク機器の設定情報を扱うので DM について説明する。

ネットワーク記述モデルは、インターネットの普及に伴い、複雑になるネットワークを把握するためのものである。従来、ネットワークの構成を記述する場合、定まった記述方式はなく、場合に応じて独自の方法で記述されていた。そのため、異なるプロジェクト間で相互運用を行うのに問題があった。それは、ネットワークに様々なデバイスが接続される、また大規模で不確定要素が多く、忠実なモデルの構築が難しいとされてきたためである。現在では、様々な団体が複雑さを軽減させるためにはどうのように抽象化されたモデルを表現するか、標準化のための活動を推めている。このような活動が推められることで、ネットワークの設定の自動化やソフトウェア上での扱いが容易になることが考えられる。

ネットワーク記述言語に関して、2.4 章の要求事項より、ネットワーク機器の設定情報を用いてネットワークトポロジを記述、またその情報を用いて検査するには以下の要件が必要である。

- 記述分離性: 物理ネットワーク構成と仮想ネットワークの構成を独立して記述可能であること
- 物仮記述性: 物理ネットワークの構成情報と仮想ネットワークの構成情報を分離して保持可能であること
- 再現可能性: 分離して保持した場合でも各ネットワークの構成情報を表現可能であること

3.2.1 NDL

NDL(Network Description Language) は 2005 年にネットワークが将来に向けて複雑化することを予測し、その複雑さを軽減するために作成されたものである [9]。人とコンピュータの両方が進歩するネットワークをより把握し、時間のかかる単調な作業を減らすのが目的である。

NDL は、ネットワークを記述するためのスキーマに RDF (Resource Description Framework) を用いている。RDF は、複数リソースの関係を記述するための枠組みである。NDL は光ファイバネットワークを対象としているため、実験ネットワークを構築する場合は、ネットワーク提供者とエンドユーザの間で連携が必要であったが、統一された手法はなかった。NDL はこの問題のために作られた記述言語であり、プロジェクト内で共通の記述方式を使用することで、より円滑な進行が実現する。また、それらの記述された情報は相互に関連付けが可能なため、ドメイン間の接続等に用いることが可能である。図 3.2 は NDL のスキーマである。

NDL の問題点として、物理ネットワークのみ記述可能であることが挙げられる。そのため、ノードの仮想化が表現できない。また、物理ネットワークと仮想ネットワークを分離して記述できない。そのため、OSI 参照モデルにおける Data Link 層の LAN と VLAN が同じ記述方式になってしまう。

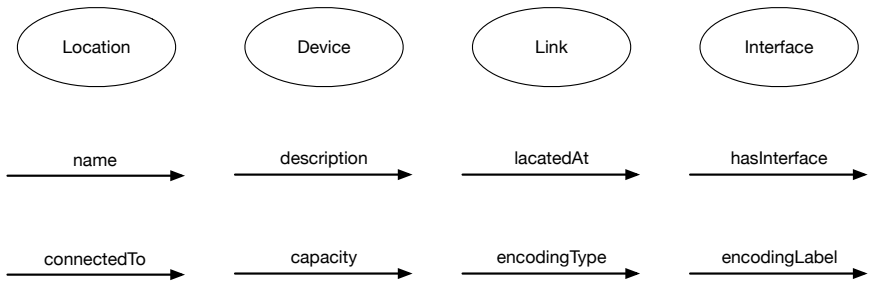

図 3.2: NDL の topology scheme

3.2.2 INDL

INDL(Infrastructire and Network Description Language) は、2012 年にデバイスの機能に依存しない記述を提供することを目的に作られた [10]。また、コンピュータインフラストラクチャにおける仮想化の概念を把握し、リソースの能力も記述が可能である。INDL は、記述に関して NDL の理念を受け継いで作られたため、NDL や、他の NDL を基にして作成された記述言語と関連付けて相互に利用可能である。図 3.3 は、INDL の全体の構造を示す構造木である。

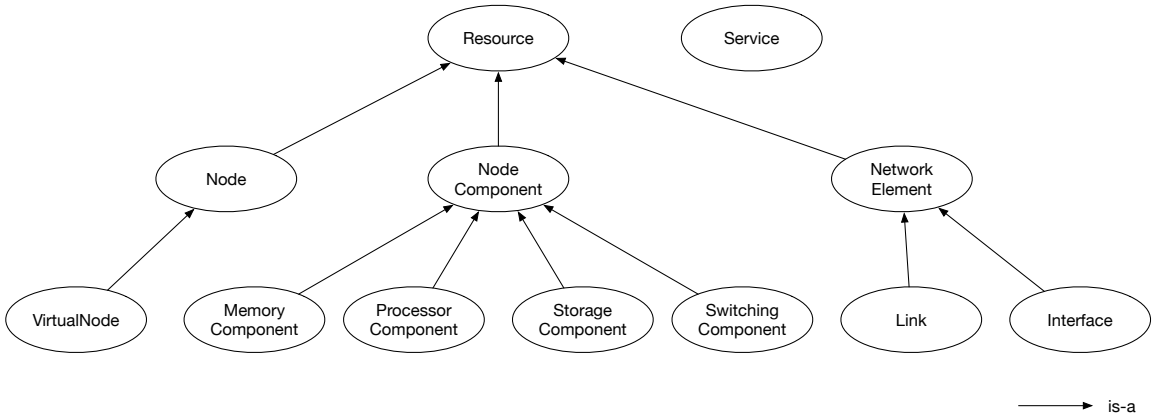

図 3.3: INDL の構造木

3.2.3 Multi-Layered Network Description Model

マルチレイヤネットワークモデルは、ネットワークシステムに対して機器の機能特徴を失うことなく、仮想環境も論理的に表現可能なマルチレイヤネットワークモデルである [11]。グラフ理論でのグラフ $G = (V, E)$ を拡張したものであり、ノードとリンクの集合と四つの写像で表現される。図 3.4 のように一つの機器は一つのノードで表さず、機能の要素としてノードが各レイヤに存在する。よって、それらのノードの部分グラフとして一つのネットワーク機器が表現される。

各レイヤのノードは、三つの情報を持っている。一つ目がレイヤの情報、二つ目がデータを中継するノードであるかの情報、三つ目がレイヤ上で一意に識別可能な情報である

IP アドレスや MAC アドレス、ポート番号等の情報である。そのため、各レイヤで論理的な接続関係を記述することが可能である。

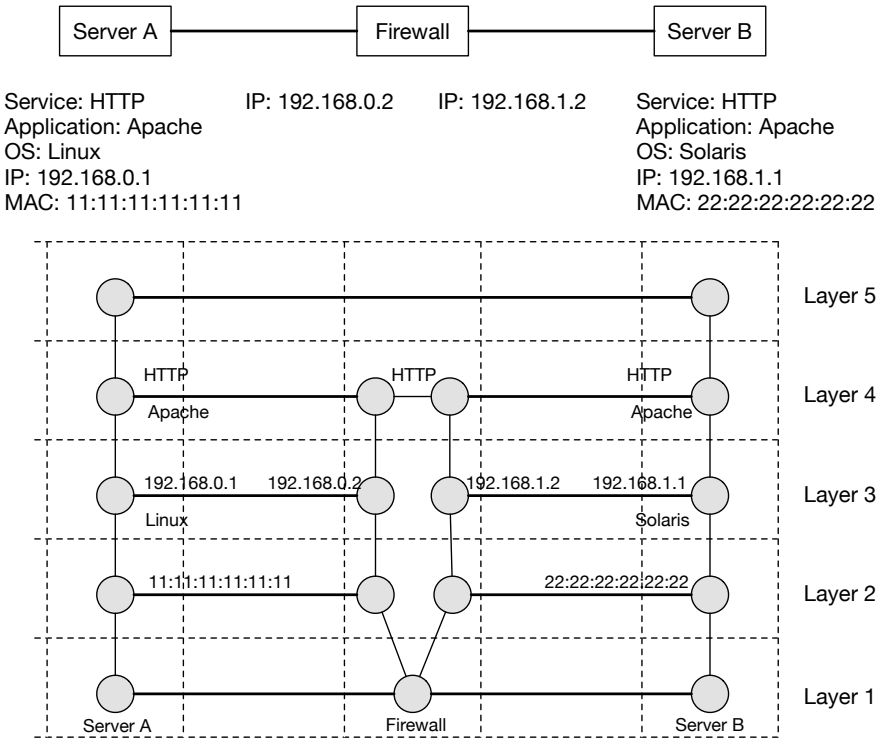

図 3.4: Multi-Layered Network Description Model の表現例

3.2.4 YANG Data Model for Network Topology

YANG Data Model for Network Topology は、YANG(Yet Another Next Generation) というデータモデルでネットワークトポロジを表現することを目的とした IETF のドラフトである [12]。

YANG は、RFC6020[13] で規定されたネットワーク管理機器の管理プロトコルである NETCONF で使用される設定項目のデータフォーマットである。ネットワーク機器の設定項目や状態などの木構造化された情報を人が理解しやすい形式で記述することが YANG の目的である。

3.3　ネットワークの検査における問題点

本章では、既存のネットワーク検査手法における問題点を挙げ、その問題に対する要求事項について議論する。

ネットワークの設計を行ってからネットワーク機器に対して設定作業を行う工程で、ヒューマンエラーによって起こるネットワーク障害の原因は、ネットワークオペレータの設計書に対する認識の誤りや、設定作業中のキーボードのタイプミスである。これらは、ネットワーク機器に設定を反映させる前に人間もしくはコンピュータが誤りに気付くことができれば誤りを訂正しネットワーク障害を未然に防止することが可能である。

提案されている設定情報の検査手法 [2][3] は単一のプロトコルまたは、定められた複数のプロトコルにおける設定情報の該当箇所をモデル化することで矛盾がないか検査を行う。しかし、提案されている設定情報の検査手法は設定項目を書き換える頻度が多い場合は、少しの変更であっても繰り返すと、ネットワーク全体で見れば大きな変更を異なる設定項目に行っている場合があるため、効果が得られない可能性がある。

また、データセンタのようなネットワークではユーザの要求に迅速に答えるため、web ページ上のクリック一つでネットワークを含むサーバやアプライアンスの構成が変更されることがある。そのようなソフトウェアによって頻繁に設定情報を書き換えられることが頻繁に行われる場合にも対応できない。

第4章　マルチレイヤに対応した一貫性検査手法の提案

ネットワークの検査とは、ネットワークに属する複数の機器の振る舞いを様々な面から観測し、その検証を行うことである。ネットワーク機器の設定情報やソフトウエア、ハードウエアに加えて、動作するプロトコル等に対して検査を行う手法が存在する。

前章では、ネットワークオペレータの誤りによって起こるネットワーク障害を予防するためには、設定情報の検査が必要であることを示した。また、既存の手法ではサービスに対してネットワークが頻繁に組み替えられるようなネットワークに検査を行うことが出来ないことを指摘した。本章では、そのようなネットワークに対して設定情報の検査を行うにはどのような機能が必要か提案し議論する。

4.1　設定情報の一貫性検査と有用性

本論文で述べる一貫性検査とは、複数のデータの間で意味する事柄に矛盾が発生していないか調べることである。つまり設定情報の一貫性検査とは、複数のネットワーク機器の設定情報の中で、対応して記述されるべき箇所に矛盾がないか調べることである。

誤った設定をネットワーク機器に投入する前に一貫性検査を行うことによって、ネットワークが稼働する前に誤りを検出することができ、ネットワークオペレータに誤りを認識させることが可能である。そのため、設定情報の一貫性検査はヒューマンエラーの対策として有用であると言える。

また、ネットワークオペレータのヒューマンエラーによって発生する設定情報の誤りは、前章で述べたネットワークの静的検査手法のように複数のネットワーク機器の設定情報を参照し検査することで検知することが可能である。しかしこれらの手法は、設定情報全体に対して検査を行っているのではなく、あるプロトコルや単一のネットワークレイヤの設定情報を検査するものであった。

4.2　マルチレイヤネットワークへの対応

マルチレイヤネットワークとは、ネットワークの階層構造において複数のレイヤでデータの転送が可能なネットワークのことである。コンピュータが持つ通信機能の OSI 参照モデルで定義されているように、あるレイヤは下位レイヤによって抽象化された情報が与えられ、上位レイヤに対して自身のレイヤの実装の出力を抽象化し渡す。各レイヤの要件に対する実装が存在し、それらをレイヤ毎に隠蔽することで様々な技術が複雑に組み合わさることを防いでいる。そのため、サービスごとにネットワークを組み替えを行う際、あるネットワーク構成を別のネットワークの一部として組み替えるため、既存の一貫性検査では対応出来ない。 つまり既存の手法では、あるプロトコルや単一のネットワークレイヤに対してのみ検査を行うので、ネットワークレイヤを横断して設定情報を検査するには不十分である。

従来の設定情報の検査手法は、前節で述べたようにネットワーク機器群の設定情報に含まれる単一のプロトコルに対して一貫性検査を行う。それらの手法は、限定されたレイヤまたはサービス上でのみで行われているため、複数のプロトコルやネットワークレイヤを横断して一貫性を担保することが出来ない。したがって、複数のネットワークレイヤとの依存関係を持つネットワーク仮想化技術やトンネル技術に対して効果が得られない。そのため、提案手法では検査の対象となるネットワークレイヤから、その下位レイヤにあたるネットワークレイヤを再帰的に検査を行う必要がある。

4.2.1　単一レイヤの構成

設定情報の一貫性検査をマルチレイヤネットワークに対応させるにあたって、単一のネットワークレイヤが提案手法でどのように表現されるか述べる。図 4.1 は、モデル化を行うネットワークとそれに属する機器の簡略化された設定情報である。また、図 4.2 は、提案手法において単一レイヤの構成を示す。

Protocol Function

Protocol Function は、属するレイヤ上でのプロトコルの要素を表す。各レイヤの Protocol Function は四つの情報を持つ。一つ目がどのネットワーク機器の設定情報から作成

されたのかを示す情報、二つ目が作成される基となった設定情報群、三つ目が属するレイヤの情報、四つ目がが属するレイヤ上で一意な識別子である。

Protocol Channel

Protocol Channel は、同一レイヤ上で二つの Protocol Function を結ぶためのものである。異なるネットワーク機器間でデータの到達性があることを示す。Protocol Channel は、四つの情報を持つ。一つ目は二つの Protocol Function の識別子、二つ目が作成される基となった設定情報群、三つ目が属するレイヤの情報、四つ目が属するレイヤ上で一意な識別子である。

Protocol Relay

Protocol Relay は、同一レイヤ上で二つの Protocol Channel を結ぶためのものである。同一のネットワーク機器間でデータの到達性があることを示す。ルータではネットワークレイヤで表現され、スイッチではデータリンク層で表現される。Protocol Relay は、四つの情報を持つ。一つ目は二つの Protocol Function の識別子、二つ目が作成される基となった設定情報群、三つ目が属するレイヤの情報、四つ目が属するレイヤ上で一意な識別子である。

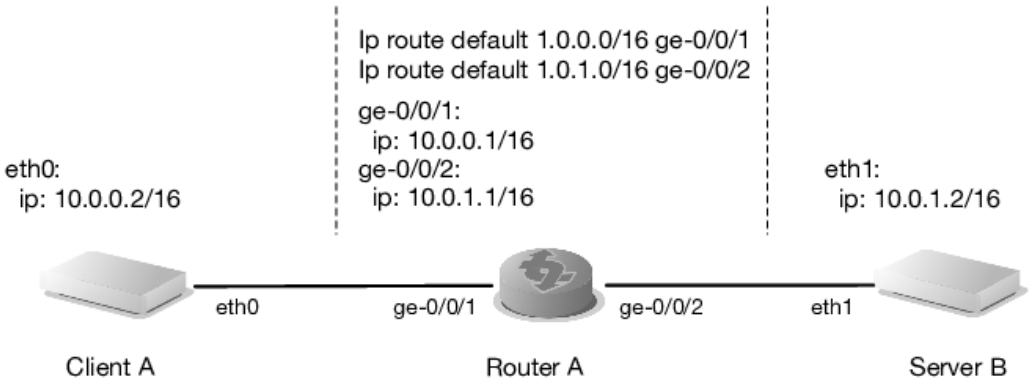

図 4.1: サンプルネットワーク (IP)

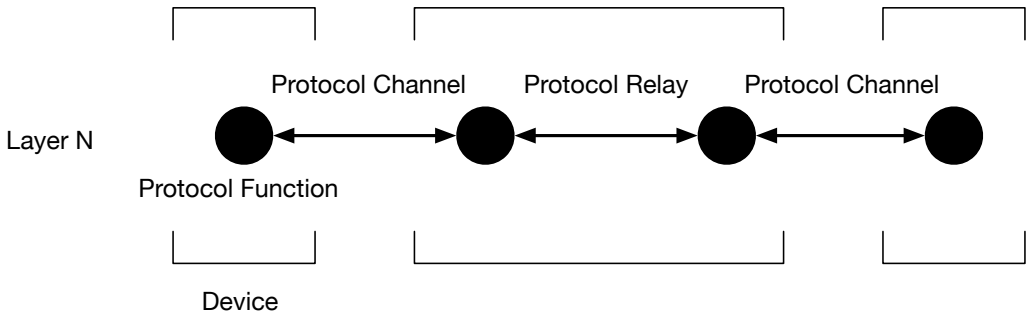

図 4.2: 提案手法におけるネットワークレイヤの構成

4.2.2 レイヤ間の構成と関係性

ネットワークレイヤにおける階層構造は小さな機能が下から上に順に積み重なって一つの大きな機能を構成している。そのため下位レイヤから上位レイヤに向けて 再帰的に検査を行うには、上位レイヤの Protocol Function または Protocol Channel が下位レイヤでどの機能を担っていたか判別できる必要がある。そのため、Protocol Function と Protocol Channel は少なくとも一つの下位レイヤの属すものに対して依存関係を持つ。依存関係の向きは上位レイヤから下位レイヤである。図 4.3 は、モデル化を行うネットワークとそれに属する機器の簡略化された設定情報である。図 4.4、4.5 は、Protocol Function と Protocol Channel における依存関係を表す。

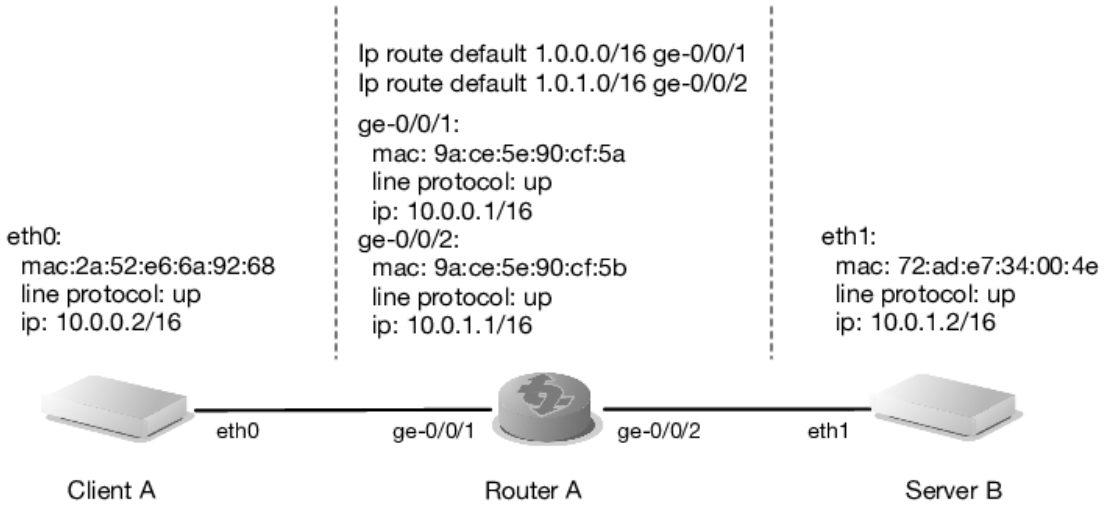

図 4.3: サンプルネットワーク (MAC + IP)

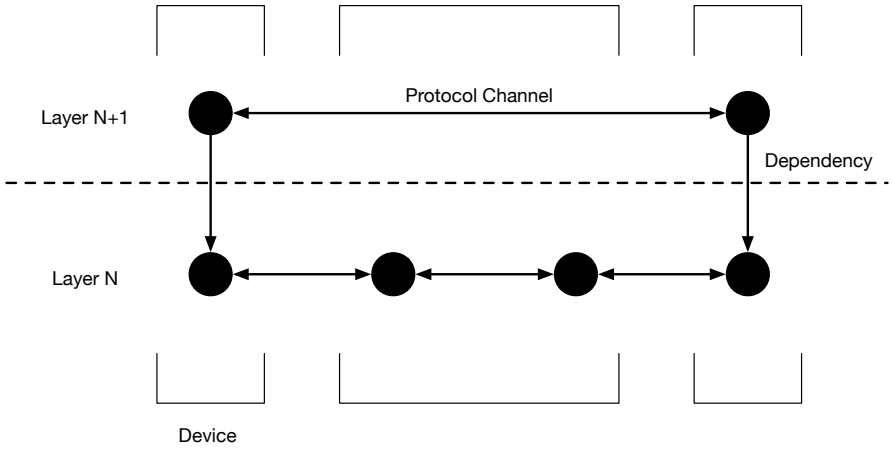

図 4.4: Protocol Function における依存関係

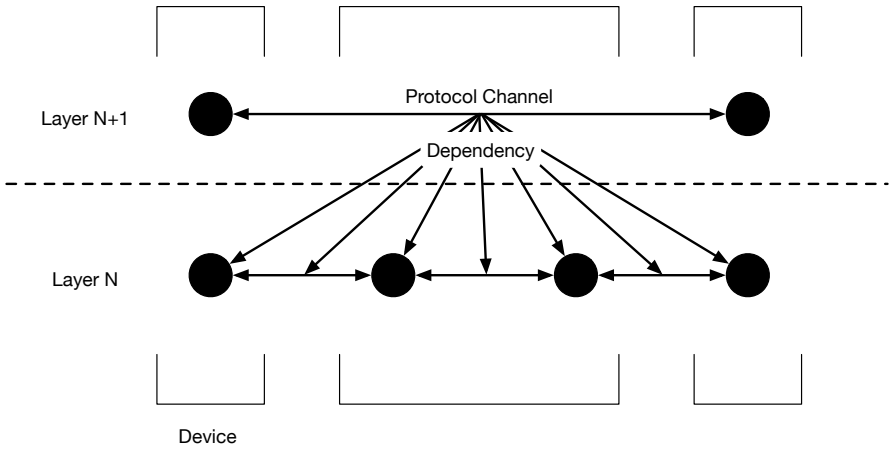

図 4.5: Protocol Channel における依存関係

4.2.3 レイヤ間におけるネットワーク仮想化技術についての考察

ネットワーク仮想化技術とトンネル技術におけるレイヤ構造を、従来のネットワークレイヤと同様に単一のレイヤとして扱う。ネットワーク仮想化技術はサーバ仮想化技術が進歩した結果、開発されたものである。つまり、サーバ仮想化における恩恵を活かすためのネットワーク側への要求から後になって出来た技術である。よってネットワーク仮想化技術の代表として挙げる VLAN はレイヤ 2.5 とも呼ばれる。また、トンネル技術は下位レイヤにおける二つのノードを上位レイヤで論理的に一つのノードのように扱うことができる。したがって、ネットワーク仮想化技術とトンネル技術をネットワークレイヤで表現する場合は、新たにレイヤを作成し既存のレイヤと同様に扱うことで一般的に扱うことが

可能である。

しかし、ネットワーク仮想化技術とトンネル技術において、上位レイヤと下位レイヤの Protocol Function または Protocol Channel は、依存関係が一対一に対応しないことがある。その場合、下位レイヤで論理的に一つに見えている要素が上位レイヤで二つ以上になることを多重化と呼ぶ。一方で下位レイヤで二つ以上に見えている要素が上位レイヤで一つになることを集約と呼ぶ。

モデル化された結果、ネットワーク仮想化技術毎の差異は次節で述べるレイヤ毎に定義するルールによって吸収されるため、多重化と集約を用いることでネットワーク仮想化技術を汎化的にモデル化することが可能である。

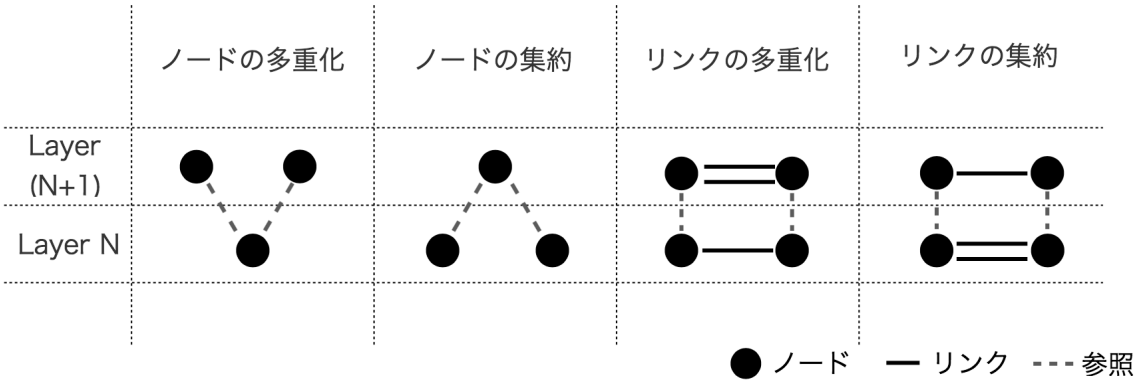

図 4.6: ネットワーク仮想化技術における多重化と集約

4.3 ルールに基づく検査手法

本節では、ルールに基づいて行う設定情報の検査について述べる。

提案手法によるモデルは、設定情報から生成される。そのため設定情報に矛盾がないか検査するには、設定情報からモデルを作る際に条件を設定し、それに基づいて検査する必要がある。

あるレイヤにおいて、Protocol Function、Protocol Channel、Protocol Relay は、依存関係を持つ下位レイヤの要素が存在しなければならない。したがって、上位レイヤの一貫性は下位レイヤの一貫性の上に成立する。そのため、検査を行う際は下位レイヤから上位レイヤに向かって検査を行う必要がある。したがって、上位レイヤになるほど支えられてるレイヤの数が多くなるので、一貫性を担保するのは難しくなる。逆に、下位レイヤにな

るほど下位レイヤの数が少なくなるので、一貫性を担保するのは容易である。図 4.7 は、マルチレイヤネットワークの依存の向きと検査の向きを示す。

提案する設定情報の一貫性検査は、ネットワーク機器の設定情報をネットワークレイヤ毎に分離し、レイヤ内のルールとレイヤ間のルールに従って行う。これらのルールは、設定情報からレイヤ毎にモデル化を行うための条件であり、それぞれのレイヤとレイヤ間で異なるルールを持つ。

レイヤ内のルールは、単一のレイヤ内の Protocol Function、Protocol Channel、Protocol Relay をモデル化するためのルールである。レイヤ間のルールは、二つのレイヤの関係を示すものである。そのため、Protocol Function、Protocol Channel、Protocol Relay の依存関係はレイヤ間のルールで示される。図 4.8 は、任意のレイヤを検査する場合のレイヤ内のルールとレイヤ間のルールの検査範囲である。例えば、レイヤ C における一貫性の検査を行う場合、レイヤ A、B、C のルールとそれらのレイヤ間のルールに基づいて、最も下位のレイヤからモデル化を行う。また、レイヤ E における一貫性の検査を行う場合は、レイヤ A、B、C、D、E のルールとそれらのレイヤ間のルールに基づいて最も下位のレイヤからモデル化を行う。

このようにルールに基づいて検査を行うことで、ネットワークの変更に伴ってレイヤの構造が変わった場合でも、レイヤ内のルールを変更する必要はなく、レイヤ間のルールの追加だけで設定情報の同様の検査を行うことが可能である。また、新たなネットワーク仮想化技術によって、新規のレイヤが作られる場合でも、既存のレイヤ内のルールに影響を与える事はない。

4.3.1 レイヤ内のルール

レイヤ内のルールは、単一のレイヤにおいて Protocol Function、Protocol Channel、Protocol Relay が成立するための条件である。また、レイヤ内のルールは大きく二つに分類することが出来る。以下にその分類を示す。

- 設定情報に含まれているべき項目についてのルール
- 異なるネットワーク機器の設定情報の間で満たすべきルール

設定情報に含まれるているべき項目についてのルールは、設定情報から Protocol Function と Protocol Relay を作るためのものである。一方、異なるネットワーク機器の設定

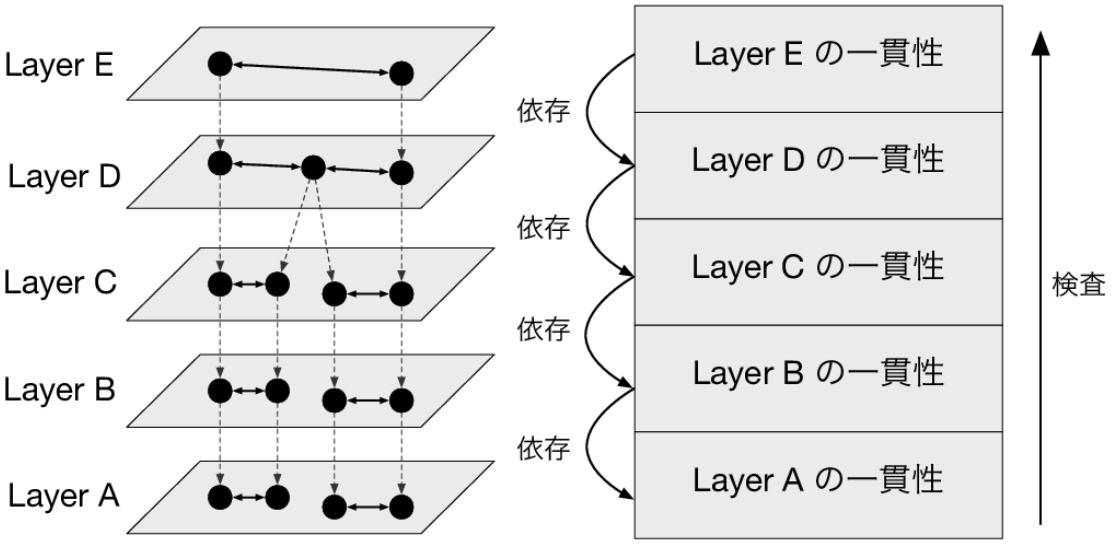

図 4.7: マルチレイヤネットワークにおける一貫性検査

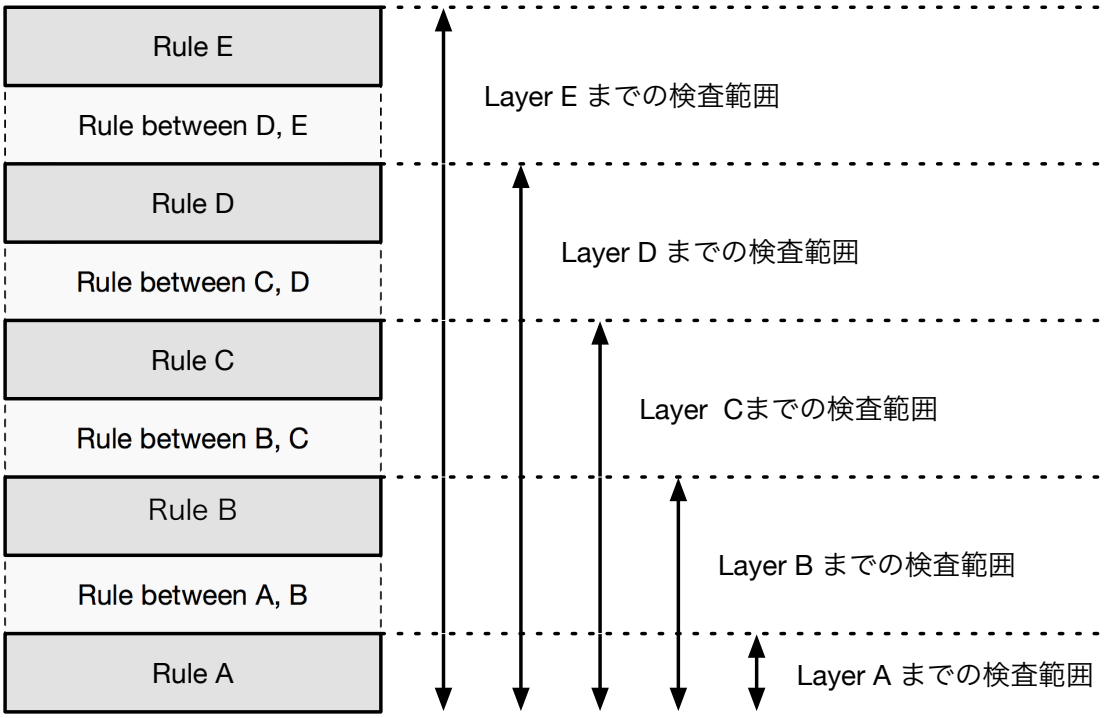

図 4.8: ルールと検査範囲

情報の間で満たすべきルールは、Protocol Channel を作るためのものである。

4.3.2 レイヤ間のルール

レイヤ間のルールは、Protocol Function と Protocol Channel が上位レイヤから下位レイヤに向けての依存関係が成立するための条件である。下位レイヤの要素に対して依存関係がない場合、Protocol Function と Protocol Channel は作られない。つまり、設定情報がそのレイヤ間でプロトコルの要件を満たしていないと言える。

第5章　システムの設計と実装

前章では、ネットワークにサービスや機能の追加が行われた場合、設定情報内でネットワークレイヤを横断したオペレーションが行われるという背景から、既存手法に代わるマルチレイヤネットワークに対応した設定情報の一貫性検査手法について述べた。

ネットワークの設定作業において、オペレータは一つの機器に対して設定を行うが、オペレータの目的はその機器を含むネットワークが正しく動作することである。しかし、設定作業において、物理的または論理的に隣接する機器の設定情報を全て正しくするのは難しい。よって、設定情報の矛盾をなくすため隣接する複数の機器の設定情報の一貫性を検査することで、機器へ設定を反映する前に、設定情報のレベルでネットワークが正しく動作することを可能にする。

提案手法に基づいて、物理的または論理的に隣接するネットワーク機器の設定情報の一貫性検査を行うには、まず設定情報をモデル化する。モデル化はネットワークレイヤ毎に設定情報を分離するモデル化を行うことで、マルチレイヤネットワークに対応する一貫性検査を可能にする。次に、設定情報をレイヤ毎に分離されたモデルを検査するには、上位レイヤは下位レイヤに機能が依存するので、下位レイヤから上位レイヤに向かって検査を行う。以下に提案手法に基づく一貫性検査に必要な事項を挙げる。

- ネットワークレイヤ毎に設定情報を分離するモデル化を行う
- マルチレイヤネットワークにおける検査は下位レイヤから上位レイヤの順に行う

本章では、これらのアイディアを踏まえたシステムの設計について述べる。

5.1　設定情報の一貫性検査に必要な要素

ネットワークの誤作動が発生する原因として、ネットワーク機器の設定作業において誤った設定情報をネットワーク機器に反映してしまうことが挙げられる。そのため、設定

情報の誤りを予防するには、ネットワーク機器に設定を投入する前に投入予定の設定項目を検査する必要がある。投入予定の設定の項目の検査は、反映済みの設定情報と照合することで行う。したがって、一貫性検査を行うに投入予定の設定情報に加えて、ネットワーク機器群に反映済みの設定情報も合わせて用いる必要がある。

5.2 システム全体の構成

提案する手法に基づくシステムの構成を図 5.1 に示す。一貫性検査エンジンはユーザからの問い合わせに対して検査を行い、その結果に基づいてフィードバックを返す。検査はレイヤ内のルールとレイヤ間のルールに従って行う。これより、ユーザは一貫性検査エンジンのフィードバックから ANSIBLE [15] や NETCONF/YANG [16] などのネットワーク設定ツールを用いて実際のネットワーク機器に対して設定作業を行うことで、設定作業におけるヒューマンエラーを予防する。

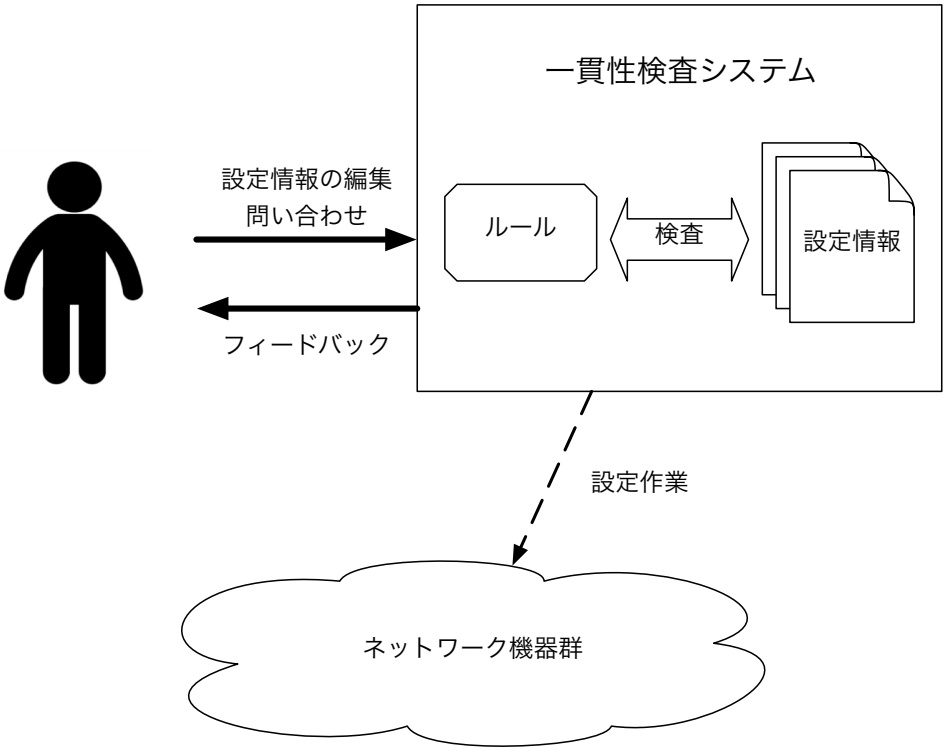

図 5.1: システムの構成

5.3　一貫性検査エンジン

一貫性検査エンジンは大きく分けて三つの機能を持つ。以下がその機能である。

- 設定情報のモデル化を行う機能
- 検査を行う機能
- 問い合わせに対してフィードバックを行う機能

まず、一貫性検査エンジンはネットワークオペレータの問い合わせに対して、与えられた設定項目が反映済みの設定情報と矛盾しないか検査を行う。次に、検査の結果をフィードバックとしてユーザに返す。

5.3.1　設定情報のモデル化

本システムの一貫性検査エンジンにおいて設定情報のモデル化を行う際の要件は、ネットワークレイヤ毎に記述が可能である事と、ネットワーク仮想化技術およびトンネル技術が一般的に記述可能である事が挙げられる。そこで、マルチレイヤにネットワークトポロジの情報を記述することが可能であり、またそれぞれのレイヤで個別に記述項目を拡張することが可能である 3.2.4 章で述べた YANG Data Model for Network Topology を参考にモデル化を行った。 図 5.2 は、設計したモデルの構成である。

設計したネットワークレイヤはフィジカルレイヤ、データリンクレイヤ、VLAN レイヤ、IP レイヤ、トランスポートレイヤ、アクセスコントロールレイヤである。これらの六つのレイヤ上に、ルータやスイッチ等の主要な設定項目がそれぞれ該当するよう設計を行った。

一貫性検査システムに与える設定情報は、予め反映済みの設定情報として与えられる場合があると、問い合わせとして与えられる場合がある。一貫性検査エンジンを動作させるには予め設定ファイルに信用済みの Protocol Function、Protocol Channel または、Protocol Relay があることを記述する。一般的なネットワーク機器の設定情報には物理的な結線についての情報は含まれていないため、少なくともフィジカルレイヤについては予めユーザが設定情報に記述しておく必要がある。また、フィジカルレイヤよりも上位のレイヤを記述することで、その記述については検査済みという扱いにすることが出来る。問

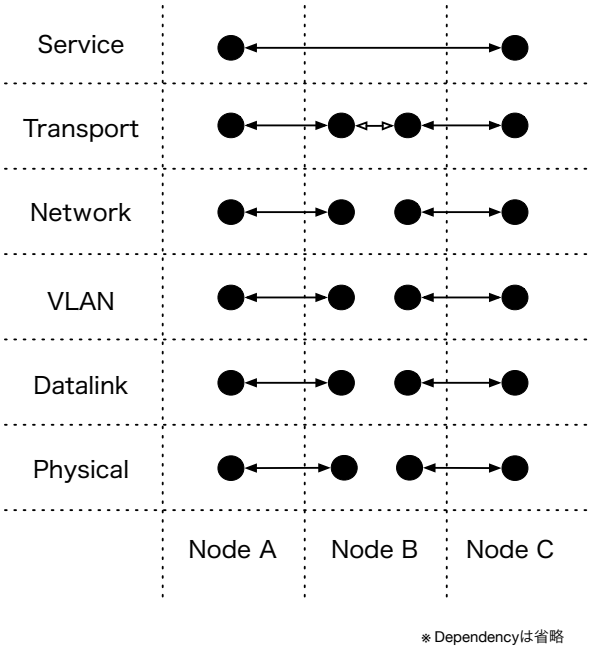

図 5.2: 設定情報のモデルの構成図

い合わせとして設定情報が与えられる場合は、与えられた設定情報はこれからネットワーク機器に投入する予定の未反映の物として扱う。したがって、問い合わせとして設定情報が与えられた場合、未検査の設定情報と検査済みの設定情報の間で照合し、検査を行う。

5.3.2 検査

一貫性検査エンジンは設定ファイルを読み込んだ後、ユーザの問い合わせに応じて検査を行う。前章で述べたように、検査は未検査の設定情報と検査済みの設定情報の間で矛盾がないかルールに乗っ取って照合することで行う。

5.3.3 問い合わせ

一貫性検査エンジンのフィードバック機能はユーザからの問い合わせに対してフィードバックを行う。以下に問い合わせの例を示す。

- レイヤ C 上で、Protocol Function 1 から Protocol Function 3 にデータの到達性はあるか

- レイヤ B 上で、Device 1 から Device 4 にデータの到達性はあるか

5.4 Prolog による実装

本章で説明した機能とシステムの要件を実現するために、一貫性検査エンジンの実装には Prolog を用いた。ネットワーク機器の設定情報を抽象度の高いまま扱う事が可能で、それらの関係性について表現することが可能だからである。

Prolog は 1970 年代に開発された論理プログラミング言語の一つである。「事実」とそれらの間の「ルール」に関する問題について解決するのに使われる。プログラム中で述語論理を記述し、任意の命題について推論を行うことが可能である。

そのため、ネットワーク機器の設定情報を抽象度の高いまま扱うことが可能であり、それらの関係性をプログラミングすることが可能である。

5.5 Prologによるレイヤ毎のルール

レイヤ内のルールとレイヤ間のルールは扱うプロトコルが異なるので、それぞれのレイヤで異なる。実際に実装したネットワークのルールについて説明する。

提案手法におけるモデルで、Protocol Function 及び Protocol Channel の生成のためのルールについて Prolog で実装したネットワークレイヤ毎に述べる。実装は 3 章で述べた Multi-Layered Network Description Model [11]、YANG Data Model for Network topology [12] を参考に行った。

Physical レイヤ

Physical レイヤで表現される Protocol Function と Protocol Channel を構成する要素は、ネットワーク機器の設定情報に一般的に含まれない。そのため、本レイヤではネットワーク機器群の物理的な構成をユーザ、もしくは LLDP 等の手法を用いて記述する必要がある。そのため、本提案手法に基づく検査システムにおいて Physical レイヤの情報はトラストアンカーとして扱われる。Physical レイヤにおいて Protocol Function と Protocol Channel は、以下のように表される。

```
physical_func(Func_id, -1, Int_name, Config_id).
physical_chan(Chan_id, -1, [Func_id_a, Func_id_b]).
```

Data Link レイヤ

Data Link レイヤにおける Protocol Funtion は、ネットワーク機器の設定情報中にラインプロトコル、全二重/半二重、MAC アドレス、MTU、スピードの項目が含まれている事、かつ同じインターフェース名を属性に持つ Physical レイヤ上の Protocol Function が存在することで生成される。また、Protocol Channel は、二つの Protocol Function の属性に対して照合を行い、適切である場合生成される。

```
is_datalink_func(Func_id, Ref_func_id, Int_name, Device_id) :-
    include_datalink_config(Int_name, Device_id),
    physical_function(Reference_func_id, _, Int_name, Device_id).

is_datalink_chan(Chan_id, Ref_Chan_id, [Func_id_a, Func_id_b]) :-
    datalink_func(Node_id_a, Ref_func_id_a, Int_name_a, Device_id_a),
    datalink_func(Node_id_b, Ref_func_id_b, Int_name_b, Device_id_b),
    physical_chan(Ref_chan_id, _, [Ref_func_id_a, Ref_func_id_b]),
    P = [Func_id_a, Func_id_b],
    is_different_device(P),
    is_same_duplex(P),
    is_different_macaddr(P),
    is_same_mtu(P),
    is_same_speed(P).
```

VLAN レイヤ

VLAN レイヤにおける Protocol Function は、ネットワーク機器の設定情報中に VLAN のタイプと VLAN ID が含まれている事、かつ下位レイヤである Data Link レイヤ上に同じインターフェース名を属性に持つ Protocol Function が存在することで生成される。VLAN についての設定項目が含まれない場合は、Data Link レイヤの Protocol Function

と同じように扱う。また、Protocol Channel は、二つの Protocol Function の属性に対して照合を行い、適切である場合生成される。

```
vlan_func(Func_id, Ref_Func_id, Int_name, Device_id, Type, Vlan_id) :-
    include_vlan_config(Int_name, Device_id, Type, Vlan_id),
    datalink_func(Func_id, Ref_func_id, Int_name, Device_id).

vlan_chan(Chan_id, Ref_Chan_id, [Func_id_a, Func_id_b]) :-
    vlan_func(Func_id_a, Ref_func_id_a, Int_name_a, Device_id_a, Type_a, Id_a),
    vlan_func(Func_id_b, Ref_func_id_b, Int_name_b, Device_id_b, Type_b, Id_b),
    datalink_chan(Ref_chan_id, _, [Ref_func_id_a, Ref_func_id_b]),
    P = [Func_id_a, Func_id_b],
    is_vlan_reachable(P).
```

Network レイヤ

Network レイヤにおける Protocol Funtion は、ネットワーク機器の設定情報中に IP アドレスの項目が含まれている事、かつ同じインターフェース名を属性に持つ VLAN レイヤ上の Protocol Function が存在することで生成される。また、Protocol Channel は、二つの Protocol Function の IP アドレスに対して照合を行い、適切である場合生成される。

```
ip_func(Func_id, Ref_func_id, Int_name, Device_id) :-
    include_ip_config(Int_name, Device_id),
    vlan_func(Func_id, _, Int_name, Device_id).

ip_chan(Chan_id, Ref_chan_id, [Func_id_a, Func_id_b], Duplex) :-
    ip_node(Func_id_a, Ref_func_id_a, Int_name_a, Device_id_a),
    ip_node(Func_id_b, Ref_func_id_b, Int_name_b, Device_id_b),
    P = [Func_id_a, Func_id_b],
    vlan_chan(Ref_chan_id, _, [Ref_func_id_a, Ref_func_id_b]),
    is_different_device(P),
    is_reachable_ipaddr(P).
```

Transport レイヤ

Transport レイヤにおける Protocol Funtion は、ポート番号とステートレスファイアウォールの項目を属性として持つ。ネットワーク機器の設定情報中にポート番号の項目が含まれている事、かつ同じインターフェース名を属性に持つ IP レイヤ上の Protocol Function が存在することで生成される。また、Protocol Channel は、二つの Protocol Function のポート番号とステートレスファイアウォールアドレスに対して照合を行い、適切である場合生成される。

```
is_transport_func(Funct_id, Ref_Func_id, Int_name, Device_id, Port, Acl) :-
    include_transport_config(Int_name, Device_id),
    ip_func(Ref_func_id, _, Int_name, Device_id).

is_transport_chan(Chan_id, Ref_Chan_id, [Func_id_a, Func_id_b], Duplex) :-
    transport_func(Func_id_a, Ref_Func_id_a, Int_name_a, Device_id_a, _Port_a ,
        Acl_a),
    transport_func(Func_id_b, Ref_Func_id_b, Int_name_b, Device_id_b, _Port_b ,
        Acl_b),
    ip_chan(Ref_chan_id, _, [Ref_func_id_a, Ref_func_id_b], Duplex).
```

第6章　評価実験と考察

前章では、提案手法に基づいたシステムの実装について述べた。加えて、Physical レイヤ、Data Link レイヤ、VLAN レイヤ、Network レイヤ、Transport レイヤのルールの構成について述べた。本章では、その提案手法に基づいて実装したシステムについて、提案手法の適用性、性能評価の面から評価を行う。また、それらの結果を踏まえて関連研究と提案手法の比較を行う。

6.1　提案システムの適用性

提案手法による検査が、どのようなネットワークに適用可能であるか利用者に示す必要がある。そこで、提案手法がどのようなネットワークトポロジの場合に有効であるか比較検討を行った。

提案手法は設定情報を入力として検査を行う。そのため、入力がどのような場合に有効であるか調査するため、複数の擬似的なネットワークトポロジを作成し、それらに含まれるネットワーク機器の設定情報を実装したシステムの入力として与えた。それぞれのトポロジで異なるレイヤに Protocol Relay が作成されることを想定してネットワークトポロジの作成を行った。

本実験においてネットワークトポロジ中のブリッジはデータリンク層で中継機能を持ち、ルータはネットワーク層で中継機能を持つ。また、サーバ A は HTTP クライアントであり、サーバ B は HTTP サーバである。各ネットワーク機器の設定情報をコード 6.1〜コード 6.4 に示す。作成したネットワークトポロジを図 6.1〜図 6.5 に示す。また、図 6.6〜図 6.10 はネットワーク機器の設定情報を基に生成される提案手法による論理モデルである。

ソースコード 6.1: サーバの設定情報

```
config(Config_id, line_protocol('eth0', 1)).
config(Config_id, duplex('eth0', auto)).
config(Config_id, mac_address('eth0', '14:10:9f:e2:db:61')).
config(Config_id, mtu('eth0', 1500)).
config(Config_id, speed('eth0', auto)).

config(Config_id, client_port('eth0')).

config(Config_id, ip_address('eth0', '192.168.10.10/24')).
```

ソースコード 6.2: ブリッジの設定情報

```
config(Config_id, line_protocol('ge-1/0/1', 1)).
config(Config_id, duplex('ge-1/0/1', auto)).
config(Config_id, mac_address('ge-1/0/1', '14:10:9f:e2:db:62')).
config(Config_id, mtu('ge-1/0/1', 1500)).
config(Config_id, speed('ge-1/0/1', auto)).

config(Config_id, bridge_port('ge-1/0/1')).

config(Config_id, line_protocol('ge-1/0/2', 1)).
config(Config_id, duplex('ge-1/0/2', auto)).
config(Config_id, mac_address('ge-1/0/2', '14:10:9f:e2:db:63')).
config(Config_id, mtu('ge-1/0/2', 1500)).
config(Config_id, speed('ge-1/0/2', auto)).

config(Config_di, bridge_port('ge-1/0/2')).
```

ソースコード 6.3: ルータの設定情報

```
config(Config_id, line_protocol('ge-1/0/1', 1)).
config(Config_id, duplex('ge-1/0/1', auto)).
config(Config_id, mac_address('ge-1/0/1', '14:10:9f:e2:db:62')).
config(Config_id, mtu('ge-1/0/1', 1500)).
config(Config_id, speed('ge-1/0/1', auto)).

config(Config_id, router_port('ge-1/0/1')).
config(Config_id, ip_address('ge-1/0/1', '192.168.10.1/24')).

config(Config_id, line_protocol('ge-1/0/2', 1)).
config(Config_id, duplex('ge-1/0/2', auto)).
config(Config_id, mac_address('ge-1/0/2', '14:10:9f:e2:db:63')).
config(Config_id, mtu('ge-1/0/2', 1500)).
config(Config_id, speed('ge-1/0/2', auto)).

config(Config_id, router_port('ge-1/0/2')).
config(Config_id, ip_address('ge-1/0/2', '10.0.0.1/16')).

config(Config_id, ip_route('192.168.10.0/24', 'ge-1/0/1')).
config(Config_id, ip_route('10.0.0.0/16', 'ge-1/0/2')).
```

ソースコード 6.4: ファイアウォールの設定情報

```
config(1, line_protocol('ge-1/0/1', 1)).
config(1, duplex('ge-1/0/1', auto)).
config(1, mac_address('ge-1/0/1', '14:10:9f:e2:db:62')).
config(1, mtu('ge-1/0/1', 1500)).
config(1, speed('ge-1/0/1', auto)).

config(1, router_port('ge-1/0/1')).
config(1, ip_address('ge-1/0/1', '192.168.10.1/24')).

config(1, line_protocol('ge-1/0/2', 1)).
config(1, duplex('ge-1/0/2', auto)).
config(1, mac_address('ge-1/0/2', '14:10:9f:e2:db:63')).
config(1, mtu('ge-1/0/2', 1500)).
config(1, speed('ge-1/0/2', auto)).

config(1, router_port('ge-1/0/2')).
config(1, ip_address('ge-1/0/2', '10.0.0.1/16')).

config(1, ip_route('192.168.10.0/24', 'ge-1/0/1')).
config(1, ip_route('10.0.0.0/16', 'ge-1/0/2')).

config(1, ip_access_control(10, 'permit', '192.168.10.0/24', 'any', 80)).
```

サンプルトポロジ A は、二つのサーバの間にブリッジが配置された構成である。ブリッジ A は、Data Link レイヤにおいて Protocol Relay を持ち、VLAN レイヤより上位のレイヤでは、Protocol Function を持たない。

サンプルトポロジ B は、二つのサーバの間にルータが配置された構成である。ルータ A は、Network レイヤにおいて Protocol Relay を持つ。これにより、IP ネットワークの静的ルーティングがモデル化されている。また、VLAN レイヤより上位のレイヤでは Protocol Function を持たない。

サンプルトポロジ C は、二つのサーバの間にブリッジとルータが配置された構成である。ブリッジ A は、Data Link レイヤにおいて Protocol Relay を持つ。また、ルータ A は、Network レイヤにおいて Protocol Relay を持つ。異なるレイヤで Protocol Relay が作成されてもモデル化が可能である。

サンプルトポロジ D は、二つのサーバの間にブリッジとルータが配置され VLAN を用いる構成である。物理構成はサンプルトポロジ C と同じであるが、論理モデルは異なる構造を持つ。ルータ A は、VLAN レイヤにおいて Protocol Relay を持ち、付加属性として VLAN ID が付与されている。VLAN ID 毎に Protocol Function が生成されるため、頻繁に使われるポート VLAN や、タグ VLAN をモデル化することが可能である。

サンプルトポロジ E は、二つのサーバの間にルータとファイアウォールが配置された構成である。Firewall A はステートレスなアクセスコントロールの機能を持ち、Transport レイヤ上に Protocol Relay が作成される。その Protocol Relay は、 付加属性としてアクセスコントロールリストの情報を持つ。利用者からの問い合わせの際に、送信元及び送信先の IP アドレスとポート番号を参照し照合を行う。

一方で、提案する手法でモデル化を行うことが出来ないネットワークトポロジは、OSPF や BGP 等の動的ルーティングプロトコル、ステートフルファイアウォール等がある。その理由として、設定情報が動的に書き換わることが挙げられる。本提案手法はある時間の設定情報全体の検査を行うため、時間を要する。そのため、短い連続した時間の設定情報の検査を行うことは困難である。

6.2 性能評価

設定情報をネットワーク機器に反映する前に検査を行うにあたって、検査時間は可能な限り短いほうが良い。そこで、提案手法に基づいて実装したシステムの実行時間を測定し

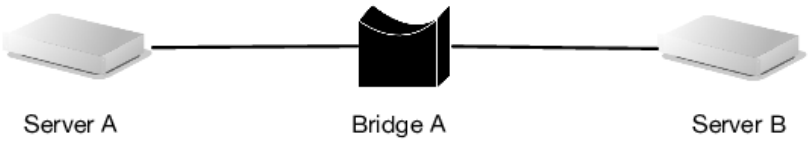

図 6.1: サンプルトポロジ A の物理構成

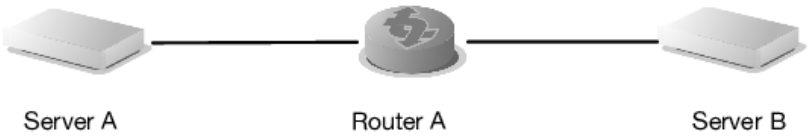

図 6.2: サンプルトポロジ B の物理構成

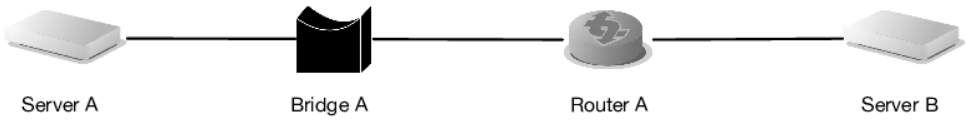

図 6.3: サンプルトポロジ C の物理構成

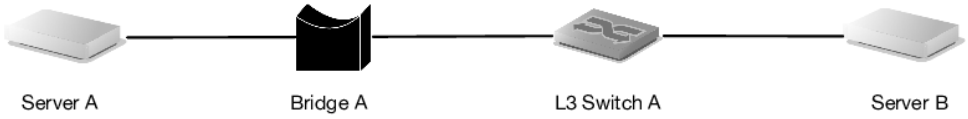

図 6.4: サンプルトポロジ D の物理構成

図 6.5: サンプルトポロジ E の物理構成

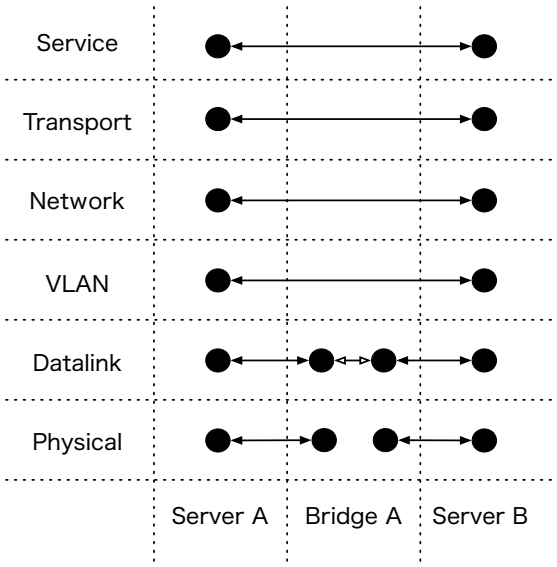

図 6.6: サンプルトポロジ A の論理モデル

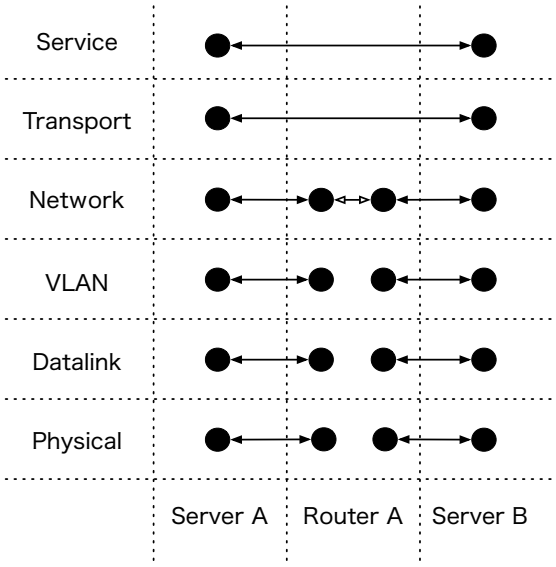

図 6.7: サンプルトポロジ B の論理モデル

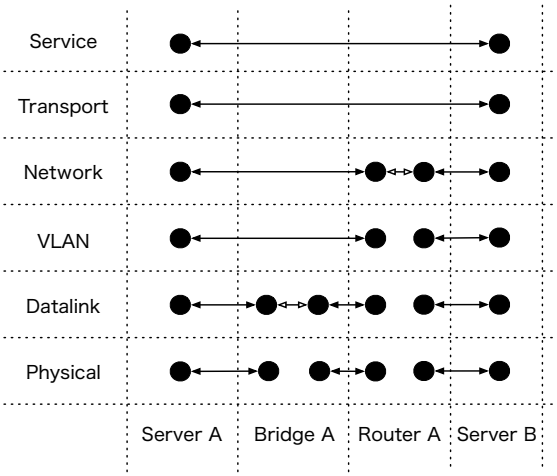

図 6.8: サンプルトポロジ C の論理モデル

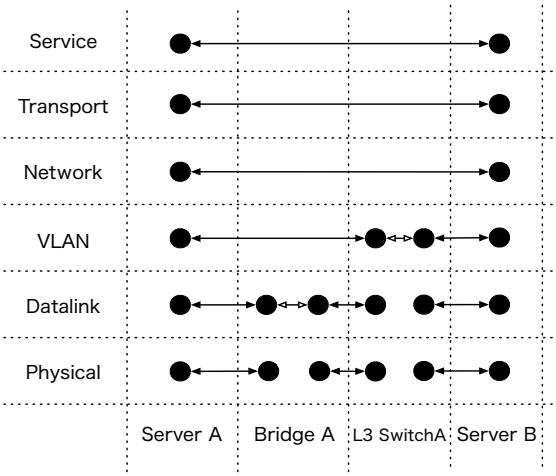

図 6.9: サンプルトポロジ D の論理モデル

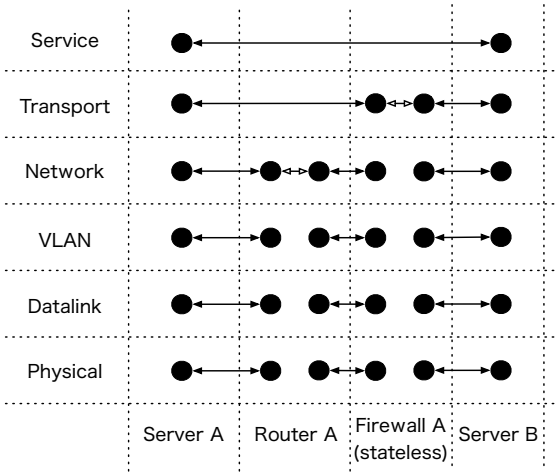

図 6.10: サンプルトポロジ E の論理モデル

た。実験を行ったコンピュータは以下の構成である。

- OS: MAC OS X 10.12.3
- CPU: 2.7GHz Intel Core i7 (4 core)
- Memory: 16GB 1600 MHz DDR3
- Processing system: SWI-Prolog

実装したシステムは起動時に機器の設定情報を読み込み、論理モデルを構築する。よって、読み込む設定情報に依存して論理モデルを構築する時間が変わることが予想される。したがって、異なるレイヤでモデルを構築する設定情報を与えることで、起動時間にどのような影響を与えるか計測を行った。

本実験を行うため、データリンクレイヤで中継機能を持つブリッジと、ネットワークレイヤで中継機能を持つルータの設定情報を用意した。また、HTTP クライアントと HTTP サーバのモデルを表現する設定情報を擬似的に用意した。これらの設定情報は、ブリッジまたはサーバが HTTP クライアントと、HTTP サーバの間を数珠状及び、一直線に繋がれたトポロジを想定して作成した。測定は、全てのネットワーク機器がルータである場合、全ての機器がブリッジである場合、ネットワーク機器の半分がルータであり、一方の半分がブリッジでそれらが交互に接続された場合の三パターンについて行う。図 6.11 は、測定結果を示すグラフである。

提案手法はマルチレイヤに対応するため、下位レイヤからモデル化を行う。そのため、初期化にかかる時間がレイヤ数の多い、全ての機器がルータである場合、ルータとブリッジで半分ずつの場合、全ての機器がブリッジの順になる。また、それぞの場合で処理時間が、機器の数を増加させる毎に指数関数的に増加する。

筆者の主観であるが、処理時間を考慮するとキャリアや ISP ような大規模なネットワークに適用することは現実的ではない。しかし、それらよりも小さい中小規模のネットワークには、多少の待ち時間が生じるが、運用に組み込むことが可能であると考えられる。また、設定作業者がネットワークに精通していない場合が多いホームネットワークにおいても、設定作業者の補助として提案手法を基に実装したシステムが働くことが考えられる。

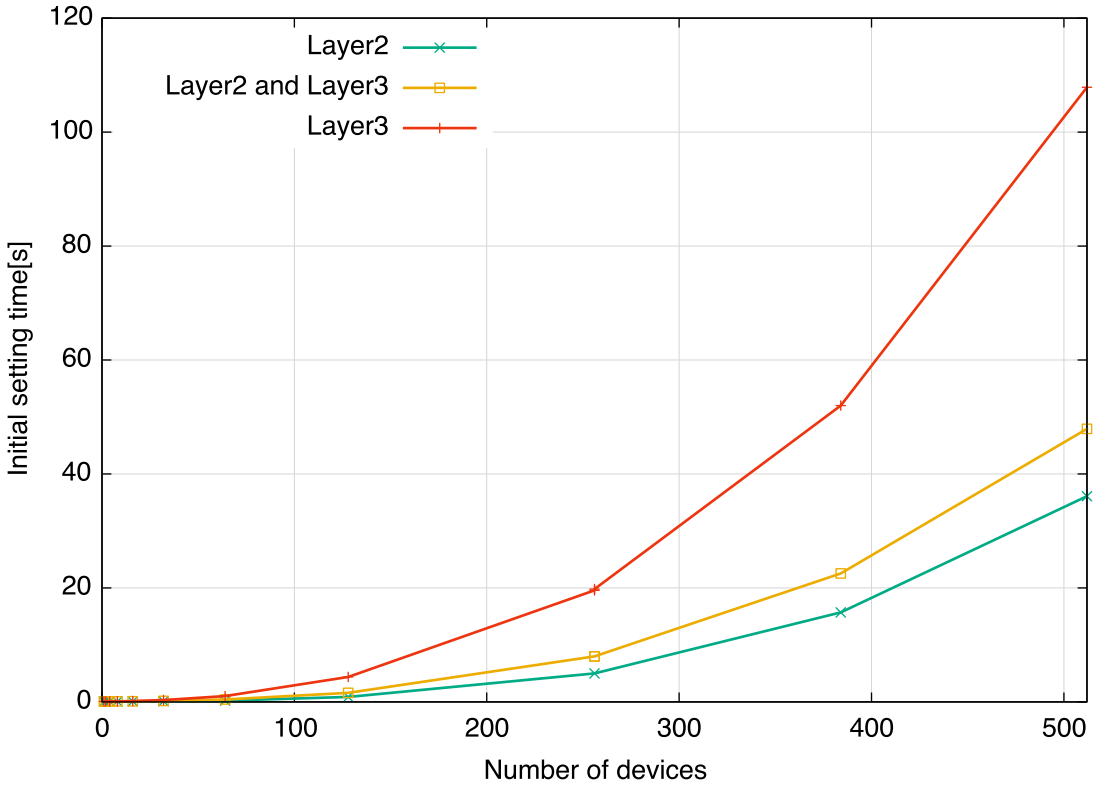

図 6.11: システムの初期化に要した時間

6.3 関連研究との比較

本節では、提案手法と 3.2 節で挙げた NDL、Multi-Layered Network Description Model、YANG Data Model for Network Topology の記述性について考察を行う。

Multi-Layered Network Description Language は、リンクとノードの集合を組み合わせたグラフモデルを用いることによって表現が困難であったスイッチやルータ、ファイアウォール等のネットワーク機器における機能の特徴を失うことなく表現可能なネットワークモデルである。また、ノードの仮想化とリンクの仮想化が表現可能であるため、様々なプロトコルを表現可能であると言える。

YANG Data Model for Network Topology は、YANG をネットワークトポロジに応用したネットワーク記述言語である。基本となるトポロジレイヤが定義されており、各ネットワークレイヤで属性を付与していくことでマルチレイヤに対応したネットワークを記述することを可能にしている。YANG が NETCONF 上で扱うために設計されているので、設定情報を組み合わせることでトポロジを記述することが可能である。

提案手法で扱うネットワークモデルは、Multi-Layered Network Description Language と、Multi-Layered Network Description Language を参考に設計を行った。

表 6.1 は提案手法と関連研究との比較項目である。比較項目を以下に示す。

表 6.1: 提案手法と既存手法の比較

手法	提案手法	Multi-Layerd NDL	YANG Data Model
ノード仮想化の表現	○	○	○
リンク仮想化の表現	○	○	△
物理/論理ネットワークの分離	○	○	○
機器設定ツールとの協調性	△	×	○

第7章　おわりに

本章では、本論文におけるまとめと今後の課題と展望について述べる。

7.1　まとめ

本論文では、マルチレイヤネットワークに対応した設定情報の検査手法について提案した。様々なネットワーク仮想化技術が開発され、ネットワーク機器の設定情報が複雑化する傾向にあるため、ヒューマンエラーによって誤った設定が反映されてしまうことを防止する事を目的とした。提案手法では、対象とするネットワークに属する機器の設定情報をネットワークレイヤ毎に記述し、一貫性検査を行う。ネットワークレイヤ毎に記述された設定情報を検査することで、ネットワークレイヤを跨った設定情報であっても一貫性検査を行うことができると考える。この提案手法を基に、実際にシステムの設計と実装を行った。また実装したシステムにおいて、スケーラビリティ及び関連研究との比較について評価を行った。

7.2　今後の課題と展望

本論文では、ネットワークの構築と運用におけるネットワークオペレータの誤った設定作業を未然に防ぐための手法について議論を行い、その手法に基づくシステムの提案、実装を行った。実装したシステムは、ネットワーク機器の設定情報を筆者が作成したルールに基づいて検査を行う。しかし、作成したルールに不備があれば検査に信頼性はなくなる。そのため、機械学習を用いて過去の様々な正常稼働時の設定情報からルールを作成することで、人の手を出来るだけ介さないルールベースの検査が考えられる。

また、本論文では設定情報を検査することでネットワーク障害を予防することを目的としていたが、過去の障害情報からルールを作成し検査を行うことも有効であると考えられる。

謝辞

本研究を行うにあたり、多くの方から多大なご助言やご助力を頂きました。それらの方々のご協力がなければ、本研究は成り立ちませんでした。ここに深く感謝し、心から厚くお礼申し上げます。

本研究を進めるにあたり、指導教員である篠田陽一教授には様々な助言、適切な御指導を賜りました。心から深く感謝します。また、助言を頂いた主テーマ審査員である丹康雄教授、副テーマ指導教員である寺内多智弘教授に感謝致します。

本研究室の知念賢一特任准教授、宇多仁助教には、研究に関して活発な議論や多大なご指導を賜りました。心から感謝致します。

情報通信機構北陸リサーチセンターの高野祐輝氏、井上朋哉氏、三浦良介氏には研究に関して様々な助言、ご指導を賜りました。心から感謝致します。

本研究室の博士後期課程の Muhammad Imran tariq 氏、明石邦夫氏、太田悟史氏、阿部博氏には、研究に関して活発な議論、ご指導を賜りました。心から感謝致します。

本研究室修了生の岩橋紘司氏、園田真人氏には、研究に関して活発な議論、ご指導を賜りました。また、研究生活を送る上で様々なご助力を頂きました。心から感謝致します。

本研究室の博士前期課程の可児友邦氏、三木晶司氏、石原俊氏、大川昌寛氏、押川侑樹氏、橋本光世氏、村上正樹氏、阿波史和氏、砂川真範氏には活発な議論や、研究生活を送る上で様々なご助力を頂きました。心から感謝致します。

最後に研究や生活で支えてくれた家族へ心から感謝致します。

参考文献

[1] 大野 健彦, 中谷 桃子, 高山 千尋, 草野 孔希, ネットワーク保守におけるヒューマンエラー削減の取り組み

[2] Ari Fogel, Luis Pedorosa, Meg Walraed-Sullivan, Proactive Network Configuration Validation with Batfish

[3] Sanjay Narain, Rajesh Talpade,Gary Levin, Network Configuration Validation

[4] Haohui Mai, Ahmed Khurshid Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, Samuel T. King, Debugging the Data Plane with Anteater

[5] Jeroen van der Ham, Mattijs Ghijsen, Paola Grosso, Cees de Laat, Trends in Computer Network Modeling Towards the Future Internet

[6] ping, http://linuxjm.osdn.jp/html/netkit/man8/ping.8.html

[7] traceroute, https://linux.die.net/man/8/traceroute

[8] tcpdump, http://www.tcpdump.org/tcpdump_man.html

[9] Network Description Language, https://ivi.fnwi.uva.nl/sne/ndl/

[10] Mattijs Ghijsen, Jeroen van der Ham, Paola Grosso, Cosmin, Dumitru, Hao Zhu, Zhiming Zhao and Cees de Laat, A Semantic-Web Approach for Modeling Computing Infrastructures

[11] 金岡 晃, 原田 敏樹, 加藤 雅彦, 勝野 恭治, 岡本 栄司, 安全なネットワークシ ステム設計のためのマルチレイヤネットワークモデルの提案と応用

[12] YANG Data Model for Network Topology, https://datatracker.ietf.org/doc/draft-ietf-i2rs-yang-network-topo

[13] RFC6020, YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)

[14] 山崎 智史, 八鍬 豊, 登内 敏夫, 形式手法における SDN/OpenFlow 高信頼技術の研究動向

[15] ANSIBLE, https://www.ansible.com

[16] NETCONF, RFC6020, RFC6241

[17] SWI-Prolog, http://www.swi-prolog.org