博士論文

大規模ネットワーク管理のための HTTP/SNMP リバースプロキシと 管理トラヒックの公平性に関する研究

宋　泰永

内容梗概

インターネットの普及と伴い，大規模化し複雑化した分散型のネットワーク管理を中央集中型の SDN (Software-Defined Networking) [3] や NFV(Network Functions Virtualization) [4] により合理化することをネットワーク事業者が検討するようになった．ネットワーク管理者が直接ネットワーク機器にターミナル接続しコマンドを入力して管理する時代から，中央のコントローラ上の管理アプリケーション (以下では Network Management System あるいは NMS と略称する）がネットワーク内のトラヒックフローの変更や不具合を自動的に探知し，ネットワークリソースとして仮想化されたサーバや firewall を動的に配置して問題解決するといったネットワーク管理の自動化が進んでいる．この結果，中央のコントローラが数万台以上の機器を管理するようになっている．しかし，このような新しい管理に対応していない機器は依然として数多く使われていて，その全てを一気に新しい機器へ置換することは難しい．本研究では既存のネットワーク機器の管理手法である SNMP を大規模かつ中央集中化した SDN や NFV のような新しいネットワーク管理技術と共存させるため，新しいキャッシュ管理手法に基づくプロキシを提案する．

Hamid 等 [15] が提案したプロキシ Tambourine は，管理者がインターネットを介して大規模なネットワーク管理する場合に HTTP とネットワーク管理プロトコル間の相互変換を行うプロキシであり，HTTP で送る管理用のリクエストを SNMP のリクエストに変換しネットワーク機器 (以下 SNMP 機器と呼ぶ）に転送していた．その際，HTTP リクエストに対する SNMP 機器からの管理情報を OID(Object Identifier) ごとにキャッシュし，頻繁な管理リクエストを処理することに起因する SNMP 機器の負荷を下げ，かつ各情報の更新周期に合致した最適なキャッシュの TTL(Time-to-live) を自動学習することに成功していた．しかし，HTTP リクエストを頻繁に値が更新する OID に集中させることにより，SNMP 機器を DDoS (Distributed Denial of Service) 攻撃できる脆弱性があった．

このため，本論文では，下記の 4 要求条件に基づく新たなキャッシュ制御手法を考案している．条件 1 として，Tambourine は，SNMP 機器の CPU 使用率の増加分が一定値 (TH) 以下になるよう管理トラヒックを制御する．このためには，管理トラヒックによる SNMP 機器の CPU 使用率を計算しキャッシュの TTL を設定する必要がある．条件 2 として，同一リクエスト間隔の NMS 間で情報取得の公平性を実現しなければならない．NMS からの HTTP リクエストに対し Tambourine が得た SNMP レスポンスの数の平均値でその標準偏差を割って相対化した変動係数 (R) を公平性の評価尺度とした．変動係数 (R) が小さければ，そのグループ内の NMS は公平であると見なすことができる．条件 3 として，異なるリクエスト間隔の NMS グループの間でも情報取得の公平性を実現する．前述のように，変動係数 (R) は NMS からの HTTP リクエスト総数に対し Tambourine が得た SNMP レスポンスの数の標準偏差を平均値で相対化したものであるから，変動係数 (R) が同じであれば，リクエスト間隔に依らず，NMS グループ間の相対的なばらつきが同じであることを意味する．このための評価尺度として上記の変動係数 (R) を用いることができる．最後に，条件 4 として，頻繁な問い合わせる NMS に対しては，上記の条件を満足する範囲内で，最大限応じなければならないとした．そのための評価尺度として，NMS から Tambourine への HTTP リクエストがキャッシュにヒットした場合，SNMP 機器から実際に得た OID 情報はそれ以前に取得した値

であるため，取得ギャップが生ずることに注目し，取得遅れの平均値を鮮度 (Fr) として定義し評価した．

提示手法は，Tambourine から SNMP 機器へ送る SNMP リクエスト頻度と SNMP 機器の CPU 使用率間の関係を事前実験により求め，SNMP 機器への SNMP 頻度 (リクエスト/秒) から SNMP 機器の CPU 使用率を推定することを基本とする．このための前処理として，PC から Alaxala3640s と Catalyst 2950 に SNMP リクエストを任意の一定間隔で送り，各機器の過去 1 分間の CPU 使用率を測定し近似式を求めた．

条件 1 に対しては，NMS からの問い合わせに対し，SNMP 機器の CPU 利用率が高い輻輳状態ではキャッシュから値を返し，他の場合は SNMP 機器から情報を取得して返すが，その際，TCP Reno に類似したキャッシュの更新間隔 (TTL) 調整手法を提案した．ここで，CPU 使用率増加が予め指定した閾値 (TH) 以上になることを輻輳と定義している．また，TCP の RED に類似したリクエストの制御手法を提案し，リクエスト送信頻度の高い NMS ほど，CPU 使用率の計算の際，TH 以上の値を得る確率を上げ，これにより，NMS 間で公平に SNMP 機器からレスポンスを取得でき，条件 2 と 3 を満足させることができることを示した．最後に，条件 4 の評価尺度として，NMS の問い合わせ間隔と取得データの遅延との関係を Fr で評価した．

評価は，ns-2 で構成したシミュレーションで実施した．条件 1 は Reno に類似したキャッシュの制御で，SNMP 機器の CPU 使用率が TH として設定した 20%近傍に収束することを示した．条件 2 は，HTTP リクエスト間隔が同じ NMS 間では，RED でキャッシュ制御手法行うことにより，公平性に SNMP 機器から SNMP レスポンスを得ることができることを示した．また，条件 3 は NMS からの HTTP リクエスト間隔より大きい RTO を選択し RED でキャッシュの制御をすることで，異なるリクエスト間隔の NMS グループ間でも公平性が改善されることを示した．最後の条件 4 に対しては，提案手法により NMS からの HTTP リクエスト間隔と Fr が比例することが分かった．これは短いモニタリングほど短い遅延時間で SNMP 機器からレスポンスを得ていることを意味し，条件 4 を満足していることを意味する．以上の評価実験により，提案手法は条件全てを満足する公平なキャッシュの制御を実現できることを示すことができた．

インターネットのトラヒック制御は，これまでアプリケーション間のデータが流れるデータプレーンに対して主として考えられてきたが，本論文ではネットワーク機器を制御するコントロールプレーンにおけるトラヒック制御を扱っている．また，SDN や NFV の場合，コントロールプレーンはアプリケーションと中央コントローラ間の Northbound API と，中央コントローラとネットワーク機器間の Southbound API に分けて扱うが，本論文の対象は，これまであまり検討が進められていない Northbound API に関係している．

目 次

図 目 次

表 目 次

第1章

序論

1.1　本研究の背景

IP ネットワークの管理は，当初企業内ネットワーク程度の小規模ネットワークを対象に設計され，SNMP [1] や CLI(Command Line Interface) を使って運用されてきた．インターネットの拡大に伴い，スケーラビリティの課題が検討され SNMPv3 [1] や NETCONF [2] が開発されてきたが，未だに人手に頼ったネットワーク管理手法が主流である．一方，近年普及してきたクラウドコンピューティングは，何千台規模の通信機器をデータセンタ内で一括管理し，短時間の管理者の需要に合わせて柔軟に計算リソースや通信リソースの割り当てを行うようになってきている．また，SDN (Software-Defined Networking) [3] や NFV(Network Functions Virtualization) [4] のようなネットワーク仮想化 [5] [6] に代表される大規模かつ中央集中型で準リアルタイムで構成変更が可能な技術の普及により，各ネットワーク機器の通信リソースをモニタする頻度と粒度は飛躍的に高まると考えられる．モニタ主体が人間ではなく運用管理システムとなること，新興国を中心にネットワークの運用を機器メーカにアウトソースする動きも活発化していることから [7] ，大規模化したネットワークの管理プレーンを流れるトラヒックは，質量ともに拡大し，予測しにくくなる．

1.1.1　ヘテロジニアスなネットワーク環境

インターネットの発達過程で，Cisco や Juniper Networks など大手企業が各自開発を進めたため，ネットワーク管理者は各会社ごとに異なるネットワーク機器のコマンドやレスポンスを理解する必要があった．そのため，異なるメーカの機器で構築されたネットワークを管理するために管理者に大きな負担がかかることになった．しかし，インターネットの普及は様々なニーズを生み出し，多様な顧客のニーズを満足させるネットワーク機器が多くのメーカから発売されている．そのため，複数のメーカからの機器で構築されたヘテロジニアスなネットワークを管理するため，メーカやデバイスに依存しない統一された型式による管理プロトコルが必要になった．

ネットワーク機器へのコマンドやレスポンスの型式を標準化する過程で，IETF は 1990 年に Simple Network Management Protocol(SNMP) [1] を開発した．SNMP とは管理者端末と被管理機器の間のやり取りを行うメッセージを定義したプロトコルであり，異なるメーカのネットワーク機器でも SNMP を実装していれば，SNMP を実装した管理者端末から一括で管理を行うことが可能になる．しかし，SNMP は主にモニタリング目的で利用され，各機器の設定は未だ管理者が直接端末に SSH [8] や telnet を利用して接続し，CLI(Command Line Interfaces) を使い設定する方法が一般的であった．

2003 年，IETF は NETCONF working group を作り，XML ベースで設定可能なネットワーク管理プロトコルを作った．それが NETCONF [2] である．NETCONF は，ネットワーク機器のコンフィギュレーションに関して，管理対象の状態と設定を明確に分離し，XML スキーマ [9] や YANG [10] を基にコマンドとレスポンスを定義した．最近は Openflow [11] switch の設定プロトコルとして NETCONF が採用され注目を集めている．

1.1.2　迅速なネットワーク管理の必要

もはやインターネットやネットワーク技術無しではビジネスは不可能になっている．アメリカの ebay や Amazon，中国の AliExpress のようにインターネットを通じて顧客からの注文やコミュニケーションを行う企業だけでなく，一般企業でもインターネットや社内ネットワークを利用し通常業務を行っているため，ネットワーク障害に素早く対応することは競争力と直結している．そのた

め，人力に依存するネットワーク管理方法から自動化による機敏なネットワーク管理へのニーズも高まっている．

前述した NETCONF の場合，人間だけでなくパソコンでも理解しやすいコマンドとレスポンスを XML スキーマ [9] や YANG [10] を基に定義したため，Java などを利用して作成したプログラムでネットワーク機器のモニタリングや設定を自動的に行うことを可能にしている．また，SDN [3] と NFV [4] の普及により，ネットワーク管理機能を中央のコントローラへ集中するトレンドにあり，これもネットワーク管理の自動化を促すもう一つの要因になっている．

1.1.3　分散管理から中央集中管理への回帰

図 1.1 で示すように，ネットワーク管理は，技術の発達と共に制御形態が変化している．インターネットがなかった時期，公衆電話網の管理は中央の電話交換機で集中制御していたが耐災害性は十分ではなかった．これを解決するため，自律分散処理で作られたインターネットでは，管理機能がネットワーク機器へ分散され一部のネットワークに不具合が発生しても，残りのルートを使うことで戦争や災害に強い回復力を保つことができた．しかし，運用コストは人力に頼るところが増え，逆に上昇している．このため，SDN と NFV により，ルータやスイッチだけでなく，firewall や load balancer などのサーバも仮想化し，また中央のコントローラ上のプログラムで集中管理し，プログラム制御により，ダイナミックに変わるユーザのニーズに早く対応することが求められるようになった．例えば，特定の地域に急に人が集まり，サーバへのアクセス数が急上昇した場合，サービス・プロバイダは，その地域から近いデータセンターへサーバの仮想マシンを移行・起動させ，ユーザからのアクセスを分散することで，サービス品質を上げることができる．また一連の処理を自動的に行うため，ネットワーク管理の運営コストも削減できる．

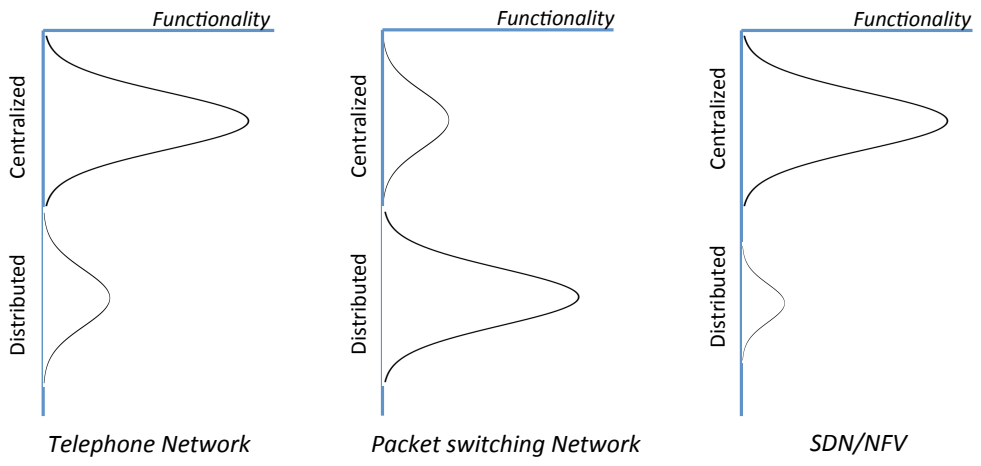


図 1.1: ネットワーク管理機能は集中管理から分散管理へ移行したが SDN/NFV で集中管理へ回帰 [12]

表 1.1: 大規模ネットワークと中央集中管理

	Previous	Current	Primary Factors
Types of Network	Homogeneous	Heterogeneous	SNMP/NETCONF
Controller's Location	Distributed	Central	SDN/NFV
Configuration Methods	Manually	Automatic	SDN/NFV
Number of Devices	Small	Large	SDN/NFV and Network Outsourcing
Access Scale	Local	Global	Network Outsourcing

1.2　大規模ネットワーク管理

ネットワークの規模は，6.2 で議論するように仮想化技術と共に数千数万台を超える機器から構成されるようになっている．また，表 1.1 に示したように，ヘテロジニアスなネットワークを中央集中型で管理を自動化し，数万台の機器をグローバル規模で管理するようになった．既存の小規模のネットワーク管理の場合，不具合が発生すると，管理者は問題が起こった機器のところへ移動し，手持ちの端末を直接機器に繋いで問題の把握や解決をする必要があった．このため，管理すべき機器の数が上昇すると，ネットワーク管理や運営コストも上昇してしまった．そのため，図 1.2 で示すように一般企業だけでなく新興国の ISP もネットワーク管理と運営を外部の専門家に委託運営してもらうネットワーク管理のアウトソーシングビジネスが成長している [7] [13] [14] .

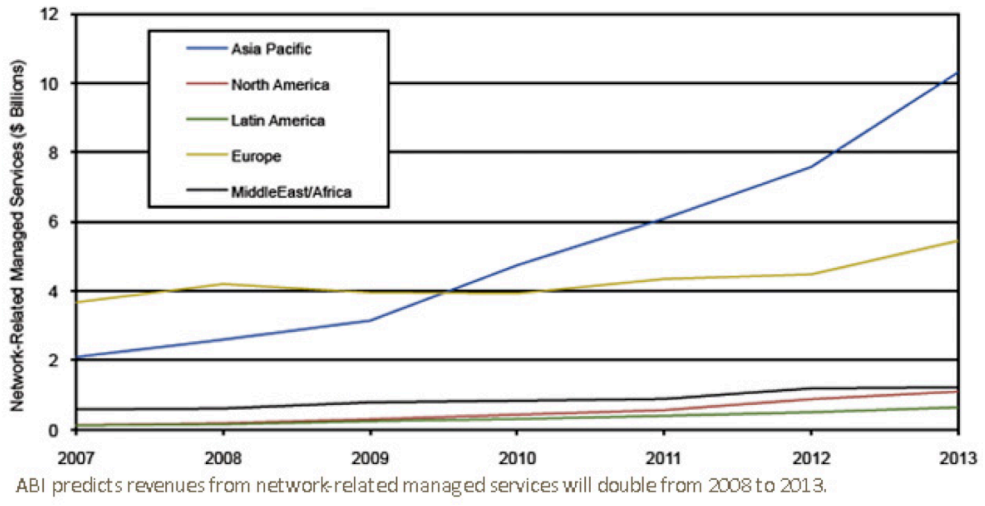


図 1.2: Managed Network Service 市場の成長 [13]

1.2.1　Tambourine:プロキシを利用したネットワーク管理

グローバルスケールで大規模ネットワーク管理する場合，プロキシを利用して HTTP とネットワーク管理プロトコルとの変換を行う手法が一般的である．Hamid 等 [15] は，ネットワーク管理のアプリケーションインタフェースを HTTP で統一し，各管理プロトコルへの翻訳を RESTful アーキテクチャ[16] を基にしたネットワーク管理用アクティブプロキシ Tambourine で実行することにより，大規模な SNMP 機器のモニタリングが可能になることを示した．

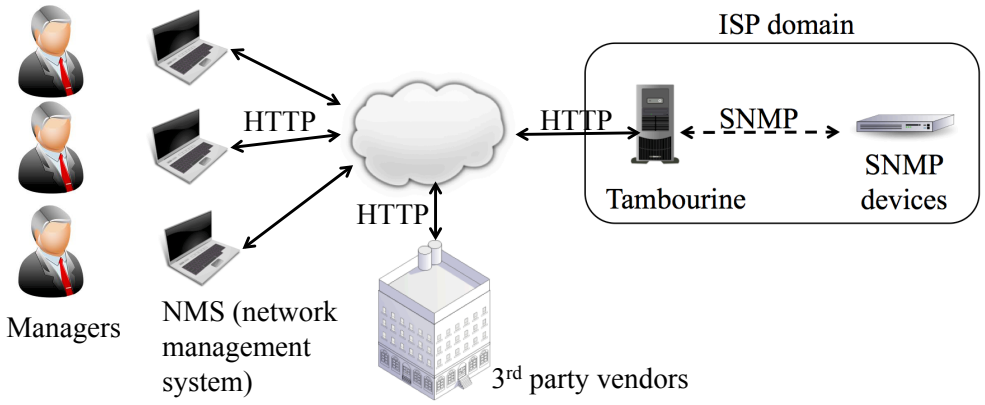


図 1.3: Tambourine の仕組み [15]

Tambourine のプロキシとしての仕組みは HTTP Get request を SNMP GetRequest に変換してルータ/スイッチなど SNMP 機器上の SNMP エージェント（以下 Agent）に送り，これに対するレスポンスを XML に変換して，ネットワーク管理者に返すことである．図 1.3 で示すようにネットワーク管理者は NMS(Network Management System) を利用しインターネット経由で任意の SNMP 機器へアクセスできる．Tambourine の主な利点は，管理者からのリクエストに対する SNMP 機器からの管理情報を OID(Object Identifier) ごとにキャッシュすることで，頻繁な管理リクエストによる SNMP 機器への負荷を下げ，各情報の更新周期に合致した最適なキャッシュの TTL(Time-to-live) を自動設定できることである．しかし，頻繁に値が更新される OID に DDoS (Distributed Denial of Service) 攻撃等でアクセスが集中すると TTL が短くなりすぎ，SNMP 機器の CPU 使用率が過大になるという脆弱性があった．

1.2.2　OpenFlow による中央集中型ネットワーク管理

Openflow [11] で代表される SDN(Software-Defined Network) [3] の登場により, ネットワーク管理の自動化が加速化されている．Openflow では, これまでネットワーク機器上に分散されていたネットワーク管理機能を中央サーバへ集中させることでネットワーク管理コストの大幅な削減や新たなプロトコルの導入が容易になるメリットがある．この場合,Openflow 対応機器は中央のサーバ 1 台で管理するため，多数の管理者がアクセスすることによる負荷は，各 Openflow 対応機器よりも Openflow コントローラ自体に大きく課されるものと考えられる．一般ユーザによる VPN サービス利用までサービスを拡大すると，既存とは比べられないほど大量の制御トラヒックが Openflow コントローラへ集中する可能性がある．

1.3　目標と制約条件

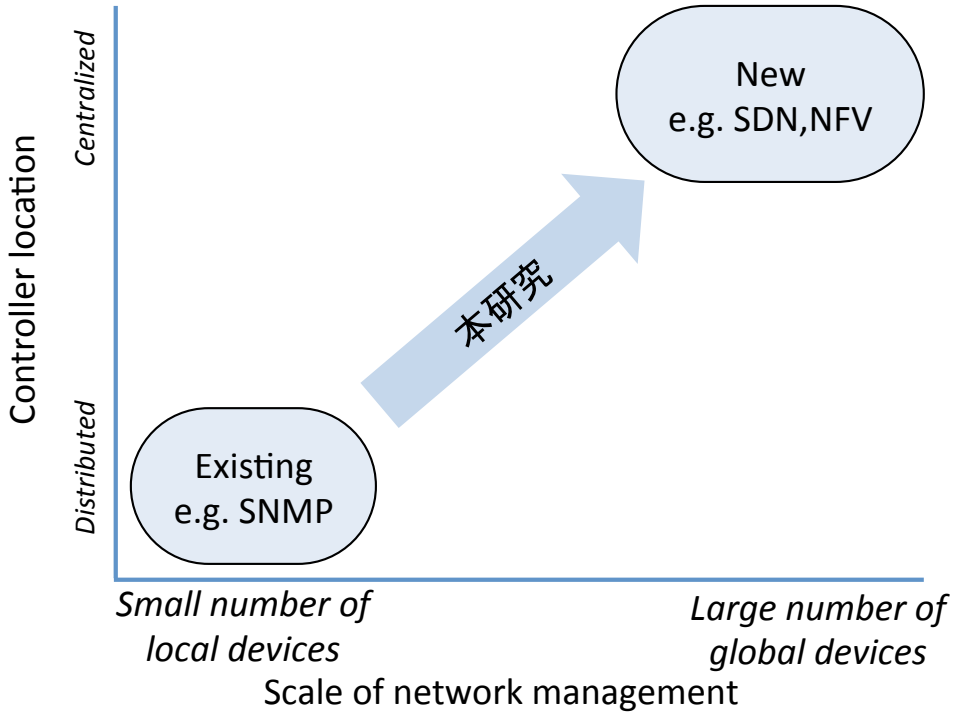


図 1.4: 大規模かつ中央集中型管理へ移行するネットワーク管理トレンド

インターネットのトラヒック制御は，これまでアプリケーション間のデータが流れるデータプレーンに対して主として考えられてきたが，本論文ではネットワーク機器を制御するコントロールプレーンにおけるトラヒック制御を扱っている．また，SDN や NFD の場合，コントロールプレーンはアプリケーションと中央コントローラ間の Northbound API と，中央コントローラとネットワーク機器間の Southbound API に分けて扱うが，本論文の対象は，これまであまり検討が進められていない Northbound API に関係している．

本論文の目標は，図 1.4 で示すように，大規模かつ中央集中化しているネットワーク管理に既存のネットワーク管理手法である SNMP を共存させることを目標にし，Tambourine のようなプロキシによる新たなキャッシュ管理手法を提案することにある．課題の設定と提案をする前に，以下に運用ビジネス上の制約条件を述べる．

第 1 に ネットワークの運用としては，数千社から数万社を対象とした VPN が設定され，それを管理する管理者が数百人規模で存在する場合を想定する

第 2 に 世界のキャリアネットワークの動向を考え，運用の一部はアウトソースされ，運用者全体の統制が取りにくいことも制約条件に入れる．

1.4　本論文の構成

本論文の構成は以下の通りである．

第 1 章 序論
第 2 章 関連研究
第 3 章 設計条件
第 4 章 実装と評価
第 5 章 考察
第 6 章 結論
第 7 章 謝辞と参考文献

まず，2 章では，関連研究，既存のネットワーク管理技術の特徴と問題について，特に SNMP リバースプロキシ Tambourine の基本構成やキャッシュ管理アルゴリズム，並びに Tambourine の課題，および TCP の輻輳制御や Web cache 管理について説明する．3 章では，大規模なネットワーク管理を可能にするための通信エンティティの要求条件を明確にした上で，設計上の制約条件を詳細に検討する．そして，4 章で TCP Reno の輻輳制御手法に類似した機構に RED に似た手法と組みあわせることにより，モニタリングの公平性を改善できる SNMP 機器負荷制御手法を提案する．5 章では，シミュレーションにより，提案手法を SNMP 機器の CPU 使用率抑制，NMS 間のアクセスの公平性およびモニタリング精度の観点から評価し，6 章ではスケーラビリティに関する議論をする．7 章は結論である．

第2章

関連研究

2.1　はじめに

本章では，まず本研究の背景になるネットワーク管理に関する主な技術について説明する．インターネットが始まった時期に使われた技術からネットワーク管理の自動化や仮想化のために注目を集めている技術まで説明する．次に，Web cache と TCP の Flow Control や RED がなど提案手法と関連する技術を説明する．

2.2　主なネットワーク管理技術

電話網にオーバーレイして構成されていた初期のインターネットでは各 ISP の管理規模は小さく，管理者が直接機器へアクセスしモニタや設定変更を行う手法でも十分運用できていた．しかし，現在では IP ネットワークにオーバーレイして電話サービスを行なうように主客が転倒している．各 ISP の所有する IP 系のネットワーク機器の数は飛躍的に増加し，コア・ルータとエッジ・ルータを合わせて数千台になることもめずらしくなくなっている．大規模ネットワークの出現により，マニュアル管理から自動化への道を進むのは必然であり，CLI(Command-line interface) から SNMP や NETCONF といった管理ツールも進化してきた．この節では，代表的なネットワーク管理ツールに関して解説する．

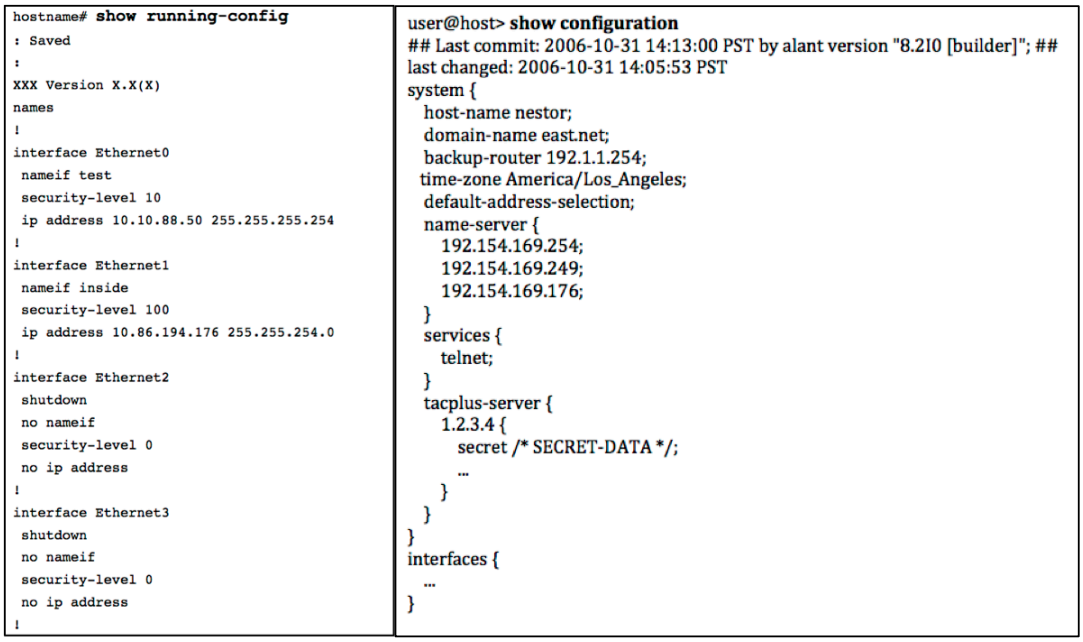
2.2.1　CLI

CLI(Command-line interface) は，初期のネットワーク管理に置いて一番重要な tool であった．CLI を一言で言うと，文字入力を基盤とするユーザとパソコンとのインターフェイスであり，ユーザに対する表示は文字によって行い，入力はキーボードを用いて行う．初期のルータなどのネットワーク機器は Unix 上のソフトウェアであったため，自然と CLI がネットワーク管理にも使われるようになった．help などコマンドを案内する機能や，コマンドを一部を入力すると残りは tab キーで自動的に補完する機能，コマンドと Unix の演算子を組み合わせて望む結果を表示することなど，手打ちでコマンドを入力する不便さを克服するための種々の工夫がされて来た．例えば，図 2.1 のように，Unix の pipe と連携し“ Ethernet ”と始まる内容だけを表示することも可能である．

```
Router# show interface | begin Ethernet
Ethernet0 is up, line protocol is up
Hardware is Lance, address is 0060.837c.6399 (bia 0060.837c.6399)
  Description: ip address is 172.1.2.14 255.255.255.0
  Internet address is 172.1.2.14/24
.
.
.
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
--More--
+Serial
filtering...
Serial1 is up, line protocol is up
Serial2 is up, line protocol is up
Serial3 is up, line protocol is down
Serial4 is down, line protocol is down
Serial5 is up, line protocol is up
Serial6 is up, line protocol is up
Serial7 is up, line protocol is up
```

図 2.1: 複数のコマンドと演算子の組み合わせた CLI の使用例 [18]

しかし，CLI はもともとネットワーク管理のためのツールとして設計されたものではなかったため，ネットワーク管理の道具としては次のような問題が指摘されて来た [17] ．まず，表 2.1 のように同じ目的に使用する場合でもネットワーク機器のメーカや機器の OS 毎にコマンド名が異なり，異なる種類の機器が存在するヘテロジニアスなネットワークの管理を行わければならない状況では，管理者は全てのコマンド名の使用をマスタしなければならず，過度な運用負荷がかかることになる．そして，もっと重要な問題は，同じ目的のコマンドに対するレスポンスが各メーカや機器の OS 毎に異なった形式で表示されるため，結果を分析し自動化することが困難であった．例えば，図 2.2 には Cisco と Jupiter 社の機器上で，現在の設定情報を示すコマンドとその結果を示したものである [19] [20]．使うコマンドと結果表示が共に異なることが分かる．



```
hostname# show running-config
: Saved
:
XXX Version X.X(X)
names
!
interface Ethernet0
 nameif test
 security-level 10
 ip address 10.10.88.50 255.255.255.254
!
interface Ethernet1
 nameif inside
 security-level 100
 ip address 10.86.194.176 255.255.254.0
!
interface Ethernet2
 shutdown
 no nameif
 security-level 0
 no ip address
!
interface Ethernet3
 shutdown
 no nameif
 security-level 0
 no ip address
!
```

(a) Ciscoの結果

```
user@host> show configuration
## Last commit: 2006-10-31 14:13:00 PST by alant version "8.2I0 [builder]"; ##
last changed: 2006-10-31 14:05:53 PST
system {
  host-name nestor;
  domain-name east.net;
  backup-router 192.1.1.254;
  time-zone America/Los_Angeles;
  default-address-selection;
  name-server {
    192.154.169.254;
    192.154.169.249;
    192.154.169.176;
  }
  services {
    telnet;
  }
  tacplus-server {
    1.2.3.4 {
      secret /* SECRET-DATA */;
      ...
    }
  }
}
interfaces {
  ...
}
```

(b) Juniperの結果

図 2.2: 同じ目的に対しメーカ毎異なるコマンド形式の例 [19] [20]

2.2.2　SNMP

CLI が持っている短所である，ネットワーク機器のコマンドとレスポンスがメーカや OS 毎に異なる形式を持つ点に関しては，SNMP(Simple Network Management Protocol) [1] により解決された．SNMP を一言で言うと，IP ネットワーク上の機器を監視・制御するためのプロトコルである．図 2.3 で示すように，SNMP によるネットワーク管理は NMS(network management system)，Agent，SNMP プロトコルそして管理情報で構成される．NMS はネットワーク管理者が利用するネットワーク管理ステーション上でのアプリケーションで，管理対象に対し命令を下し，管理対象からのレスポンスを蓄積や分析する役割をもっている．Agent は管理対象上のアプリケーションで，NMS からのリクエストに応える役割をする．管理情報は Agent が管理するネットワーク機器に関する情報であり，管理情報データベースとしてネットワーク機器に保存されている．最後に SNMP プロトコルは，NMS と Agent の間に通信を行うために使うプロトコルである．本節では，SNMP によるネットワーク管理の中で管理情報と SNMP プロトコルを説明した上で，SNMP を利用して大規模ネットワーク管理を試みた既存技術やその課題について述べる．

管理情報

ネットワーク機器は管理情報をデータベースとして持っていて，Agent は NMS からの問い合わせに応じ機器の状態を調べる際に，管理情報のデータベースへアクセスする．管理情報のデータベースは MIB(Management Inforation Base) [22] [23] に従って定義され，管理対象オブジェクトから構成されている．管理対象オブジェクトとは，オブジェクト識別子とその値から成り [24]，Structure of Management Information (SMI) [31] によってその構造と使用可能なデータ型が定義

表 2.1: Cisco と Juniper のコマンドの比較 [21]

Cisco Command	Juniper Command	Co-Ordinating Definition
show running-config	sh configuration	Show running configuration
show processes cpu	show system process	displays utilization statistics
show logging	show log messages	display the state of logging to the syslog
ping dest	ping dest rapid	to check to see if a destination is alive
terminal monitor	monitor start messages	Change console terminal settings
terminal length 0	set cli screen-length 0	sets the length for displaying command output

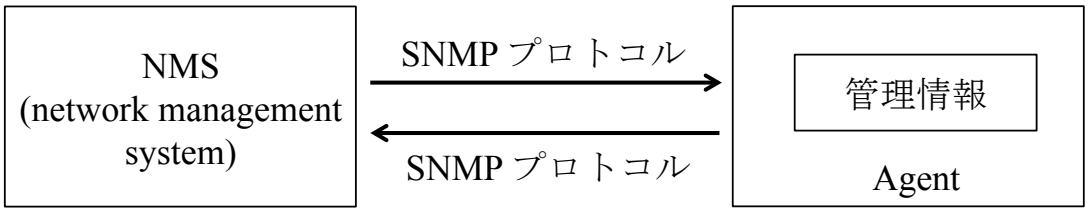


図 2.3: SNMP によるネットワーク管理の基本コンポネント

されている．SMI で管理対象オブジェクトとオブジェクト配列を定義する際の公式モデルとしては抽象構文表記法 1 (ASN.1: Abstract Syntax Notation One) が使われ，エンコード方式としては BER(Basic Encoding Rules) が採用されている [25].

ASN.1 は，電気通信やコンピュータネットワークでのデータ構造の表現・エンコード・転送・デコードを記述する標準的かつ柔軟な記法であり [26] ，ITU-T Recommendation X.209 [27] および X.690 [28] に定義されている．例えば，データの型，モジュールとその構造，INTEGER，BOOLEAN，構造型，キーワードの意味 (BEGIN，END，IMPORT，EXPORT，EXTERNAL など)，型が正しくエンコードされるように “ タグ付け ”する方法などが定義されている [29]．SMI は，この ASN.1 で定義されている．しかし，ASN.1 は特定の標準，エンコード方法，プログラミング言語，ハードウェアプラットフォームとは独立であり，例えばエンコード方式として前述した BER 以外にも DER (Distinguished Encoding Rules) と CER (Canonical Encoding Rules) を使うこともできる [28].

BER(Basic Encoding Rules) は [28]，TLV 符号化と呼ばれるエンコード方式で，ASN.1 で使われるエンコード方式の一種であり，SNMP でのエンコード方式として使われている．TLV とは，Type(型)，Length(長さ)，Value(値) の略であり，図 2.4 で示すように，BER による符号化後の情報は，Type・Length・Value の 3 つのフィールドに構成される．Type(型) フィールドは ASN.1 のデータ型を示し，Length(長さ) フィールドは復号化された値の長さを，Value(値) フィールドは値そのものを意味する．

前述したように SNMP では，ASN.1 で作った SMI により管理対象オブジェクトを定義している．SMI には，SMIv1 [31] と SMIv2 [32] があり，本章では SMIv1 について簡単に説明する．SMI による管理対象オブジェクトの定義は，オブジェクト識別子・構文・符号化の 3 つの項目からなる．オブジェクト識別子は，管理対象オブジェクトを識別するための識別子であり，数字と文字を用いる 2 つの表

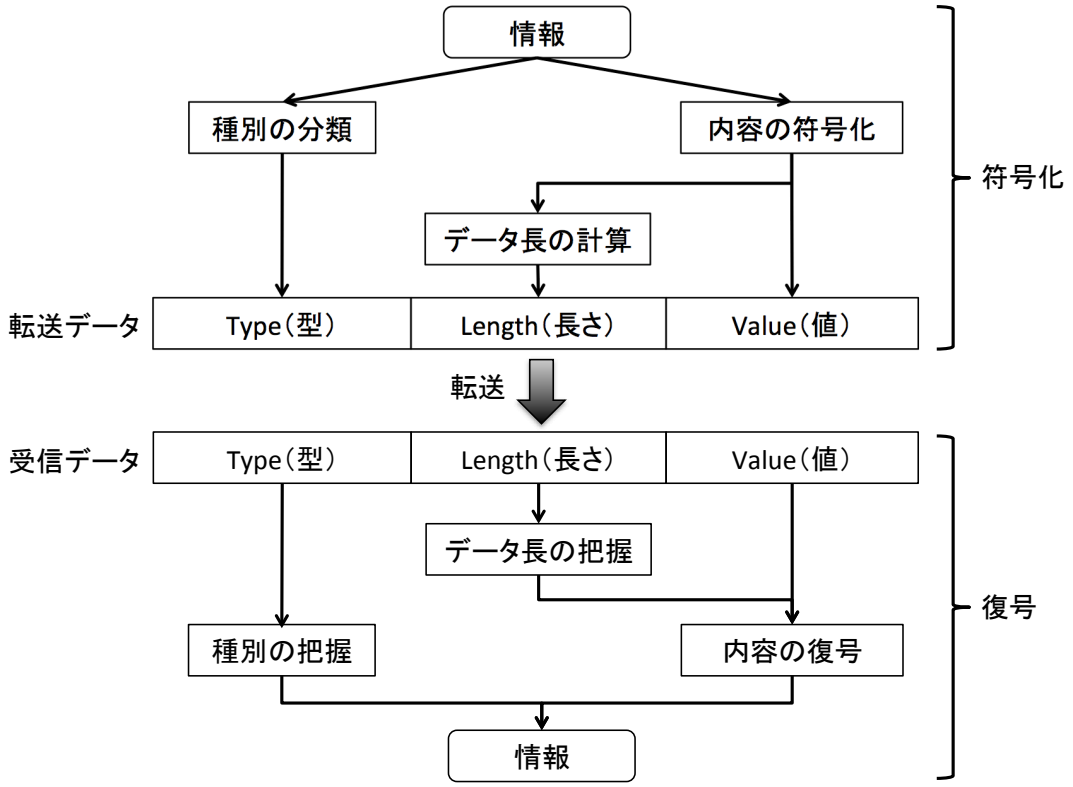


図 2.4: BER による情報の符号化と復号 [30]

現法がある．オブジェクトの識別子は OID(Object IDentifier) と呼ばれる．次に構文は管理対象オブジェクトの値を分類するもので，大きく分けて Primitive 型・Constructor 型・Application-wide 型がある．Primitive 型には，INTEGER・OCTET STRING・OBJECT IDENTIFIER・NULL の４種類があり，ASN.1 で定義するデータ型を利用する．Constructor 型は，SEQUENCE・SEQUENCE OF の２種類があり，いわゆる構造体を示すためのデータ型である．Application-wide 型は，Primitive 型や Constructor 型では表現できないデータ，例えば NetworkAddress・IpAddress などを表すために SMI で定義したデータ型である．最後に符号化は前述の BER のことである．

SMIv1 は図 2.5 のようなツリー構造を定義していて，SNMP による機器管理と関係があるのは mib-2 サブツリーと enterprise サブツリーである．各サブツリーの説明の前に前述した OID について internet(1) の例をもって説明する．internet(1) はルートノードから iso(1).org(3).dod(6) を経由している．OID を文字で表現すると，.iso.org.dod.internet になり，数字で表すと，.1.3.6.1 となる．mib-2 サブツリーには一般的な管理対象オブジェクトが定められていて，例えば，system には管理者の名前や連絡先など機器の一般的情報，ip には IP アドレスや MAC アドレスなど管理上もっとも重要な情報が定義されている．enterprise サブツリーでは，機器のメーカに依存する管理情報オブジェクトを定めていて，拡張 MIB とも呼ばれている．個人や企業，組織が自由に管理対象を定義することが可能である．SNMP による機器管理は主に，mib-2 サブツリーと enterprise サブツリーを組み合わせて管理を行っている．

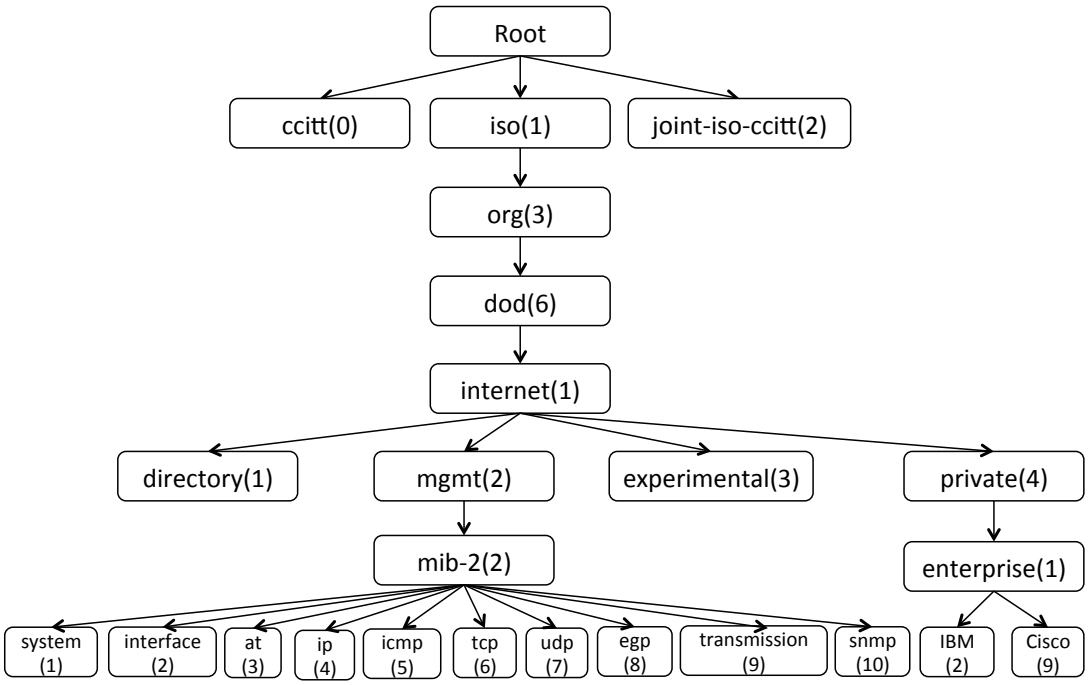


図 2.5: SMIv1 のオブジェクトツリー

SNMP プロトコル

ネットワーク機器の管理情報の獲得や設定をするため NMS と Agent は SNMP プロトコルを利用している．SNMP プロトコルには SNMPv1 [1]，SNMPv2 [33] そして SNMPv3 [34] があるが，本論文では SNMPv1 について簡単に説明する．図 2.6 で示したように SNMP プロトコルではネットワーク層としては IP をトランスポート層としては UDP を採用している．そして SNMP パケットは，使用している SNMP のバージョンを示す Version フィールド，NMS と Agent の間の認証に使われる Community Name フィールド，そして管理情報の取得や変更などのオペレーションのコマンドを含む PDU(Protocol Data Unit) フィールドで構成されている．Community Name とは，NMS と Agent との間のパスワードであり，read-only と read-write そして trap の 3 種類の Community Name がある．read-only community name を使うと，例えば NMS は Agent の特定の port の転送パケットの数を読み取ることはできるものの，Agent の設定を変えることはできない．read-write community name の場合は，Agent の設定変更も可能である．trap community name は，Agent が非同期通知を NMS へ送る場合使われる．このような Community Name を利用した認証の問題は，SNMPv1 と SNMPv2 の場合，Community Name が平文で転送されることであり，傍受される危険があった．SNMPv3 では MD5 や SHA1 などでハッシュ関数を使うことで平文のままで Community Name を転送することは避けている．

次に PDU の場合，SNMPv1 [1] には，GetRequest, GetNextRequest, GetResponse, SetRequest, Trap の 5 つの PDU があり，GetRequest, GetNextRequest, SetRequest の 3 種類は NMS から Agent に対し送信され，GetResponse と Trap は Agent から NMS へ送信される．SNMP のオペレーションは大きく分けて，Polling, Setting, Trap に分類することができる．Polling とは，NMS が Agent から管理情報を取得する動作であり，GetRequest と GetNextRequest の PDU が使われ

る．Setting は，NMS が Agent の管理情報の値を変更する操作で，SetRequest が使われる．Polling や Setting のオペレーションによる Agent からのレスポンスは，GetResponse PDU を用いている．最後に Trap とは，管理対象に状態の変化やトラブルが発生した場合 Agent から NMS へ通知するためのオペレーションで，Trap PDU が使われる．

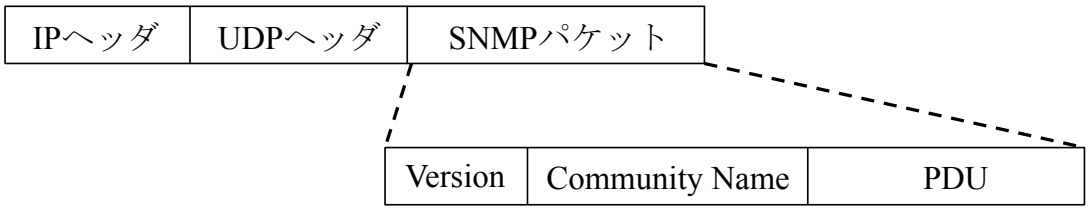


図 2.6: SMIv1 メッセージパケットのフォーマット

このような SNMP を使うことで，マルチベンダーで構築されたヘテロジニアスなネットワークでも標準的な設定情報やトラヒック情報を取得するなどネットワーク管理を行うことが可能になった．しかし，SNMP は設定よりモニタリングに力点が置かれたことや，enterprise サブツリーのような拡張 MIB に残されている機器メーカへの依存性など大規模かつ自動化されたネットワーク管理への需要を満足するものにはなっていない．

SNMP リバースプロキシ Tambourine

Hamid 等による SNMP リバースプロキシ Tambourine [15] は MIB 情報をキャッシュすることにより，Web からのネットワーク機器への過度な問い合わせを防ぎながら大規模なネットワーク管理を可能にした．図 2.7 に Tambourine で行う SNMP/HTTP 変換手順を HTTP Get request の場合について示す．ここで，host は Tambourine を示す FQDN もしくは IP アドレスであり，agent は管理対象の SMNP 機器を管理する Agent の IP アドレスである．SNMP の Agent は SNMP 機器上の MIB 情報にオブジェクト識別子 (以下 OID) を割り当てて管理する．Tambourine は，MIB 情報をリソースと見なし，各 MIB 情報の OID(oid) に http://host/tbsnmp/get/agent/community/oid のような URI (Uniform Resource Identifier) でアクセスすることを想定している．管理者がこの URI に対して HTTP Get request を送ると，Tambourine はその oid に対する SNMP GetRequest コマンドに変換し，関連の SNMP 機器の Agent に SNMP プロトコルで転送する．ここで community とは SNMP プロトコルにおけるコミュニティ名である．SNMP GetNextRequest の場合は，上記 URI の get を getnext に置換すれば良い．

図 2.8 に Tambourine のソフトウェア構造を示す．URI Manager は，URI で記述されているリソースを管理し，NMS からの HTTP Get request を処理する．NMS は図 1.3 の管理者が使うネットワーク管理用機器である．Request Processor は，SNMP Polling Driver を呼び出し，SNMP の応答を所定の XML 形式に変換して URI Manager 経由で NMS に返す．また，SNMP トラップ情報は Weblog ping receiver [35] に転送して処理する．

SNMP GetRequest:
General syntax
GET http://host/tbsnmp/get/agent/community/*oid*
Example
GET http://example.com/tbsnmp/get/agent/private/.1.3.6.1.2.1.1.6.0

SNMP GetNextRequest:
General syntax
GET http://host/tbsnmp/getnext/agent/community/*oid*
Example
GET http://example.com/tbsnmp/getnext/agent/private/.1.3.6.1.2.1.1.6.0

図 2.7: HTTP Get request の SNMP GetRequest への変換 [15]

Tambourine はキャッシュ機構により，SNMP 機器の負荷を上げずに，HTTP 経由でモニタができる．図 2.9 に示す $OID_i(1 \leq i \leq M$, M は OID の総数) の MIB 情報を保存するキャッシュ(以下 $cache_i$) の TTL 制御手法では，NMS が，ある OID_i に対応した URI_i で HTTP ポーリングをすると (図 2.9 の (2))，Tambourine は $cache_i$ の TTL(以下 TTL_i) タイマーがタイムアウトしてなければ (図 2.9 の (3))，$cache_i$ の値を NMS に返す (図 2.9 の (12))．タイムアウトしていた場合，SNMP GetRequest で Agent に OID_i を問い合わせて，SNMP GetResponse で新しい MIB 情報と既存の値（LFV: last fetched value）を比較し (図 2.9 の (5))， $cache_i$ が変わっていなければ TTL_i を 2 倍に増やし (図 2.9 の (9))，変わっていたら 1/2 に減らす仕組みである (図 2.9 の (6))．ttl_max と ttl_min は TTL の最大値と最小値であり，TTL_i が ttl_max と ttl_min の間に値するよう調整した後 (図 2.9 の (6), (7), (8), (9), (10), (11))，$cache_i$ の値を NMS に返す (図 2.9 の (12))．

図 2.9 の $cache_i$ の制御手法は，OID_i の MIB 情報の変化速度に動的に対応し，OID_i に最適な TTL_i を割り当てることができることを，Hamid 等 [15] は，SNMP Ratio（SNMP GetRequest の数/ HTTP Get request 数）とリバースプロキシ上のキャッシュの Staleness ratio を使って示した．しかし，頻繁に値が変化する OID の現在の TTL 値が TTL_i のときに DDoS 攻撃があると，このアルゴリズムでは TTL がタイムアウトするたびに図 2.9 の (5)-(6)-(7)-(8) の処理により 1/2 になるため，$2TTL_i$ 秒程度で OID の TTL が ttl_min に達する．この結果，SNMP 機器の CPU 使用率が過大になるという脆弱性があった．

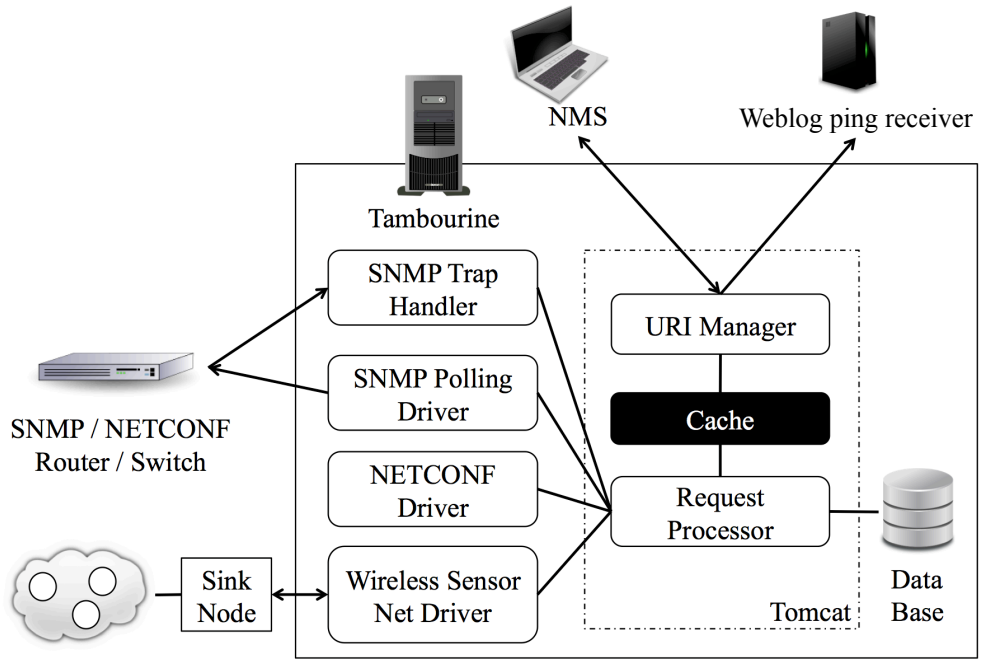


図 2.8: Tambourine のアーキテクチャ [15]

OpenView

OpenView は，Web をヒューマンインターフェースとして SNMP による大規模なネットワーク管理を試みた代表例であった [36]．OpenView は，アメリカの Hewlett Packard 者がネットワーク管理ツールとして 2007 年まで販売した商品で，2000 年代初めには市場占有率が一位に達するほど [37] [38] 普及していた．OpenView は主に NNM(Network Node Manager)，CV(Customer Views) と SIP(Services Information Portal) で構成されている．NNM は SNMP をベースに，管理対象となる機器の発見，マッピング，データの収集を行い，SNMP GetRequest と trap 両方を使い管理対象からデータを収集する．CV は一つのネットワーク上に複数の管理者から管理される機器が混在している場合，管理者と機器の紐付けを行い，機器が障害になったとき，その影響を受ける管理者を特定できる機能がある．SIP は NNM が収集したデータに，Web 経由でアクセスするために必要な Web サーバである．NNM が管理対象から収集したデータは NNM のデータベースに保存され，SIP によって Web ベージとして管理者に公開される．管理者は CV からアクセス権限を得たリソースに対し Web ブラウザを使ってアクセスできる．OpenView は NNM がネットワーク機器から SNMP で得たデータをキャッシュし，Web 経由で管理者がアクセスできるようにした点では Tambourine と類似している．しかし，Tambourine と同じく，NNM がネットワーク機器から SNMP GetRequest でデータを収集する場合，ネットワーク機器の負荷を考慮してないため DDoS 攻撃によりネットワーク機器が過負荷になる問題を持っている．

2.2.3　NETCONF

インターネットがすべての産業へ普及されることに連れ，多様なメーカによりネットワーク装置が開発された．しかし，複数のメーカの機器が同時に駆動される大規模なネットワークを CLI と

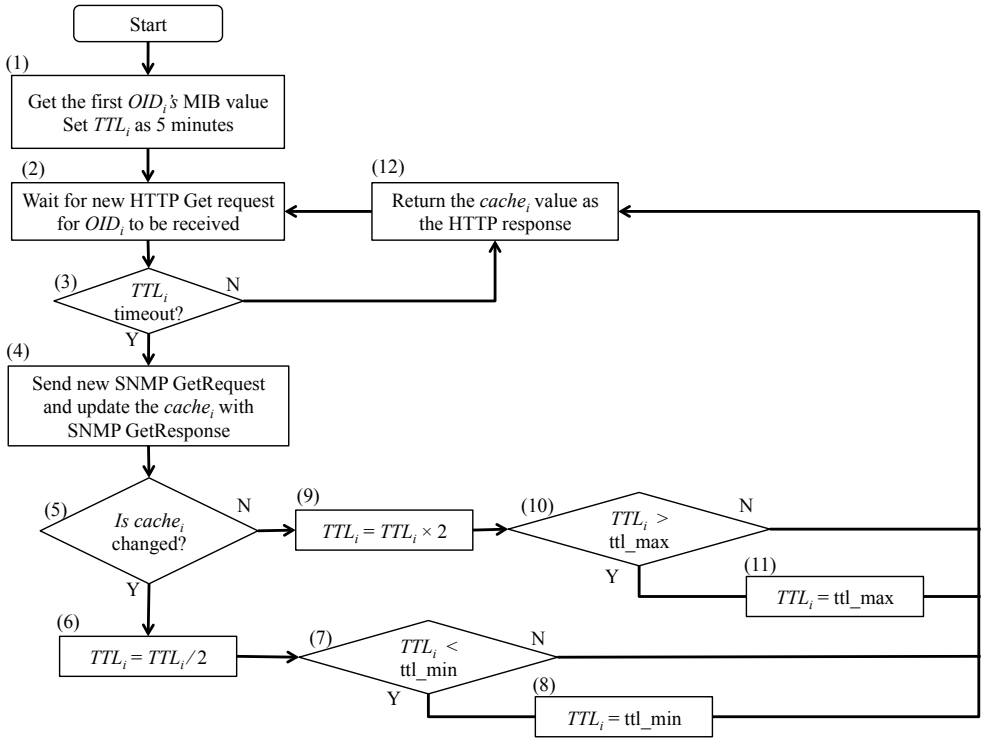


図 2.9: Tambourine のキャッシュ$cache_i$ の管理

SNMP で管理するには限界があった．2000 年に IAB(Internet Architecture Board) が開いたネットワーク管理に関する国際研究会 [39] でも指摘があったように，CLI の問題点としては，ヒューマンインターフェイスとして実装され自動化が困難であること挙げられた．SNMP の場合，仕様上は機器の設定も可能であっても，設定のロールバックやトランザクションなど設定に必要な機能が不足しているため実際にはモニタリングと通知にだけ使われていることが挙げられた．また，SNMP の場合 UDP をトランスポート層として使っていることから送られる SNMP メッセージの最大の大きさが 484 バイトであるため，大量かつ大きいサイズの管理情報 (bulk data) の転送には向いていない [40] 問題もあった．以上の SNMP の限界を踏まえ，ネットワーク機器の制御機能の充実を狙って標準化が進められたのが NETCONF である．NETCONF(Network Configuration Protocol) は 2006 年に RFC4741 [41] として IETF の Working Group(WG) である NETCONF(Network Configuration) [2] により標準化された．その後，2011 年に機器操作の拡張やエラー処理，ロックなどを含めて RFC6241 [2] として公開された．本節では RFC6241 を参考し NETCONF について簡単に説明する．

アーキテクチャ

NECONF のメッセージは図 2.10 で示すように多様なネットワーク機器を管理するため，4 つのレイヤの階層構造を持っている．表 2.2 には前節 SNMP と NETCONF の比較を示した．SNMP が管理者端末 (NMS) と管理対象 (Agent) からなっていたのと同じく，NETCONF も管理者端末 (client) と管理対象 (server) から成る．用語は違うが，いずれもクライアントサーバモデルのコンピュー

表 2.2: SNMP と NETCONF の比較

	SNMP	NETCONF
Data models	MIB	XML スキーマ or YANG Core Models
Data modeling Language	SMI	XML スキーマ or YANG
Management Operations	SNMP	NETCONF
Message Encoding	BER	XML
Transport Stack	UDP	SSH/BEAP/SOAP

タネットワークである．Content 層は client と server の間ででやり取りする data(configuration や notification など) の構成について定義している．SNMP では SMI で表現した MIB が content 層に該当し，NETCONF では XML や YANG [10] によって表現される．Operations 層は管理対象の configuration data の取得や修正を行うためのオペレーションを定義し，SNMP のプロトコルの PDU に該当する．Messages 層は，RPC（Remote Procedure Call）方式を利用するため，XML を用いてメッセージの符号化や復号のための tag を定義している．SNMP では BER に該当する．最後に Transport 層では，安全にメッセージを転送するためのプロトコルとして，SSH や BEAP または SOAP が使われている．SNMP では UDP が使われている．以降各層に関して説明する．

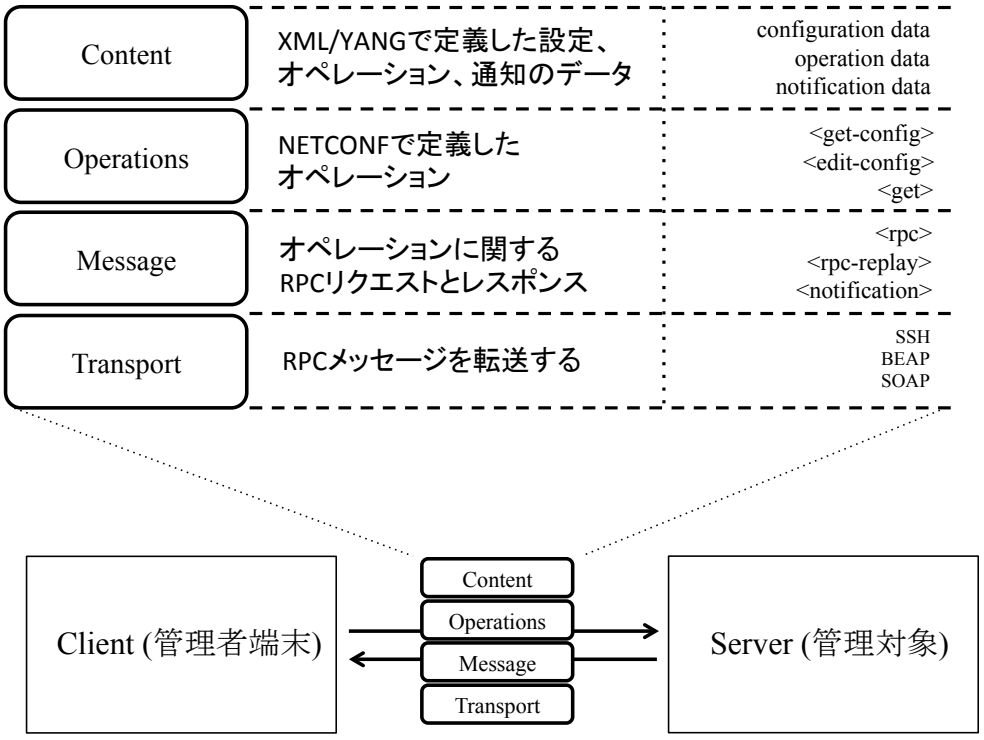


図 2.10: NETCONF のプロトコルレイヤ

NETCONF の Content 層では，管理者端末が管理対象の構成情報を容易にモニタリングや設定できるようメッセージを主に XML や YANG を使って構造化している．XML(Extensible markup

language) とは [9]，1998 年 World Wide Web Consortium(W3C) により策定・勧告されたデータ記述のための言語であり，仕様が簡単であるためインターネット上で様々なデータを扱う場合に広く使用されている．図 2.11 では，XML による管理対象 (server) から hello メッセージを Client に返しているところを示している [42] ．XML 文書はヘッダー，エレメント (tag)，アトリビュート（属性）などで構成されるが，XML 自身はこれらの意味や関連性などを定義しておらず，スキーマ言語と呼ばれる言語を用いて定義している．これを XML スキーマと呼ぶ．しかし，XML による Content 層の構成は，メーカ毎に異なっていたため，マルチベンダーで広く運用を行う為にはデータモデルの定義が必要になった．

YANG(Yet Another Next Generation) とは，全てのネットワーク機器で NETCONF が使用できるよう，IETF NETMOD WG が共有のデータ・モデルを定義したものである [10]．YANG の特徴は，Hui Xu 等 [43] が示したように，簡潔な文法とデータの階層化によって人間だけでなくコンピュータにも解読し易いのに加えて，さらに管理情報を Web の GUI によって XML へスムーズに変換できるツールなどを備え，管理が容易になるように配慮されている．H, Cui 等 [44] は YANG のメリットを，“ Data Structure ”,“ Object-Oriented ”および“ Similarity to Programming Languages ”と分類して YANG の利点を延べている．

1 Data Structure：YANG では，データを木構造で構成し，各データ (以下ノード）の意味とノード間の関係を簡略かつ明確に提供してる．

2 Object-Oriented：Java や C++で使われるオブジェクト指向型データ構造が提供されている．YANG のデータ・モデルはモジュールとして特定の要素を import や export することが可能であり，データ・モデル間で共有することが可能である．

3 Similarity to Programming Languages：YANG の文法やデータ・タイプの定義は，C 言語や C++に似ていて，プログラミングを開発する際，YANG のデータ処理が容易になるよう配慮されている．

Operation 層では，SNMP で提供している“ get ”, “ set ”,“ trap ”に対応するオペレーションだけでなく，transaction 処理，つまり，オペレーションの実行の間何らかの理由でエラーが発生した場合，オペレーションを実行する前の状態に戻る“ rollback ”，またはエラーが発生した時点で処理を止めエラメッセージを管理者端末へ送る“ stop-on-error ”などが定義されている．また，構成情報へ複数の管理者端末が同時に接続し修正を試みる場合，構成情報の一貫性を保つため構成情報に対し'lock' を掛けるオペレーションなど，SNMP では提供していない新しいオペレーションも追加している．

Message 層では，管理者端末からのリクエストメッセージと管理対象からのレスポンスメッセージを定義し，RPC（Remote Procedure Call）でメッセージのやり取りをする．NETCONF では $< rpc >$ tag で各オペレーションを示す“ message-id ”を包んで送ることでオペレーションの実行を要求することができる．$< rpc - abort >$ tag はオペレーションの中止を，$< rpc - progress >$ はオペレーション要求の進捗報告の要求を意味する．リクエストメッセージに実行させたいオペレーションの tag を付けて送ると，受信側でその tag をパーシングしオペレーションを実行し結果に tag を付けてレスポンスメッセージとして返す仕組みになっている．

最後に Transport 層は管理者端末と管理対象との接続手順（connection）について定義している．色々な Transport プロトコルの中で，NETCONF は SSH, BEEP, SOAP/HTTP を採用している．各プロトコルについて簡単に説明する．

SSH(Secure Shall) [8] は CLI と共に既存のネットワーク管理で頻繁に使われている技術である．SSH 上で NETCONF を使うためには，まず client(管理者端末) が server(管理対象) に対し SSH プ

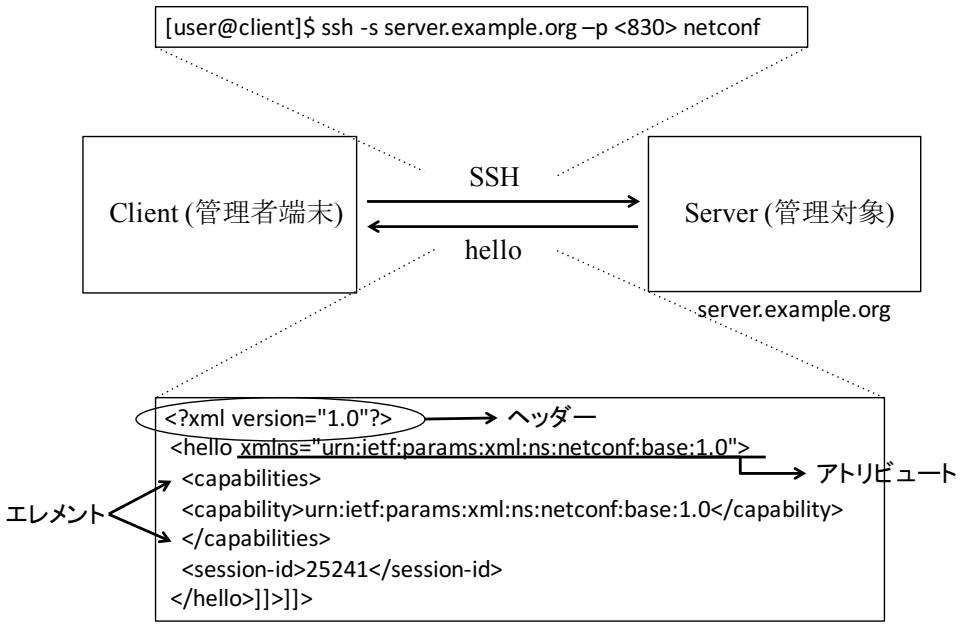


図 2.11: XML による NETCONF の hello 交換

ロトコルを使いコネクションを確立した後，メッセージの暗号化や復号のため必要なキーを交換する．その後，一般的な SSH コネクションと同じく“ ssh-userauth ”サービスによるユーザ証明や“ ssh-connection ”サービスを行う [45]．コネクションを確立した後，管理対象の NETCONF を SSH の subsystem として呼び出すことができる．ここで，subsystem とは client が SSH で server に接続した後に呼び出すことができるコマンドである．例えば，URL が“ server.example.org ”である server に 830 番の port で SSH アクセスし，netconf を立ち上げる命令は下記の通りである．このとき，server は図 2.11 で示すように“ hello ”メッセージを送る．

[user@client]$ ssh -s server.example.org -p < 830 > netconf

NETCONF のトラヒック把握を容易にし firewall などで処理をするため，IANA(Internet Assigned Numbers Authority) が割り当てた TCP port 番号，830 番，を使うことが義務付けられている．しかし，port 番号が公開されているのはセキュリティ上好ましくないことや，SSH を使う場合，管理者端末と管理対象の間は一つの TCP コネクション (NETCONF では channel と呼ぶ) しか使えない．このため，管理対象から管理者端末へ必要に応じたメッセージを非同期的に送ることができない問題がある [46].

BEEP(Block Extensible Exchange Protocol) [47] は Cisco や Juniper Networks のようにネットワーク機器のメーカが中心となって進めている Transfer プロトコルで，SSH と異なって複数の channel を使い非同期的にメッセージの伝送ができるメリットがある．また，client と server 構造ではなく，peer-to-peer 構造であり，管理者端末だけでなく管理対象も管理者端末へ管理情報の問い合わせができる特徴を持つ [48]．つまり，管理者端末と管理対象はイニシエータやリスナー両方を実装している必要がある．コネクションをつくる手順は，まずイニシエータがリスナーの TCP port 831 番に対して接続する．そして，hello メッセージで TLS(Transport Layer Security) [49] や SASL(Simple Authentication and Security Layer) [50] をサポートするかを確認した後，TLS や SASL のコネクションを作る．その後，管理対象と管理者端末はそれぞれ NETCONF での役割

（それぞれ server と client）を確認した上で，通常の NETCONF のやり取りを行うことになる．

表 2.3: NETCONF の Transport 層プロトコル

プロトコル名	TCP port 番号	TCP コネクション数	管理対象からの非同期的通信
SSH	830	一つ	不可能
BEEP	831	複数	可能
SOAP	833	一つ	不可能

最後に SOAP(Simple Object Access Protocol) [51] は NETCONF と一番親和度の高いプロトコルである．なぜなら，SOAP は RPC メカニズムを提供していることや SOAP の実装例が多いためである．しかし，SOAP は基本的に server と client 構造を持っているため，server である管理対象から client である管理者端末へメッセージを送れない問題がある．以上の内容を表 2.3 にまとめている．

しかし，NETCONF は標準化後，実装があまり進まなかった．ところが，SDN(Software Defined Networking) が登場し，2.2.4 節で説明したように OF-CONFIG の転送プロトコルとして NETCONF が使われたため，再度注目を浴びるようになった．

2.2.4　SDN

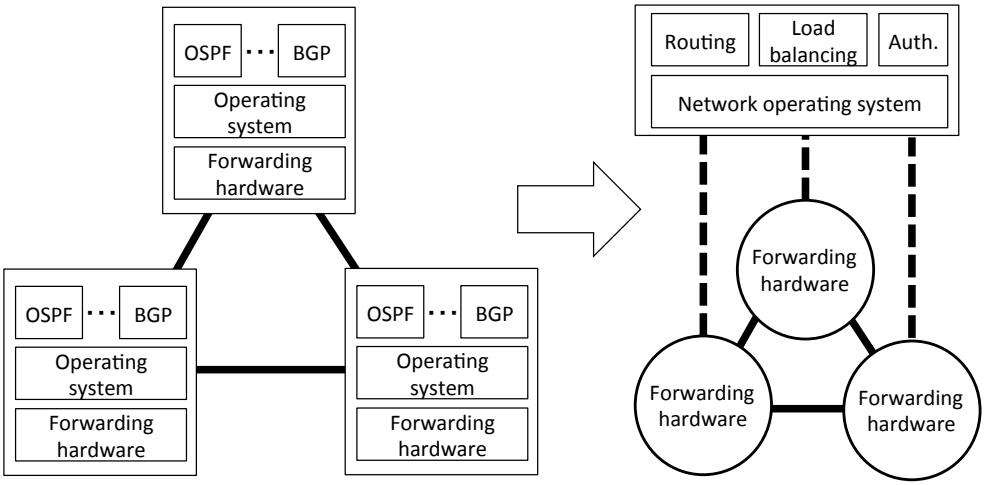


図 2.12: 自律分散ネットワークから Openflow による SDN へ [52]

既存のネットワーク機器は CLI や SNMP または NETCONF のような管理プロトコルによる設定に合わせて自律的に稼働していた．, 図 2.12 の左側で示すように，ネットワーク機器の OS(Operating System) 上のアプリケーションとして色々なプロトコルが実装されている．そのため，新しい機能を追加するには，その機能を標準化し，機器メーカに実装させて既存のネットワークへの投入することになり，新しいプロトコルの開発には混乱が起こることが多かった．SDN は図 2.12 の右側で示すように，ネットワーク機器にはデータを転送する機能のみ残し，データをどう転送するかは

Network operating system 上のアプリケーションが決める．データを転送するルールを決める機能を Control Plane，実際データを転送する機能を Data Plane または Forwarding Plane と呼ぶ．つまり，インターネットの特徴である自律分散型で既存のネットワーク機器上にあった Control Plane と Data Plane を，SDN では分離し Control Plane は中央のサーバへまとめ，分散されたネットワーク機器には Data Plane 機能のみを残している．

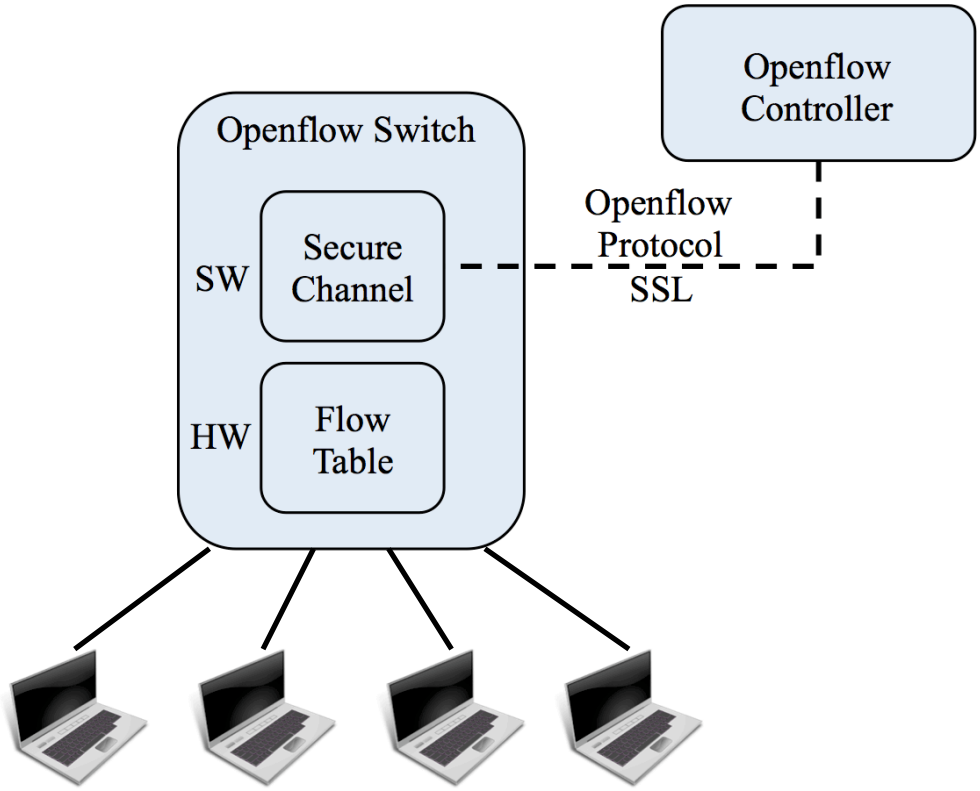


図 2.13: Openflow switch の基本構成 [11]

Openflow

Openflow [11] は，SDN を実現するためにもっとも適切な技術の一つとして評価され，SDN の中央コントローラと転送を行うネットワーク機器との間の通信プロトコルである．図 2.13 で示すように，Openflow は Openflow Controller と Openflow Switch で構成される．Openflow Controller は，最短経路や回線の速度そしてユーザの好みなどを考慮したパラメタの基に，packet の転送経路を計算する．Openflow Controller は，SSL など安全な channel 上で Openflow プロトコルを利用し，Openflow Switch の Flow Table を修正する．Flow とは，MAC アドレス，VLAN タグ，IP アドレス，TCP/UDP ポート番号のような特徴が共通なパケットのことで，入ってきたパケットの header を読んで，Openflow Switch はそのパケットがどの flow に該当するかを判断し，Flow Table に従い転送する．Flow Table は Openflow Controller で決定され送られるため，Openflow Switch へ既存のネットワーク機器のように Control Plane 機能を搭載する必要がなく，Openflow Switch を既存のネットワーク機器より安価で作ることも可能になる．Flow Table は flow entry の

集合体であり，各 flow entry は Match fields や Actions と Counters で構成される．Match fields は，Openflow Switch に到着した packet の header を読み取り，適切な table を選択するための条件であり，MAC アドレス，IPv4 アドレス，switch の port 番号などがあり，OpenFlow 1.3 には 40 個の条件が定義されている．Actions は packet の処理方法を定義し，転送や廃棄そして packet field の書き換えなどがある．Counters は，flow entry ごとのトラヒックの量を packet 数とバイトで表示したもので，Openflow Controller での経路の計算に利用される．

Northbound API

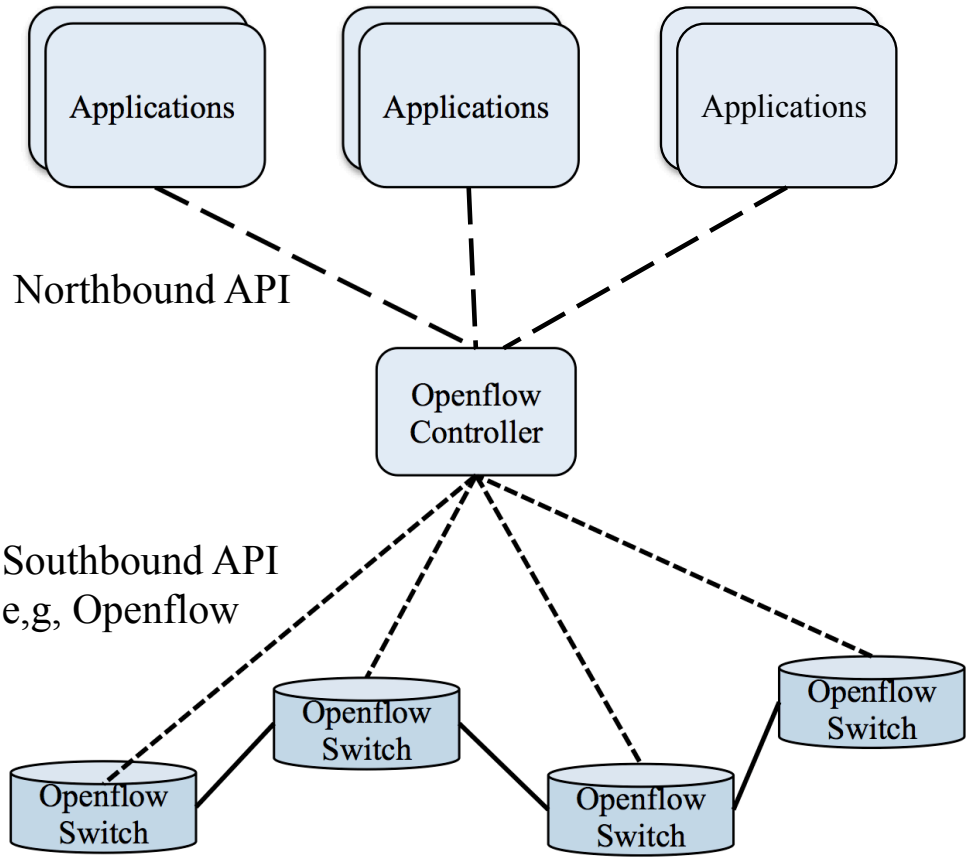


図 2.14: Northbound API と Southbound API

SDN により Control と Data plane を分離し，アプリケーションでネットワークの管理が可能になったが，色々なプロジェクトやメーカから Openflow Controller が開発され，各 Openflow Controller を制御するための標準が必要になった．つまり，異なる Openflow Controller を制御できる API の標準が必要になり，これが Northbound API である．図 2.14 に示す，Northbound と Southbound とは北が上位であるアプリケーションを南が下位である Openflow プロトコルを意味する．Northbound API の導入により，アプリケーションの開発者は細かな Openflow プロトコルに関する知識がなくても，ネットワーク機器をより抽象化の高いレベルで制御ができるようになった．

しかし，図 2.14 で示しているように，Northbound API と Southbound API による Control plane のトラヒックは全て Controller を介する構造であるため，SDN の初期から Controller のスケーラビリティの問題が懸念され，種々の研究がされてきた．Yeganeh 等は SDN のスケーラビリティに関する研究を次のように大きく 3 分類している [53] .

1 Controller の処理性能を上げる：NOX-MT はマルチスレッドで Controller を実装することで，既存の NOX（SDN Controller の一種）の性能を 30,000 flow/s から 1.6 million flow/s まで引き上げた [54] .

2 Openflow Switch のコントローラへの依存性を減らす：事前に全 data path を Controller へ転送しておく DIFANE [55] , Controller へ問い合わせを長い flow に限定している DevoFlo [56] が代表である．

3 複数の Controller へ負荷を分散させる：複数のコントローラとそれらを管理する API を提供する Onix [57] や，複数の Controller 間でネットワークの状態を同期させている HyperFlow [57] などがある．

しかし，このような研究はどれも Southbound API による負荷を軽減するためであって，Northbound API による負荷は考慮していなかった．その理由は，既存のネットワーク管理は少数の管理者によるものであったことと，今までの SDN の普及は主にデータセンタを中心に行われていたため，複数の管理者やアプリケーションによる Control Plane トラヒックは問題視されなかった．しかし，NFV など，SDN がデータセンタだけではなく，ISP のネットワークへ普及しつつあることや [4] , SDN を Northbound API を利用して各企業のネットワーク管理者に移行させる動き [58] など，Northbound API による大規模ネットワーク管理が増えているため，Control Plane トラヒックがコントローラに課す負荷に対応する必要がある．また，ユーザレベルに開放した場合，Northbound API 利用したコントローラや Switch へ DoS 攻撃が行われることも考えられる．

OF-CONFIG

Openflow プロトコルにより Openflow Controller が Openflow Switch に対し packet 処理のルールである flow table を管理しているが，機器の電源の on や off，キューのサイズの設定，機器の名前や管理者情報の更新など Openflow Switch 自体の管理も必要であり，OF-CONFIG(OpenFlow Configuration Protocol) が ONF から提案された [59]．図 2.15 に Openflow と OF-CONFIG 1.2 [59] で定義している構成要素を示している．まず，OpenFlow Capable Switch は OpenFlow Logical Switch を駆動することが可能な物理または仮想な機器であり，OpenFlow Resource を持っている．次に，OpenFlow Configuration Point は，OF-CONFIG でリモートの OpenFlow Capable Switch の管理や設定をする役割をする．また，OpenFlow Resource は，port やキューなど OpenFlow Capable Switch が持つ機能であり，OpenFlow Logical Switch は，OpenFlow Capable Switch の Resource で構成されている．最後に，OpenFlow Controller は OpenFlow プロトコルで OpenFlow Logical Switch を制御する．

OF-CONFIG 1.2 で定義している主な機能は，OpenFlow Capable Switch のキューや port の設定，OpenFlow Logical Switch を管理する OpenFlow Controller の割当，OpenFlow Capable Switch の port の up や down 設定，OpenFlow Logical Switch と OpenFlow Controller 間の安全なコミュニケーションの設定，OpenFlow Logical Switch の発見，GRE(Generic Routing Encapsulation)[1]

1 任意のプロトコルのパケットを IP プロトコルでカプセル化する技術で RFC1701, 2784 で定義されている

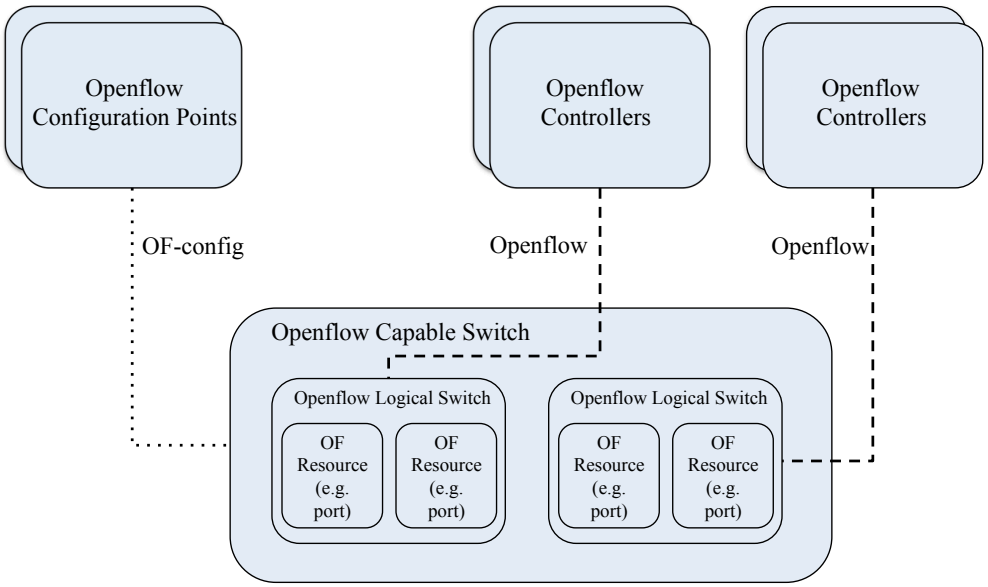


図 2.15: OF-CONFIG プロトコルと OpenFlow プロトコル [59]

や NV-GRE(Network Virtualization using Generic Routing Encapsulation)[2] と VxLAN(Virtual eXtensible Local Area Network)[3] などトンネル技術の設定などがある．

OF-CONFIG は転送プロトコルとして NETCONF を，データ・モデルとしては XML や YANG を使っている．NETCONF の採択により，各 OpenFlow Capable Switch のメーカは，標準化された設定プロトコルだけでなく，各メーカ独自の機能も管理すためプロトコルを拡張することができる．NETCONF をサポートするため，OpenFlow Capable Switch は SSH や rpc など図 2.10 のプロトコルを実装する必要がある．

OF-CONFIG を利用しグローバルスケールで大規模ネットワークを管理するためには，OF-CONFIG もリクエスト/レスポンス型であるため，図 2.16 で示すようにプロキシ・サーバを必要とする．多数の OpenFlow Configuration Point から OpenFlow Capable Switch に対しモニタリングを行う場合起こり得る問題は，SNMP による大規模ネットワーク管理の場合と同じである．

2 VLAN ID 不足など data center でのスケーラビリティ問題を解決するため，GRE を利用し layer 3 ネットワーク上で layer 2 パケットのトンネルを構成する仮想化技術で RFC 7637 で定義されている
3 VXLAN ID を利用して layer 2 のパケットを layer 3 ネットワーク上に論理的に構築するトンネル技術で RFC7348 で定義されている

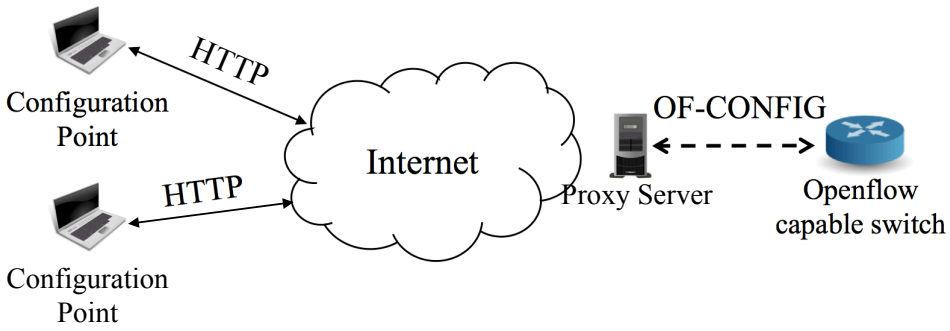


図 2.16: OF-CONFIG による大規模ネットワーク管理

2.3　Web cache

インターネットの通信形態（以下では end-to-end モデルと呼ぶ) は，末端の端末間の通信を中継するネットワークはベストエフォートで転送するだけで，通信に伴うエラー処理や認証などは各端末のネットワーク層以上のトランスポート層やアプリケーション層で行う構造になっている．End-to-end モデルのメリットとデメリットをまとめると下記の通りである [60]．メリットは，まずサーバでコンテンツの update を一回行うだけで，全てのクライアントが新しいコンテンツへアクセスできること．次に，サーバがアクセスするクライアントを把握でき，どのコンテンツに一番多くアクセスしているかなどの統計情報を獲得でき，クライアントごとに異なるセキュリティ・レベルを設定したり，コンテンツを提供できることにある．デメリットは，サーバが人気のあるコンテンツを同時に多数のクライアントへ提供する場合，サーバとネットワークへ大きな負荷がかかることである．この問題に対して，サーバと同じ機能を行う多数のサーバを配置し，トラヒックを分散させる方法が考えられた．しかし，これはサーバとクライアントの間のネットワークの負荷を削減する対策にはならないため，ネットワーク上のトラヒックを減らす Web cache が必要になった．

Web cache は主に browser である client, proxy, および server 上に配置される．Client のキャッシュは，例えば browser でユーザが以前表示したページに戻るさい，キャッシュしていたページを再表示する場合に使われる．Client のキャッシュは構成が容易というメリットがあるが，client 同士でコンテンツを共有することができず，トラヒックを減らす効果が少ない．Server 上のキャッシュは，頻繁に client から求められるコンテンツをキャッシュとして提供することで，server への負荷を減らし，client への応答を早くできる．Proxy 型キャッシュはキャッシュの位置によって，forward cache や reverse cache に分類される．Forward cache は client の近くに位置し，例えば企業の firewall や ISP のネットワーク上で運営される．企業のネットワーク上のすべての client からのトラヒックは firewall を通過すると想定されるため，企業内のユーザにより繰り返し求められるコンテンツを firewall 上の proxy に保存してオリジナルコンテンツがある server からコンテンツを受信する代わりに proxy から転送する仕組みである．Reverse cache はコンテンツを提供する server の近くに位置する proxy で，server への負荷を減らす目的で使われる．2.2.2 節の Tambourine も reverse cache に該当する．

表 2.4: TCP と Tambourine の負荷分散方式との間の類似性

	TCP	Tambourine
通信エンティティ	該当アプリケーションプロセス	OID キャッシュと SNMP エージェント
共有リソース	共有帯域幅	SNMP 機器の CPU 使用率 (Lmax (%))
輻輳検知	パケットロス	$L > L_{max}$
トラヒック制御	輻輳ウィンドウサイズ ($cwnd$)	$w_i = 1/TTL_i$ (requests/sec)

2.4　TCP の輻輳制御

TCP では，ACK を受信する前に送ることができるデータのサイズを輻輳ウィンドウ（以後 $cwnd$）として定義し，ネットワークの状況に応じて TCP の送信機が $cwnd$ を動的に変更することによりネットワークに送出するデータ量を調整する．表 2.4 で示したように，Tambourine のキャッシュ管理と TCP の輻輳制御の類似性を持っている．

Tambourine で，NMS から任意の OID_i ($1 \leq i \leq M$，M は OID の総数) への HTTP Get request 間隔（モニタ間隔）を m_i 秒，OID_i のキャッシュの TTL を TTL_i 秒，許容する CPU 使用率 L の上限を L_{max} (%)，その時の平均リクエスト数 B_{max} リクエスト/秒とすると，OID_i のキャッシュは $w_i = 1/TTL_i$ リクエスト/秒の情報源と考えることができるため，それらの OID_i のキャッシュが帯域 B_{max} リクエスト/秒の回線を共有する TCP の輻輳制御に類似した問題に帰着できる．このとき，w_i は TCP の輻輳ウィンドウサイズ ($cwnd$) に相当する．以上の類似性を基に，本論文では，TCP の輻輳制御に類似した Tambourine のキャッシュ管理手法を 4.3 節で提案する．

TCP の輻輳制御方式には，大きく分けてロスベース方式と遅延ベース方式の二種類があり，最近はこの 2 つの方式を融合したハイブリット方式の提案されている．本章では TCP 輻輳制御の古典的アルゴリズム [61] [62] を説明した後，ロスベース方式と遅延ベース方式の代表的例である TCP Reno [63] と TCP Vegas [64] について説明する．

2.4.1　古典的 TCP 輻輳制御

古典的な TCP 輻輳制御は [61] [65]，slow-start と congestion avoidance の 2 つのアルゴリズム（または phase）で構成されている．この 2 つのアルゴリズムが同時に実行されることはなく，$cwnd < ssthresh$ の場合は slow-start を $cwnd \geq ssthresh$ の場合は congestion avoidance を実行する．$ssthresh$ は slow start threshold を意味し，初期値は例えば advertised window(TCP 受信機が受信できるパケットの最大サイズ，以下 $awnd$) のように任意の大きい値として設定される．ちなみに，$cwnd$ の初期値は 1 である．

まず，slow-start phase は，TCP 送信機と受信機の間に TCP connection が新しく立ち上がった場合と，retransmission timeout (以降 RTO) が起こった場合選択され，ネットワークの輻輳を避けるためネットワークの状況で選択される．TCP 送信機は，slow-start phase では，送信した packet に対し ACK を受信すると，つまり 1 RTT ごとに $cwnd$ を 2 倍ずつ増やす．Slow-start phase の選択条件が $cwnd < ssthresh$ であるため，$cwnd$ の最大値は $ssthresh$ であり，$ssthresh$ の初期値は十分に大きいため [66]，$cwnd$ はネットワークが処理可能な packet サイズを超え packet drop など輻輳が発生するまで指数関数的に大きくなる．このような $cwnd$ の迅速な増やし方は，slow-start

と言う名前と矛盾するように見えるが，TCP 送信機は，*cwnd* を 1 から増やしていくため，slow と言える．TCP が slow-start で packet 転送を始める理由は，複数の TCP の送信機が共有するルータなどのネットワーク機器のキュー（ネットワーク容量）に対し，すべての TCP 送信機が最初から *awnd* で大きいサイズの packet を送ると，ネットワーク機器で packet drop が発生し，全ての TCP 送信機が輻輳を経験することを避けるためである．過度な *cwnd* の増加により輻輳が発生すると，TCP 送信機は，輻輳時の *cwnd* の半分で *ssthresh* を設定し，*cwnd* は 1 に初期化する．$cwnd < ssthresh$ の条件を満足するため，また slow-start phase により 1 RTT ごと *cwnd* を 2 倍に増やすが，*ssthresh* の値が前回輻輳時の *cwnd* の半分であるため，過度な *cwnd* の増加による輻輳は避けることが可能になる．Congestion avoidance phase では，前回の輻輳により設定された *ssthresh* を基準として，$cwnd \geq ssthresh$ で *cwnd* を線形的に増やしている．*cwnd* を穏便に増やすことで，packet の送信量を増えたルータのキューの空きに合わせている．

2.4.2　TCP Reno

TCP Reno は，*cwnd* を以下の式 (2.1) のように変更する [62].

$$\begin{aligned} &\text{Slow-start phase (if } cwnd < ssthresh) \\ &\qquad cwnd = cwnd + 1 \\ &\text{Congestion avoidance phase (if } cwnd \geq ssthresh) \\ &\qquad cwnd = cwnd + \frac{1}{cwnd} \end{aligned} \tag{2.1}$$

TCP Reno は，古典的 TCP と同じく slow-start と congestion avoidance の 2 つの phase を持ち，phase 選択の基準が *ssthresh* である．*ssthresh* の値は動的に変動し，その目的はロスが起こる直前の *cwnd* を記憶するためである．

輻輳が起こると *ssthesh* は式 (2.2) によって設定される．

$$ssthresh = max(cwnd/2, 2) \tag{2.2}$$

Slow-start は基本的に新しい TCP コネクションが生成された場合や RTO によってロスが検知された場合に実行される．TCP コネクションが生成されると式 (2.1) の slow-start により，*cwnd* が RTT ごとに 2 倍ずつ増え (指数増加)，いずれネッ トワークの帯域を超え輻輳が発生する．TCP Reno はロスの検知には duplicated ACK(受信機が Sequence 番号が正しく ないパケットを受信した場合，最後に正常に受信したパケットの sequence 番号を繰り返し ACK として送る) を受信すると duplicated ACK に値するパケットを再送した後，*cwnd* を直前の半分に設定し，Good ACK(再送パケットの ACK) が届くまで duplicated ACK が届くたびに *cwnd* を 1 ずつ増やす．Good ACK が届くと *ssthresh* で *cwnd* を設定する．これを fast retransmission と言い，duplicated ACK を 3 回受信するまで続ける．Fast retransmission が RTO によって失敗したとき，*cwnd* を 1 に設定し，slow-start phase に入る．

2.4.3　TCP Vegas

TCP Reno はロスによって輻輳を検知するため，ロスが起こるまで *cwnd* を増やし，ロスの後は *cwnd* を必要以上に減らすため，スループットが低下する問題があった．TCP Vegas の congestion

avoidance phase では，RTT を観測して式 (2.3) で定義した Diff で actual スループットと expected スループットの 2 つのスループットを比較し，ネットワークの混み具合を判断し，$cwnd$ の調整を行う．

$$\begin{aligned} \text{Diff} &= \text{Expected Throughput} - \text{Actual Throughput} \\ &= \frac{cwnd}{BaseRTT} - \frac{cwnd}{RTT} \end{aligned} \tag{2.3}$$

BaseRTT は受信したパケットの RTT の最小値で，RTT は最後に受信したパケットの RTT を表す．Actual Throughput は今観測したスループット意味し，輻輳していなければ BaseRTT のままだと仮定すると Expected Throughput は最大スループットを表す．輻輳で RTT が大きくなっただけなら，Expected Throughput < Actual Throughput で判断できる．したがって，Diff は正あるいはゼロの値を持ち，式 (2.4) によって $cwnd$ を更新する [62]

$$\begin{aligned} &\text{if } Diff < \frac{\alpha}{BaseRTT} \\ &\quad cwnd = cwnd + 1 \\ &\text{if } \frac{\alpha}{BaseRTT} < Diff < \frac{\beta}{BaseRTT} \\ &\quad cwnd = cwnd \\ &\text{if } \frac{\beta}{BaseRTT} < Diff \\ &\quad cwnd = cwnd - 1 \end{aligned} \tag{2.4}$$

2 つの threshold である α と β は定数であり，$\alpha < \beta$ の関係を持つ．この手法が正常に動作すると，ロスが発生しない一定の $cwnd$ 値を持ち続けることが可能になる．実際の RTT の場合はキューによる遅延も含めているため，α 個までキューに入っても輻輳と見なさないとする．BaseRTT での計測ではキューに入っているパケットはない前提なので，$cwnd - \alpha$ のときのスループットと評価する．したがって，$(cwnd - \alpha)/BaseRTT < cwnd/RTT$ が輻輳していない条件となる．これが式 (2.4) の意味である．逆にキューに β 個以上入ったら輻輳と見なすとすると，$(cwnd - \beta)/BaseRTT > cwnd/RTT$ がその条件となる．Vegas の提案者によると α と β の最小値は 1 と 3 にしている．α を 1 にした理由はネットワーク経路に一つのパケットがキューイングされている場合は輻輳と見なさず，ネットワークの使用効率を上げるためである．β が 3 である理由は，$\beta - \alpha$ を 2 することで運用に余裕を与えて，頻繁な増減によるスループットの振動を防ぐためである [63].

また，slow-start phase での $cwnd$ の増加率は TCP Reno の半分で，次式によって更新される．

$$cwnd = cwnd + 1/2 \tag{2.5}$$

2.5　RED

最後に，ルータのキューの管理により輻輳を制御する手法である Random Early Detection(RED) について説明する．まず，背景として，シンプルなルータのキューの管理手法である tail drop を簡略に説明し，その問題点について記述した後，RED の仕組みについて説明する．

2.5.1　Tail Drop

ルータなどネットワーク機器のキュー管理に使われる一番シンプルな手法が tail drop algorithm である．例えば一台のルータを複数の TCP セッションが共有している場合，ルータが処理し転送

できる以上の packet が届いてキューが一定長以上になったら，tail drop algorithm では新しく届く packet を受け入れない（以降 drop と呼ぶ）．Tail drop algorithm で packet が drop された TCP セッションは，slow-start phase に入り，$cwnd$ を小さくするため，ルータへの負荷は軽くなる．しかし，多数の TCP セッションがルータを共有していると，ルータのキューから全ての TCP 送信機の TCP packet を drop し始めると，すべての TCP 送信機が同時に slow-start phase に入ることになる．この状態を global synchronization または TCP synchronization と呼ぶ．TCP synchronization 状態では，ネットワークの帯域幅を十分利用できなくなる問題がある．また，無差別に packet を削除するため，特定の TCP セッションからバースト的にトラヒック発生されキューがいっぱいになったとしても，すべての TCP セッションの packet が削除されるため，発生するトラヒック量が少ない TCP セッションからは不公平な処理になる．

2.5.2　Random Early Detection

RED(Random Early Detection) の目的は，ルータのキューに入ってくる新しい packet をランダムに drop させることで TCP synchronization を防ぐことにある．図 2.17 で示すように，新しい packet をどのくらい drop するかの確率（P_{RED}）は，ルータのキューのサイズ（以下 $avgQ$）と minimum queue threshold(以下 $minQ$) や maximum queue threshold(以下 $maxQ$) との関係て決定される．まず図 2.17 の (1) と (6) で示すように，$avgQ$ が $minQ$ より小さい場合は，新しい packet をキューに入れ，$avgQ$ が $maxQ$ より大きい場合は図 2.17 の (2) と (5) のように廃棄する．図 2.17 の (3) から (6) のように，$avgQ$ が $minQ$ と $maxQ$ の間に位置すると，P_{RED} を計算し，drop するかキューに入れるかを判断する．

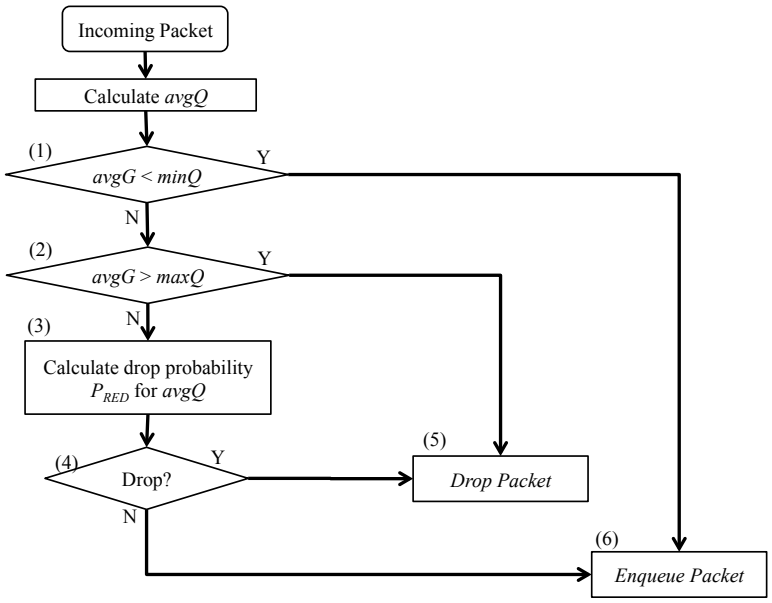


図 2.17: Random Early Detection の仕組み

P_{RED} の計算に使われるパラメータは，$avgQ$ の $minQ$ や $maxQ$，および $avgQ$ が $maxQ$ の場合の drop する確率 $maxP$ があり，図 2.18 で示すように P_{RED} は $avgQ$ に比例する．$avgQ$ が $minQ$ と $maxQ$ の間に位置する場合，P_{RED} の確率で packet が drop される．Packet を多く送る TCP

送信機ほど drop される packet の数が多くなるため、slow-start phase になる確率が大きい．そのため，キューのサイズに比例して無差別に drop する tail drop algorithm と比べ，トラヒック発生量が異なる複数の TCP 送信機が共存する環境でも公平な輻輳制御が実現できる [67].

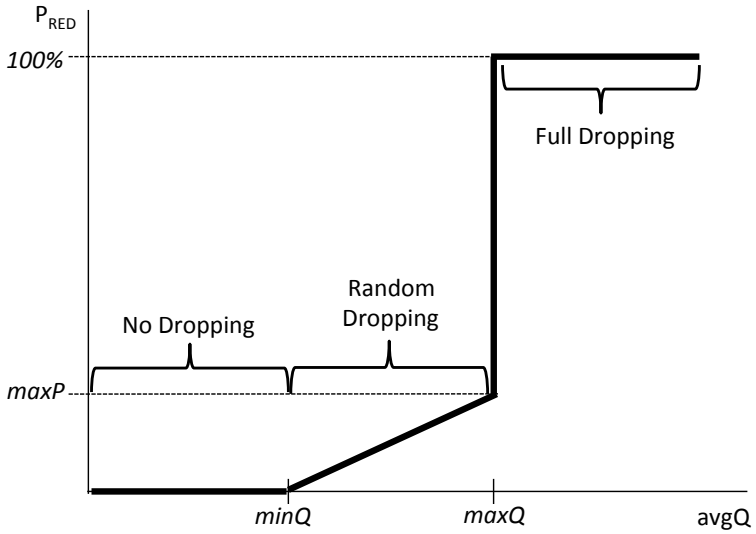


図 2.18: P_{RED} による packet drop [68]

2.6　まとめ

ネットワーク管理技術は，大規模なヘテロジニアスネットワークを管理するために，管理システムの各レイヤごとに標準化が進み，自動化と中央集中型への移行が進んでいる．Unix 上のパソコンとユーザのインターフェイスから進化した管理コマンドである CLI はコマンドと結果の標準化がなされずメーカごとのネットワーク機器の使い方を習得する必要があり，ネットワーク管理の管理コストの上昇を招いていた．SNMP により，この問題が一部解決された．SNMP は NMS，Agent，SNMP プロトコルそして管理情報で構成され，NMS はネットワーク管理者の端末，Agent は管理対象，管理情報は Agent が管理するネットワーク機器に関する情報，SNMP プロトコルは NMS と Agent の間の通信プロトコルである．NMS と Agent 間の通信プロトコルや管理情報を標準として定めことにより，複数のメーカのネットワーク機器が共存する環境でも一つの NMS で管理が可能になり，自動化も可能になった．そして，プロキシを利用することで Tambourine や OpenView などは SNMP を利用した Web からの大規模なネットワーク管理を可能にした．さらに，NETCONF は，モニタリングにとどまっていた SNMP と異なり，ネットワーク機器の設定も可能にし，SDN あるいは Openflow といった新しいネットワーク・アーキテクチャのネットワーク管理へ応用できるようになった．

近年，Openflow と SDN が産学で開発が進められるに連れ，ネットワーク管理の自動化と中央集中化が一層加速されている．SDN はネットワーク機器からデータを転送するルールを決める Control Plane を分離し，中央のコントローラへ集中させることでネットワーク管理を容易にしている．しかし，SDN のような中央集中型管理は，中央のプロキシやコントローラへの依存性が高くなるため，擬似管理トラヒックを利用した DDoS により中央管理サーバが攻撃される場合，単一障害点（Single Point of Failure, SPOF）になる危険性があり，また SNMP のプロキシである Tambourine や OpenView も同じ問題を持っている．提案手法は，中央集中型管理の単一障害点の解決を指向している．このため，関連する技術として Web cache と TCP の Flow Control や RED を説明した．Web cache は主に browser である client, proxy, および server 上に配置される．さらに Proxy 型キャッシュはキャッシュの位置によって，forward cache や reverse cache に分類され，提案対象の Tambourine は reverse cache に該当する．次に，関連するトラヒック制御方式である TCP には，大きく分けてロスベース方式と遅延ベース方式の二種類があり，ロスベース方式と遅延ベース方式の代表的例である TCP Reno と TCP Vegas について説明した．最後に関連技術である RED を紹介するため，ルータのキューの管理による輻輳制御手法として一番シンプルな方式である tail drop algorithm を説明し，その公平性問題解決技術として RED について説明した．

第3章

設計条件

3.1　はじめに

本章では，2.2.2 章で紹介した Tambourine のようなプロキシにより，大規模かつ中央集中化しているネットワーク管理に SNMP のような既存のネットワーク管理手法を共存させるために必要な制約条件を述べる．そのため，まず，Tambourine の各通信エンティティの目的と要求を定義し，各エンティティのニーズを最大に満足しながら，大規模なネットワーク管理の目的を達成するための設計条件を４つの項目にまとめる．各条件は本研究の課題となり，次の章での評価項目として使うことにする．

3.2　主たる通信エンティティの目的と要求

SNMP プロトコル使っての大規模なネットワーク管理を抽象化すると図 3.1 のように示すことができる．主たる通信エンティティの目的と要求，ならびに分析時の想定条件を明確にすると以下になる．

まず，通信エンティティの目的と要求について述べる．NMS_i は Agent が管理する任意の OID_i の MIB 情報 (以降 OID_i , $1 \leq i \leq M$, M は OID の総数とする) をモニタするのが目的で，OID_i が動的に更新される場合にはなるべく最新の値をモニタリングしたい．一方，Agent は SNMP 機器の情報を収集し NMS に返す目的を持っているが，SNMP 機器の本来の作業に支障がないよう，NMS からのモニタリングによる CPU 使用率は一定値 (TH) 未満に抑えたい．Tambourine は，Agent からモニタ情報を得て NMS へ中継する際，NMS と Agent の要求が相反するため，キャッシュで両者のバランスを取る．しかし，Tambourine が処理上のボトルネックになることは避けなければならない．以下では，$cache_i$ で OID_i の MIB 情報のキャッシュを示すことにする．

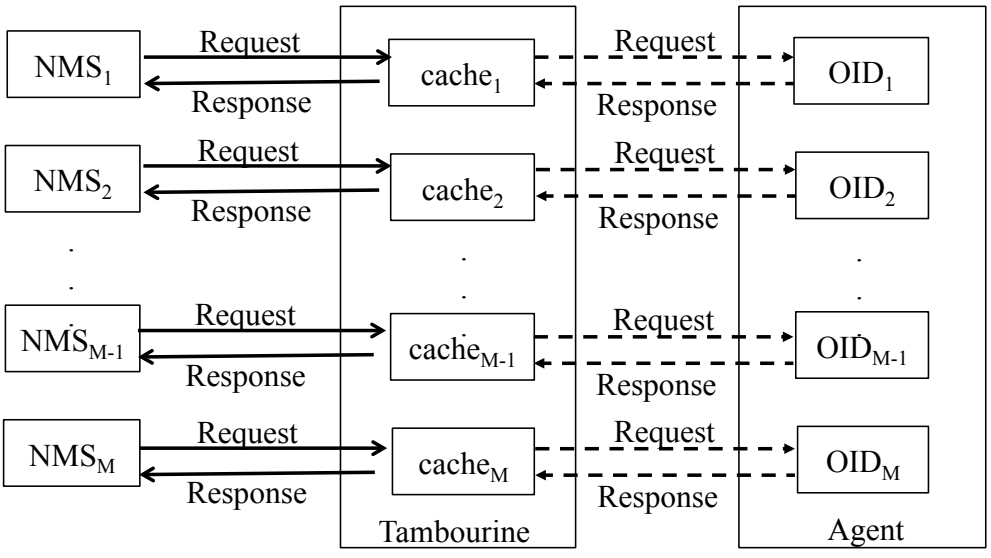


図 3.1: 大規模ネットワーク管理の通信エンティティ

想定条件としては，Agent の CPU 使用率が TH 未満ではすべての SNMP GetRequest を処理できるとする．この条件では Tambourine から Agent への SNMP GetRequest の数は Agent から

Tambourine への SNMP GetRespons の数と同じになるため，以下では SNMP メッセージ数と区別せずに言及することもある．

3.3 条件 1：管理トラヒックに起因する Agent の CPU 使用率増加を一定値 (TH) 以下に保つこと

ルータ/スイッチの CPU は IP パケットのオプションヘッダの処理，ルーティング情報の交換等に優先的に向けられるべきものであり，NMS からのネットワーク管理に伴う負荷は最小にすべきである．このため，管理トラヒックによる Agent の CPU 使用率増加に閾値 (TH) を設定し，TH 以上の負荷がかからないようキャッシュの TTL を制御し，管理トラヒックを利用した DDoS 攻撃に対処する必要がある．

3.4 条件 2：同一リクエスト間隔の NMS 間で情報取得の公平性が実現されていること

図 3.1 で示すように，Agent 上の OID_1 と OID_2 に対する NMS_1 と NMS_2 からの HTTP Get request が同一間隔でそれぞれ Tambourine の $cache_1$ と $cache_2$ に届いたとする．SNMP ratio を（一秒間に Tambourine が Agent へ送信した SNMP GetRequest の数）/（一秒間に Tambourine が NMS から受信した HTTP Get request の数）と定義すると，NMS_1 と NMS_2 間は $cache_1$ と $cache_2$ が同じ SNMP ratio を持たない限り公平とは言えない．ここで，SNMP モニタリングの観点から考えると，各 OID のモニタリングの公平性が重要である．従って，NMS_1 や NMS_2 は，１台の NMS ではなく，対応する OID_1 と OID_2 にアクセスする NMS の集合と考えている．条件 2 では NMS_1 と NMS_2 からの HTTP Get request の間隔が同じであり，NMS_1 と NMS_2 の SNMP ratio の比較は SNMP GetRequest 数の比較に帰着するため定量的な公平性の尺度として，SNMP メッセージ数に Jain ’s Fairness index を使って，cache 間の SNMP ratio のバラツキを評価する．

TCP では，同じ帯域を共有しているノードの数を M，i 番目のノードのスループットを x_i とした場合，式 (3.1) の Jain’s fairness index $J(x_1, x_2, \ldots, x_M)$ (以下 J) [69] を用いて公平性の評価を行い，J が 1 に近い値になる程公平と判断する．本研究ではスループットの代わりに，式 (3.1) の x_i を各 $cache_i$ が実験の最後の 10,000 秒の間に Agent から受け取った SNMP メッセージの数と定義し公平性を計算した．

$$J(x_1, x_2, \ldots, x_M) = \frac{\sum_{i=1}^{M} x_i}{M \sum_{i=1}^{M} x_i^2} \tag{3.1}$$

しかし，5 節で示すように，同一リクエスト間隔の NMS 間で最適なパラメータを探す際，J は値の変化量が小さく不便であったため，平均値に対する標準偏差の相対的大きさである変動係数 (R) で評価することにした．ここで，$Ave(x_i)$ は 10,000 秒の間の x_i の平均，$Std(x_i)$ をその標準偏差である．式 (3.2) に示す変動係数 (R) が 0 に近いほど，NMS が Agent から取得したレスポンスの数が cache 間でバラツキが相対的に小さく公平であると判断できる．さらに，条件 3 で論ずるように，変動係数 (R) はグループ間の公平性を論ずる際にも有効な尺度である．

$$R = \frac{Std(z_i)}{Ave(z_i)}, (1 \leq i \leq M) \tag{3.2}$$

3.5　条件 3：異なるリクエスト間隔の NMS グループ間でも情報取得の公平性が実現されていること

図 3.2 に示すように，$< NMS, cache, OID >$ の三つ組を同一の HTTP Get request 間隔 m 毎にまとめてグループを形成させる．ここで，NMS は OID にアクセスする NMS の集合である．たとえば，図 3.2 は，$Group_1$ と $Group_2$ にグループ分け構成された場合を示している．$Group_1$ の HTTP Get request 間隔 m が $Group_2$ に比べて短くなると $Group_1$ の SNMP GetRequest だけで Agent の CPU 使用率が TH に達し，Agent を独占する可能性がある．この状況では，m が長い $Group_2$ は TTL を小さくすることが困難となる．このような場合，$Group_1$ にペナルティを与え，$Group_1$ に属す cache の TTL を長くし， $Group_1$ の SNMP GetRequest 数を減らすように制御すべきである．

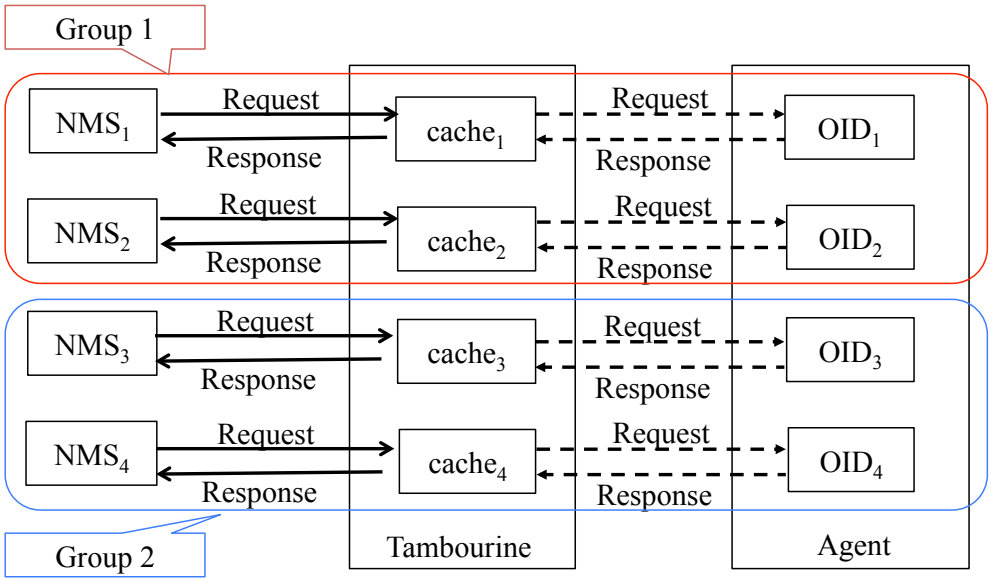


図 3.2: 異なる HTTP Get request 間隔 m を持つ NMS グループ間での情報取得の公平性

HTTP リクエスト間隔 m の大小は，変動係数 (R) の分母が示すリクエスト数の平均の小大に対応する．変動係数 (R) は標準偏差をその相対値として見たものである．このため，変動係数 (R) が同じであれば，リクエスト間隔 m に依らず，グループ間の相対的なばらつきが同じであることを意味する．このため，本研究では $Group_1$ と $Gruop_2$ の NMS の HTTP リクエスト間隔 m が異なっている場合，それぞれの Group の変動係数 (R) が同じならばグループ間で公平であると定義する．さらに，各グループ内の変動係数 (R) が 0 に近ければ条件 2 の意味で公平と言える．

3.6　条件 4：頻繁に問い合わせる NMS に最大限応じられること

最後に，頻繁に Agent に問い合わせたい NMS に対しては，条件 1 を満足する範囲内で最大限応じなければならない．大規模ネットワーク管理では異なる OID に対し，複数の NMS から Agent に問い合わせを行うことが多い．NMS からのモニタリングにより Agent が過負荷状態になった場合，Tambourine はキャッシュの TTL を長くし，その NMS からのリクエストの転送を制限する．

この結果，HTTP Get request に対して OID 情報の取得遅れ (以下鮮度，Fr) が生ずる．図 5.9 で示すように，Agent 内の任意の OID_i $(1 \leq i \leq M$，M は OID の総数) に Tambourine の $cache_i$ が SNMP GetRequest で測定時間内に N_i 回アクセスしたものとする．k 番目の SNMP GetRequest に対して SNMP GetResponse を取得した時刻を $t_{i,k}$ とし，$t_{i,k}$ から $t_{i,k+1}$ までに NMS_i から $cache_i$ へ送信した HTTP Get request の数を $req(i,k)$，その中の j 番目の HTTP Get request に対し $cache_i$ が HTTP response を返した時刻を $c_{i,k,j}$ とする．HTTP response を返した時刻と SNMP GetResponse を取得した時刻との差を $d(i,k,j) = c_{i,k,j} - t_{i,k}$ とすると，その平均値である鮮度 (Fr) は式 (3.3) で計算できる．単位は秒である．Tambourine は条件 1 を満足しつつ，最小の Fr で NMS へレスポンスを返さなければならない．

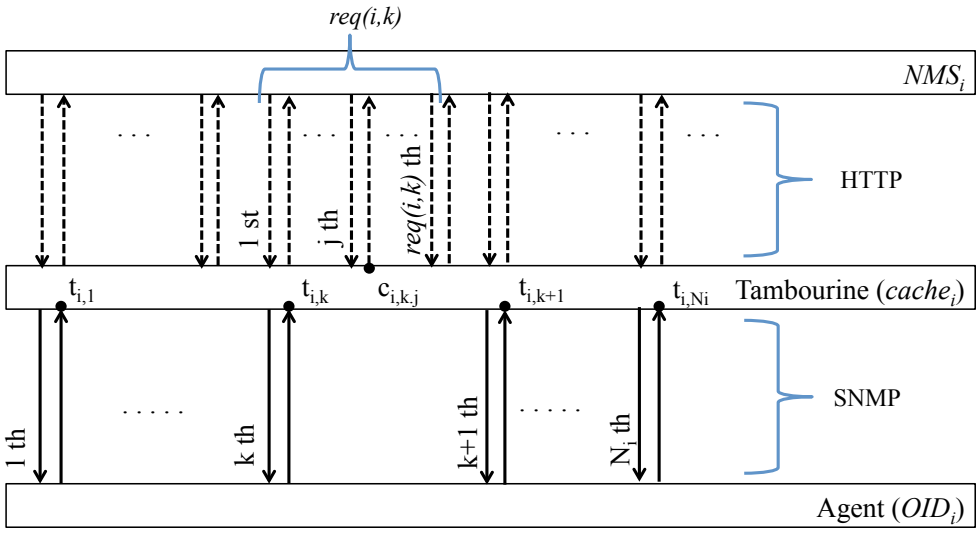


図 3.3: OID_i に関係する HTTP/SNMP メッセージ

$$Fr = \sum_{i=1}^{M} \sum_{k=1}^{N_i} \sum_{j=1}^{req(i,k)} d(i,k,j) / \sum_{i=1}^{M} \sum_{k=1}^{N_i} req(i,k) \tag{3.3}$$

3.7　まとめ

本章では，Tambourine により大規模かつ中央集中管理に SNMP プロトコルを活用するため必要な通信エンティティの目的と制約条件を説明した．まず，各通信エンティティに関して，NMS は被管理対象である Agent の状態をなるべく高い粒度でモニタリングする，Agent は SNMP 機器

本来の作業に忠実しながら NMS のモニタリング要求に応じる，Tambourine はキャッシュで NMS と Agent の要求のバランスを取る目的を持っていと分析した．そして，この目的を満足するための条件を次のようにまとめた．まず条件 1 として，NMS から管理トラヒックが出され Tambourine を介して Agent へ届く際，Agent の CPU 使用率増加が一定値 (TH) 以下に保つこととした．TH 以上の負荷がかからないよう Tambourine がキャッシュの TTL を制御することで，NMS と Agent の矛盾するニーズに応じる．条件 2 は，同一リクエスト間隔の NMS 間で情報取得の公平性が実現されていることであり，標準偏差をその相対値として見た変動係数 R を評価尺度として設け，同じ粒度で管理トラヒックを出す NMS 間での公平性を確保する．そして，条件 3 は異なるリクエスト間隔の NMS グループの間でも情報取得の公平性が実現されていることである．この条件での評価尺度としても変動係数 (R) が使え，R が同じであれば，リクエスト間隔に依らず，NMS グループ間の相対的なばらつきが同じであるため，条件 3 の評価にも使うことができる．最後に，条件 4 として，頻繁な問い合わせる NMS に最大限応じられることを上げた．Tambourine のキャッシュにより Agent の状態把握に遅れが発生するため，NMS から Tambourine への HTTP Get request に対して Agent からの OID 情報の取得遅れが生ずる．本論文ではその取得遅れの平均値を鮮度 (Fr) として定義し評価指標に使うことにした．次章では，以上の設計条件を課題として満足するための提案を説明する．

第4章

大規模ネットワーク管理の提案手法

4.1　はじめに

本章では，前章の設計条件を満足しつつ，SNMP プロトコルによる大規模ネットワーク管理を可能にする手法を提案する．提案の前にまず，ネットワーク機器の SNMP の処理に起因する 1 分間の平均 CPU 使用率（以下 CPU 使用率）が 1 秒間の SNMP リクエストの数に依存することを示し，その関係式を実験により導出する．次に，この近似式を基に機器の SNMP 処理に起因する CPU 使用率増加を推定し SNMP リクエスト間隔を TCP Reno に類似したフィードバックで制御することにより，SNMP 処理に起因する CPU 使用率を定められた閾値以下に抑える手法を提案する．最後に，ルータ等の輻輳制御に用いられている RED(Random Early Detection) に似た手法を TCP ではエッジ・デバイスに相当する Tambourine に導入することにより，NMS 間のモニタリングの公平性を実現し，その精度を保証できることを示す．

4.2　SNMP トラヒックと SNMP 機器の CPU 使用率

SNMP GetRequest による Agent の CPU 使用率への寄与を確認するため，下記の手順で実験を行った．1 台の PC (CPU：Intel(R) Xeon(R) CPU E3-1240 v3 @ 3.40GHz，16GB，OS：Ubuntu 12.04.5 LTS) を Alaxala3640s と Catalyst 2950 に接続し， sysUpTimeInstance の MIB に対して 1 秒あたり N 個 ($1 \leq N \leq 900$, Catalyst 2950 の場合は $1 \leq N \leq 200$) の SNMP GetRequest を等間隔で 1 分間送信した．1 分後，Alaxala3640s と Catalyst 2950 の過去 1 分間の CPU 使用率を PC から SNMP GetRequest を出して取得し，一秒間のリクエスト数と CPU 使用率との関係を求めた．Catalyst 2950 の場合 N の上限が 200 個である理由は，1 秒当たり 200 個以上 SNMP GetRequest を送ると，過負荷状態になり，CPU 使用率を測定するための SNMP GetRequest を処理できなかったからである．

この実験を各 N に対して 30 回繰り返し，平均と標準偏差を求めプロットしたものが図 4.1 の (a) である．Alaxla3640s の場合，1 秒間の SNMP GetRequest 数 N(リクエスト/秒) と CPU 使用率 (%) に関しては，式 (4.1) で近似（誤差 3.43%）でき，図 4.1 の (a) では $f(N)$ で示している．

$$f(x) = 100(1 - exp(\frac{-N}{350.808})) \tag{4.1}$$

しかし，後述する評価実験では CPU 使用率を 20%以下に抑えることを目標としているため，この区間では線形近似で十分である．図 4.1 の (b) に示すように，この区間での Alaxla3640s の CPU 使用率を $g(N)$ で表した時，式 (4.2) で近似（誤差 2.34%）できる．Catalyst 2950 の場合も $1 \leq N \leq 150$ で線形近似できることが図から明らかである．

$$g(x) = 0.169x + 6.236 \tag{4.2}$$

4.3　TCP Reno 類似型 SNMP 機器負荷制御手法

3 章の条件を満足しながら，大規模ネットワーク管理の実現を目指す．既存提案 [15] は,「アクセス時の値が以前と変化している」時に TTL を 1/2 にし,「変わっていない」時に 2 倍にすることにより，アクセス頻度を OID の性質に合わせるというのが骨子であった．SNMP 機器の過負荷も考慮して拡張すると,「アクセス時の値が以前と変化」した場合,「SNMP 機器が過負荷でない」時に

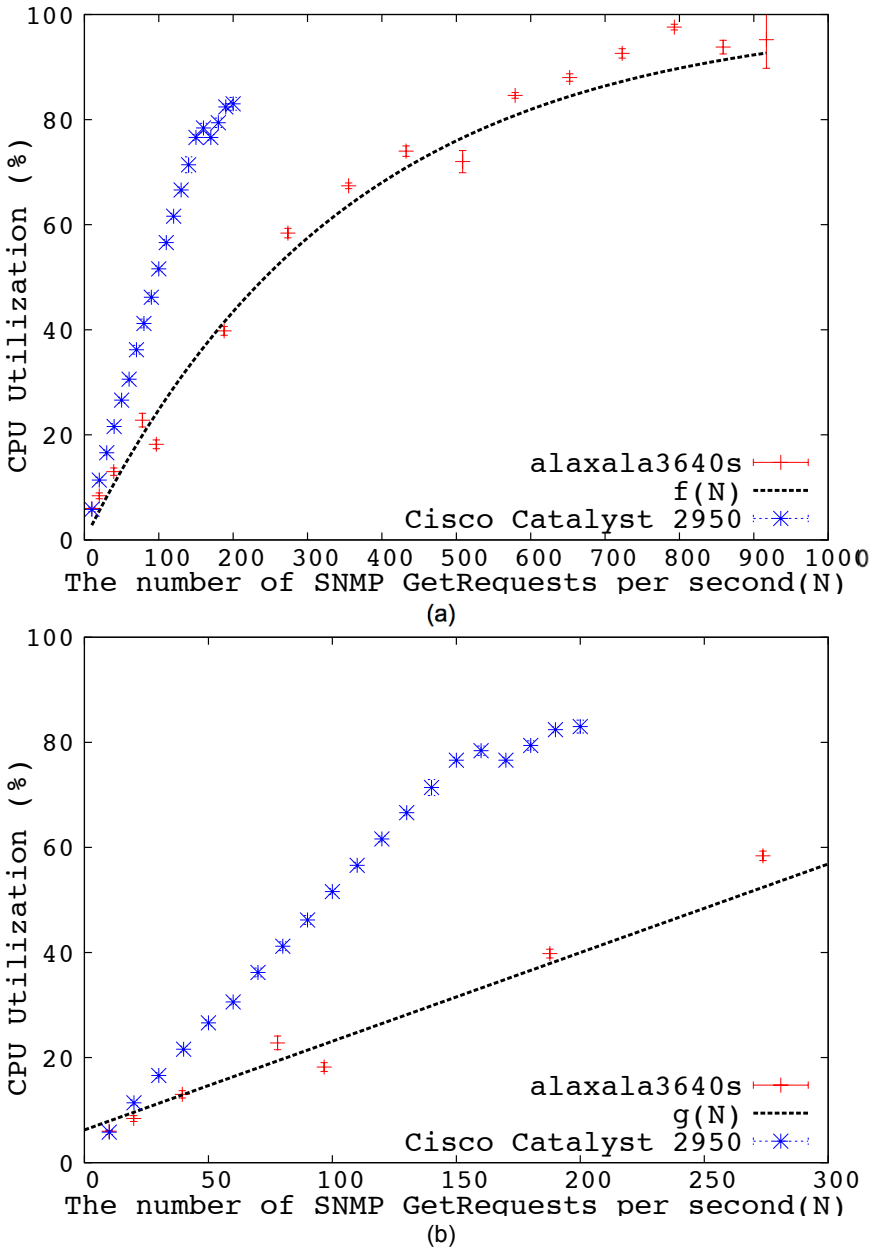


図 4.1: CPU 使用率と 1 秒間の SNMP GetRequest 数

TTL を 1/2,「過負荷」の時に 2 倍にすることにより，制御することになる．これは，2.4 節で説明したように TCP Reno が輻輳になるまで，RTT ごとに輻輳ウィンドウを倍にし，輻輳検出すると 1/2 にして輻輳でなかった RTT 前のウィンドウにする手順と類似している．このため，SNMP 機器の負荷制御にも TCP Reno 類似の手法が有効と考えられる．

TCP Reno では，3 回の duplicate ACK の受信と RTO　(Retransmission Timeout) によって輻輳を制御するが，Tambourine は SNMP 機器への総リクエスト数 $N = \sum_{i=1}^{M} 1/TTL_i$(リクエスト/秒) を把握できるため，式 (4.1) を使って SNMP トラヒックによるネットワーク機器の CPU 使用率への寄与を計算できる．図 4.2 には，SNMP GetRequest によるネットワーク機器の CPU 使用率の上昇が閾値 (TH) 以上の場合を輻輳と定義し，TCP Reno 類似型 SNMP 機器負荷制御手法を示した．

まず，本提案に使われるパラメータの定義を先に行う．$cwnd_i$ は OID_i の値に対する Tambourine の $cache_i$ の TTL_i の逆数である．$ssthresh_i$ は輻輳検出時の $cwnd_i$ 値の半分である．$RTO\ timer_i$ は Retransmission timer である．$RTO\ timer_i$，TTL の初期値 $ttl\_default$，および TTL の最小値 ttl_min と最大値 ttl_max は定数で実験のスタート前に予め設定し，設定値については 5 章で説明する．最後に CON_i は頻繁に問い合わせされる OID_i にペナルティを与えるための変数であり，詳細は図 4.3 の説明で行う．

図 4.2 の (1) では，OID_i に対し初めて SNMP GetRequest を送り取得した MIB 値で $cache_i$ を更新した後，　各パラメータの初期化を行う．HTTP Get request が到着したとき (図 4.2 の (2))，TTL_i が timeout してない場合は $cache_i$ の値を HTTP response として返す (図 4.2 の (10))．もし timeout していた場合は CPU 使用率を計算し (図 4.2 の (4))，TH と比較して輻輳であるか調べる (図 4.2 の (5))．輻輳でないと判断した場合のみ Agent へ SNMP GetRequest を送り新しく得た SNMP GetResponse の値を HTTP response で返し，$cache_i$ を更新する (図 4.2 の (11))．更新により $cache_i$ が変わっているか確認し (図 4.2 の (12)) 変わっていなければ TTL_i を 2 倍にし (図 4.2 の (13))，もし変わっていれば，$cwnd_i$ と $ssthresh_i$ を比較し (図 4.2 の (16))，congestion avoidance phase(図 4.2 の (17)) または，slow start phase(図 4.2 の (18)) を行う．輻輳の場合は (6),(7),(8),(9) で構成される congestion phase に移る．パス (6)-(7) は slow start や congestion avoidance 状態で，初めて輻輳になった場合を示し，TTL_i を 2 倍にし，$RTO\ timer_i$ をスタートする．$RTO\ timer_i$ が timeout した後も輻輳が続く場合には，パス (8)-(9) により全てのパラメータを初期化する．

輻輳と判断した場合，TCP とは異なりキャッシュの値を返す (図 4.2 の (10)) ため fast retransmission は必要ない．また，輻輳の推定を 3 回の duplicate ACK で行う TCP と違って 1 回の輻輳で計算し congestion avoidance phase へ移行することができる．また，TCP は RTT を最初の $RTO\ timer$ として使っているが，本提案では NMS 間の公平性を実現するため決められた $RTO\ timer$ 値を用いる．$RTO\ timer$ と公平性に関する説明は 4.4 節で，$RTO\ timer$ の具体的な値については 5 章で詳しく説明する．以降，TCP Reno を応用したこの手法を単に Reno と呼ぶ．

4.4　Random Early Detection による公平性の実現

前節での提案では，管理トラヒックによる TH 以上の負荷の増加は避けられるものの，NMS からの問い合わせが頻繁に行われるキャッシュで SNMP 機器の CPU 使用が独占され，NMS 間の公平性が阻害される恐れがある．インターネットでは共存する TCP セッション間の公平性をルータの RED(Random Early Detection) により実現している [67]．RED はキューの長さが一定値以上になった場合，パケットをランダムに削除して各セッションがキュー中に占めるパケット数に比例

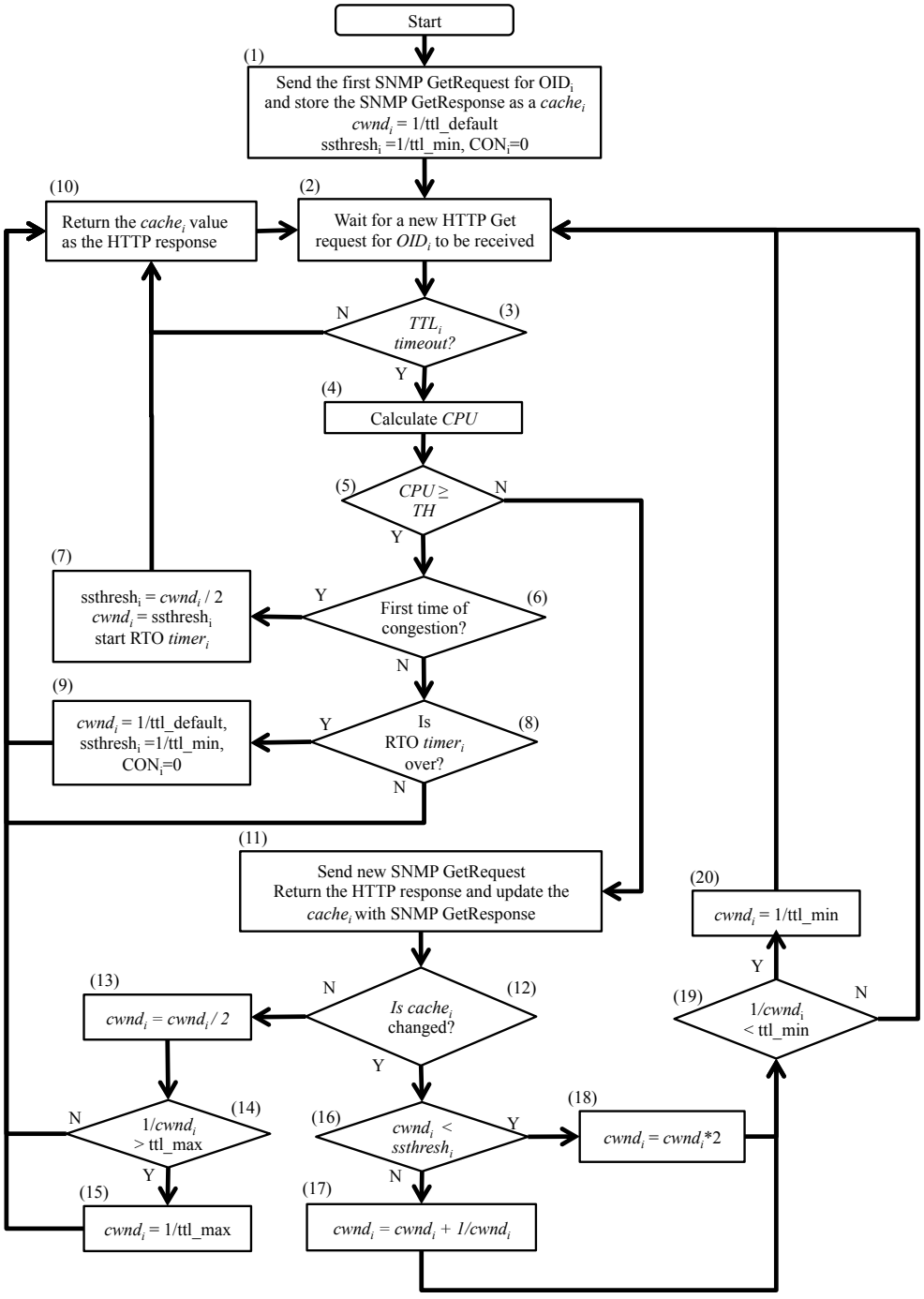


図 4.2: TCP Reno 類似提案アルゴリズムよる $cache_i$ の管理

してパケットロスが増えるように工夫したもので，TCP セッション間の公平性を上げるために使われている．本論文では，各 OID_i の TTL_i の逆数 ($w_i = 1/TTL_i$) がモニタリングのトラヒック量 (packets/sec) であることに着目し，TTL が短くトラヒック量の多い OID にペナルティを与えて NMS 間の公平性を上げる手法を導入する．以下に示すように，提案手法は RED に類似している．図 4.2 の (4) の CPU 使用率の計算をこの手法を用いて修正し，図 4.3 に示す．

図 4.3 の (1)-(2)-(3)-(4) では，式 (4.1) で計算した CPU 使用率 CPU が TH_{min} ($TH > TH_{min}$) と TH の間にある場合， 式 (4.3) に従い計算した drop probability (P_{RED}) で OID_i に対する HTTP Get request を drop させる．ここで，drop とは図 4.3 のパス (5)-(6)-(7)-(8)-(9) で SNMP 機器へ頻繁に問い合わせを行う OID_i のキャッシュに対しペナルティを与えることを意味する．

$$P_{RED} = \frac{CPU - TH_{min}}{TH - TH_{min}} \tag{4.3}$$

Drop の手順を以下に示す．まず図 4.3 の (5) で，仮想のキュー（長さ 1）を作成する．$OID_i (1 \leq i \leq M$, M は OID の総数) の $1/TTL_i = w_i = cwnd_i$ を昇順に 0 から並べ替え，式 (4.4) で正規化する．図 4.4 に仮想キュー上の ω_k の配置を示した．TTL が短く SNMP 機器に対して管理トラヒックを多く発生するキャッシュほど，大きな ω_k として 1 の近傍に割り当てられる．

$$\omega_k = w_k / \sum_{i=1}^{M} w_i \tag{4.4}$$

次に，図 4.3 の (6) で示すキャッシュ(以下 OID_s) の選択手法を説明する．まず，初期値 0 の輻輳カウンタ CON_s を用意する．また，k 番目の w_k までの合計 $SumOID(k)$ $(0 \leq SumOID(k) \leq 1)$ を式 (4.5) に従って計算する．

$$SumOID(k) = \sum_{j=1}^{k} \omega_j \tag{4.5}$$

RED の延長で考え，一様分布の確率変数 x $(0 \leq x \leq 1)$ を生成し，x が $SumOID(s-1)$ と $SumOID(s)$ の間の値になった場合，CON_s を一つ増加させる．ここで，一般に $s \neq i$ であるため，現在の処理対象の OID_i ではない OID_s にペナルティが課される．したがって，OID_i にアクセスした場合，たとえ輻輳していなくても $CON_i > 0$ であれば CON_i を 1 減算し，図 4.2 の (4) へは CPU の計算値として輻輳を示す TH を返すことにする．

OID_i の TTL_i が短く，SNMP 機器へ負荷を多くかけるキャッシュほど CON_i の値が大きくなり，図 4.2 の (4) へ CPU の計算値として TH を返す確率が高くなる．この結果，図 4.2 の (5) で輻輳だと判断し，*RTO timer* がスタートされる．したがって，RTO が timeout し，図 4.2 の (9) で示す初期化が行われる可能性が高くなるため，公平性が上がる．以降，本手法を RED ’と呼ぶ．

しかし RED ’では，5.3 節で示すように，NMS 間の取得レスポンスの公平性は期待するほど向上せず，輻輳時にドロップさせるリクエストの選択確率を大きい w_s に対しては線形以上に大きくし，逆に小さい w_s に対しては線形より小さくなるような非線形制御にしたほうが良いことが分かった．実装上は，$SumOID(s-1)$ や $SumOID(s)$ と一様分布の確率変数 x とを比較するのではなく，$0 \leq y = h(x) \leq 1$ と比較して選択することにした．ここで， $y = h(x)$ は，式 (4.6) に示すように，中心が $(a, 1-a)$ で $(1,1)$ と $(0,0)$ を通る円弧（$\alpha \geq 1$）である．$\alpha \to \infty$ では図 4.5 で示

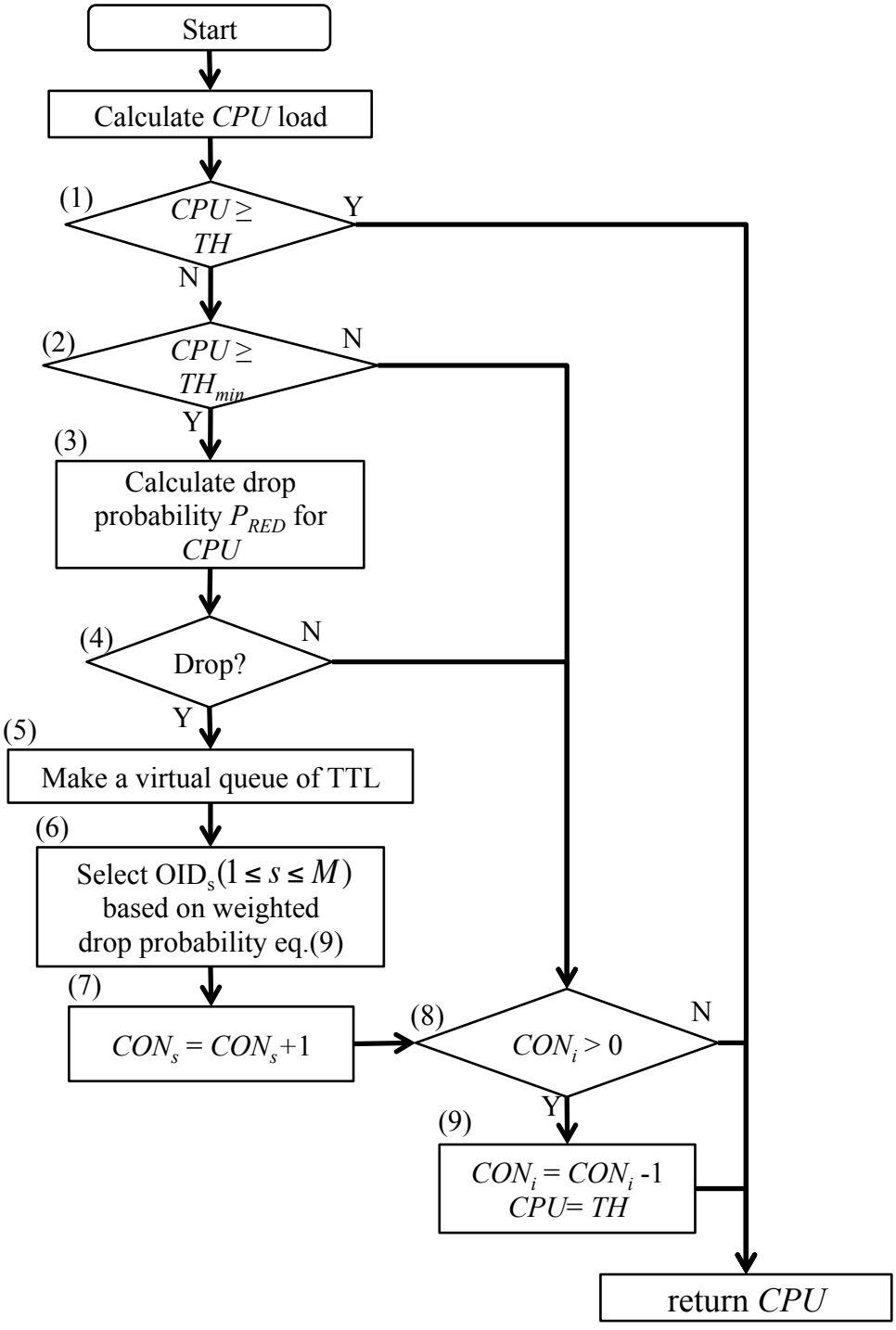


図 4.3: CPU 使用率計算手順

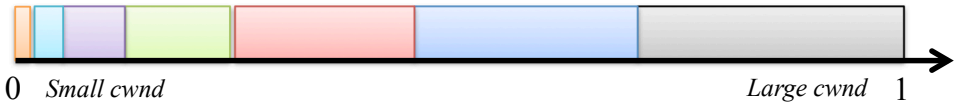


図 4.4: 仮想キュー上の ω_k の配置

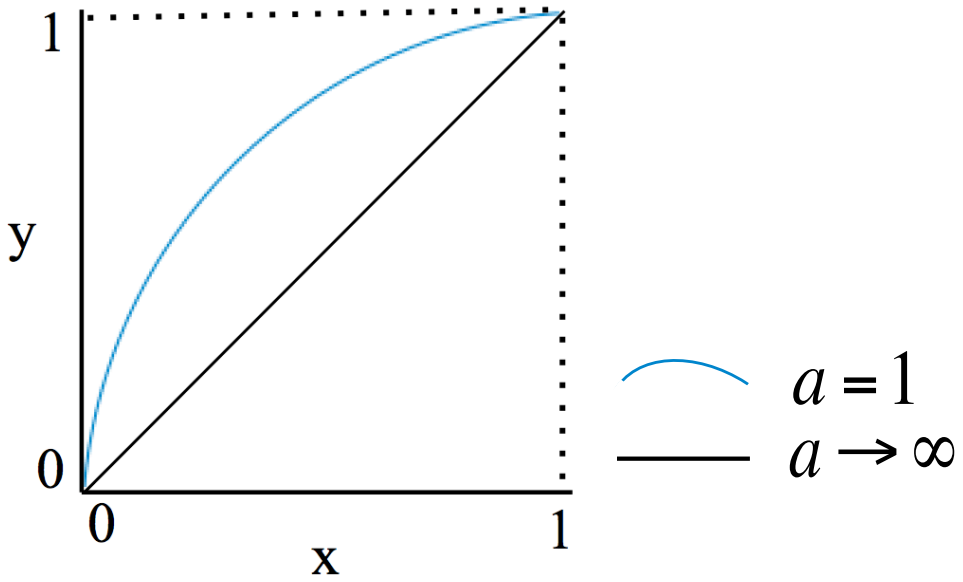


図 4.5: 確率変数の変換

す $(0,0)$ と $(1,1)$ を結ぶ直線になる．図 4.5 の曲線は式 (4.6) で $\alpha = 1$ での場合を示す．式 (4.7) にその確率密度関数 $p(y)$ を示す．以降 RED ’を修正した本手法を RED と呼ぶ．

$$y = h(x) = 1 - \alpha + \sqrt{(1-\alpha)^2 + \alpha^2 - (x-\alpha)^2} \tag{4.6}$$

$$p(y) = \frac{y - 1 + \alpha}{\sqrt{(1-\alpha)^2 + \alpha^2 - (y-1+\alpha)^2}} \tag{4.7}$$

4.5　まとめ

本章では，SNMP 機器で処理する SNMP リクエストの数と CPU 使用率との関係を示した後，前章での提案条件を満足するための Tambourine のキャッシュ管理手法を提案した．まず，SNMP リクエストに起源する CPU 使用率の上昇は，PC から Alaxala3640s と Catalyst 2950 に SNMP GetRequest を任意の一定間隔で送り，各機器の過去 1 分間の CPU 使用率を測定した．その結果，CPU 使用率が 1 秒間の SNMP リクエストの数に依存することが分かり，相関関係を近似式 (4.1) としてまとめた．次に，3.3 節の“ 条件 1：管理トラヒックに起因する Agent の CPU 使用率増加を一定値 (TH) 以下に保つこと ”を満足するため，この近似式 (4.1) を基に TCP Reno に類似した Tambourine のキャッシュ管理手法を提案した．既存の研究 [15] では，Tambourine のキャッシュの値と SNMP 機器からの新しい値に違いがあった場合，SNMP 機器の CPU 使用率を考慮せず，キャッシュの更新周期を半分にしていた．提案では，Tambourine が SNMP 機器へ送り出す SNMP リクエストの総数 N を把握しているため，近似式 (4.1) に N を代入して SNMP リクエストによる CPU 使用率の上昇分を計算できる．SNMP 機器の CPU 使用率の上昇が閾値 (TH) 以上の場合を輻輳と定義し，輻輳の場合は，Tambourine が NMS からの問い合わせに対し，キャッシュ値を返し，新しい SNMP リクエストを SNMP 機器に送らない手法で，SNMP 機器の SNMP リクエストによる CPU 使用率の上昇が TH を超えないようにした．最後に，3.4，3.5，3.6 節の条件 2，3，4 の NMS 間の公平性に関する条件を満足するため，SNMP 機器へ問い合わせを行うキャッシュに対し計算した CPU 使用率が TH_{min} ($TH > TH_{min}$) と TH の間にある場合， 式 (4.3) の drop probability によって drop させる RED に類似したキャッシュ管理手法を前述の提案に追加した．その際，リクエスト頻度の多い OID へのアクセスほどペナルティ（drop 確率）を増やすことになるが，リクエスト頻度に対して線形以上のペナルティを課すことが有効であることを示した．

第5章

実装と評価

5.1　はじめに

Alaxala3640s を Tambourine でモニタリングすると想定し，前章で示す方式を ns-2 [70] でシミュレーションし，提案が 3 章の条件を満たすことを示す．結果の説明の前に，全ての実験で共通する設定を説明する．実験構成は，図 3.1 と同じであり，図 5.1 に再掲した．Agent と Tambourine に該当するノードを各 1 台配置した．また，一般的なネットワーク管理環境ではひとつの NMS が複数の OID の管理を兼ねているが，ここでは 500 個（図 5.1 では M に値する）の OID を仮定した．500 個にした理由は，Alaxala3640s の全てのポートに 100Mbps のトラヒックを負荷として設定した状態で各 OID に対し 0.1 秒間隔で問い合わせをした結果，MIB 値が変化した OID の総数が 530 個であった実験結果からである．更に，各 OID にアクセスする NMS グループを各々一台の NMS で対応させた．

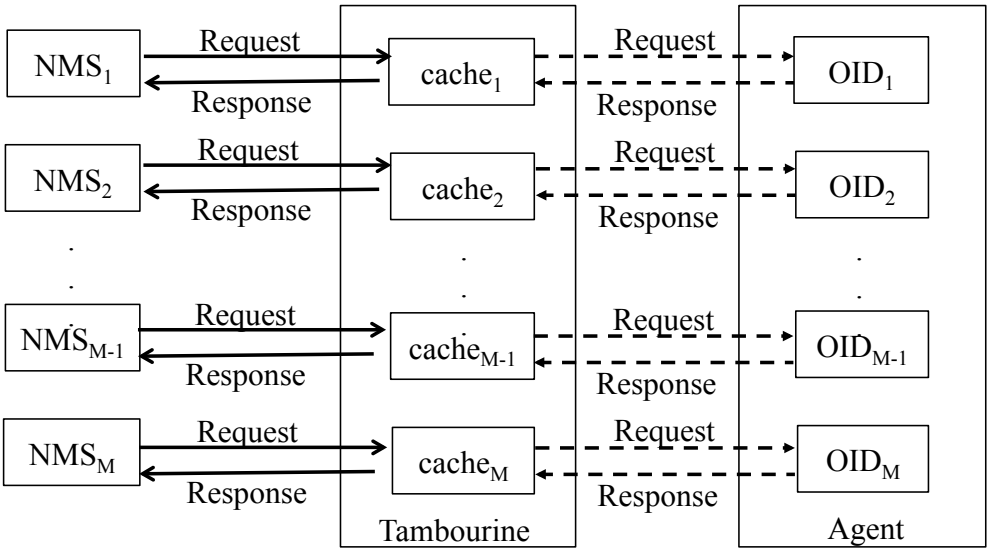


図 5.1: 実験構成

各 NMS の起動は 0 秒から 300 秒の間でランダムに行い，NMS からの HTTP Get request の送信間隔 m は $N(\overline{m}, \sigma^2)$ のガウス分布に従うとした．ここで $\overline{m}$ は 0.1 秒から 300 秒，$\sigma^2 = 0.001$ 秒^2 とした．分散に関しては，FCC の定めたインターネットサービスのジッタの品質目標が 20ms であること [71]，米国の ADSL 等のサービスの品質測定での遅延の標準偏差が 7.7 から 33.5ms であること [72]，サービス品質保証 (SLA) 型インターネットサービスのジッタ 0.05ms であることから [73]，企業向けインターネットサービスのジッタ実態を 1ms 程度と見積もり，分散 0.001 秒^2 を用いることにした．

実験での HTTP Get request 間隔 $\overline{m}$ は全て同一，もしくは数組の間隔から成る．SNMP プロトコルによるネットワーク管理では MRTG [74] のような一定送信間隔のモニタリングが多いため，実験では周期的モニタを想定した．TTL の初期値 ttl_default は MRTG のモニタリング間隔を参考し 5 分に，TTL の最小値 ttl_min は 0.1 秒，最大値 ttl_max は 1 時間に設定した．また，東京大学と沖縄大学のサーバ 間を 30 回の ping で測定した結果，平均伝送遅延が 37.867ms（標準偏差 0.297ms）であることから， NMS と Tambourine 間の遅延を 37ms に設定し，全国規模に分散した NMS の遅延の影響を考慮した．

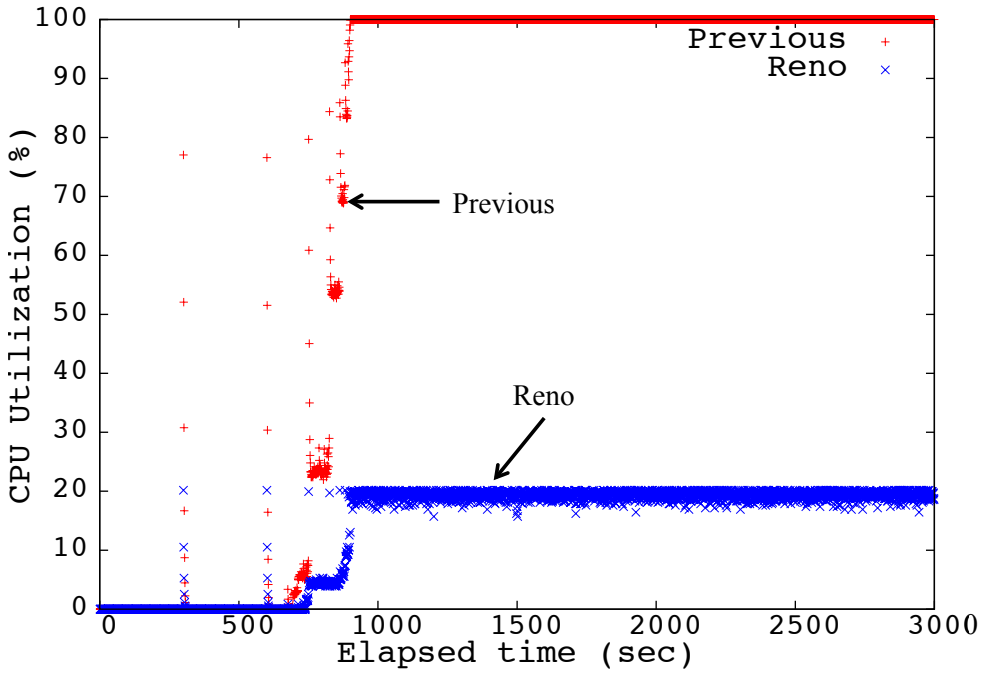


図 5.2: 既存手法 (Previous) と提案 (Reno) による CPU 使用率

評価対象にした輻輳制御手法は，前述の Reno，RED ¦ RED であり，TH=20%，TH_{min}=18%，また式 (4.6) は $\alpha = 1$ としている．ここで，モニタリングによる負荷増は，ルータの CPU の本来の処理に影響が出ない範囲内で運用しなければならず，モニタリングが軽負荷である目安が必要となる．この実験では，Cisco の CPU 使用率の下限しきい値通知の設定値が 20%である事例があることを鑑み [75]，20%を軽負荷の典型とみなして TH を選んだ．TH_{min}=18%である理由は，公平性と CPU 使用率の制御性能（ここで，CPU 使用率の制御性能 (Control Performance of CPU Usage) とは，TH と実際の CPU 使用率の差の絶対値の時間平均であり，これが小さいほど制御目標を正確に実現しているとみなすことができる.）を両方考慮し，$TH_{min} = 18\%$が最善だと判断した．詳しくは 5.4 節で説明する．

5.2　条件 1 の評価

図 5.2 に既存の手法 [15] と 4.3 節の Reno による CPU 使用率の測定結果を示した．シミュレーションの設定は，前節の設定で，全ての NMS からの HTTP Get request の間隔 $\overline{m}$ を 0.1 秒にしているため，平均 5,000 パケット/秒のトラヒックが Tambourine に向けて送信される．RTO は次節での評価で最適値であった 100 秒に設定し，Agent 上の CPU 使用率は式 (4.2) で計算した．その結果，既存の手法では 900 秒の近辺から CPU 使用率が 100%と輻輳するのに対して，Reno では TH で設定した 20%近傍に収束し，条件 1 を満足していることが分かる．

収束まで 900 秒近くかかった理由は，以下の通りである．第 1 に，図 4.1 の Alaxla3640s の結果から，Agent へ 100 リクエスト/秒でリクエストが集中すると CPU 使用率が 20%になる．これは，NMS の数が 500 台の場合は，1 つの NMS 当たり 0.2 リクエスト/秒に相当する．このときの，

Agent への SNMP GetRequest 間隔は 5 秒になる．第 2 に，Agent への SNMP GetRequest 間隔は Tambourine の TTL 値で決まり，Tambourine の TTL の初期値は 300 秒であり，アクセスされるたびに 1/2 になる．第 1 と第 2 の理由により 5 リクエスト/秒に至るまで約 600 秒かかること，さらに NMS の起動が 0 秒から 300 秒の間でランダムであることから，収束まで 900 秒近くかかっている．

5.3　条件 2 の評価

この節では，条件 2 に関する提案と公平性の側面から最適な RTO timer の設定値（以下 RTO）の選択と公平性指針としての R と J について評価する．

まず，最適な RTO の選択のためのシミュレーションの設定について述べる．NMS からの HTTP Get request 間隔 $\overline{m}$ を前節と同じ 0.1 秒に設定し，輻輳制御を RED にした場合は RTO を 0.1 秒から 2000 秒まで変え，RED ’と Reno の場合は RTO を 100 秒に固定して，3 万秒間シミュレーションを実行した．シミュレーションの最後の 10,000 秒間に Tambourine 上の $cache_i$ が Agent から受信する SNMP GetResponse の数 (以下 x_i) を計算し，シミュレーションが終わる瞬間の TTL_i も記録した．

図 5.3 に各 RTO に対する x_i の CDF (cumulative distribution function) を Reno，RED ’，RED に関して示す．CDF が Step 関数に近い程，各 cache が 10,000 秒間に Agent から得た HTTP response のバラツキが小さいことを意味する．この結果，RED の場合，RTO が 0.1 秒から 100 秒までは公平性が向上している．さらに，図の拡大部分の 500 秒と 1000 秒の例が示すように，100 秒以上では若干劣化することが分かった．このため，100 秒が最適と考えられる．しかし，4.4 節で説明したように RED ’や Reno は RTO として 100 秒を選択しても，良い公平性は得られない．

次に，シミュレーション終了時の TTL の CDF を図 5.4 に示す．図 5.3 と同じく，CDF が Step 関数に近い程，各 cache の終了時の TTL のバラツキが小さいことを意味する．この結果も図 5.3 と同じく，RTO が 100 秒の場合が最適な公平性が実現されていることが分かる．RED ’や Reno では，RTO を 100 秒にしても，公平性は改善されない．

次に公平性の尺度の選択について説明する．図 5.5 に異なる RTO に対する式 (3.1) の J と式 (3.2) の R の値を示した．3 節で説明したように，J が 1 に近い程そして R は 0 に近いほど，各 $cache_i$ の x_i のバラツキが小さく公平と判断できる．RTO が 0.1 秒から 8 秒に増加すると，J は 1 に，R は 0 に近づき，RTO が 100 秒を中心として，おおむね J は上に凸，R は下に凸の関数になっていることが分かる．この結果，RTO が 100 秒近傍で一番公平な状況になると判断できる．また，RTO が 200 秒から 1000 秒の間を見ると，R の方が RTO の変動に敏感に反応しているため，以下の議論では R を公平性の尺度として選択した．

5.4　最適な TH_{min} の選択

4.4 節で説明しているように，RED による輻輳制御では，CPU 使用率が TH_{min} ($TH > TH_{min}$) と TH の間にある場合，drop probability (P_{RED}) を式 (4.3) に従い計算し，drop するかを判断するため，TH_{min} が小さく，TH と TH_{min} の差が大きければ大きいほど，RED による輻輳制御（図 4.2 の (6),(7),(8),(9)）を経験する可能性が高くなり，NMS 間の公平性が高く（R が小さく）なる代わりに，収束時の CPU 使用率の制御性能が悪くなる．ここで，CPU 使用率の制御性能とは，HTTP Get request 受信時に計算した CPU 使用率 (図 4.2 の (4)) と TH との差の絶対値の最後の

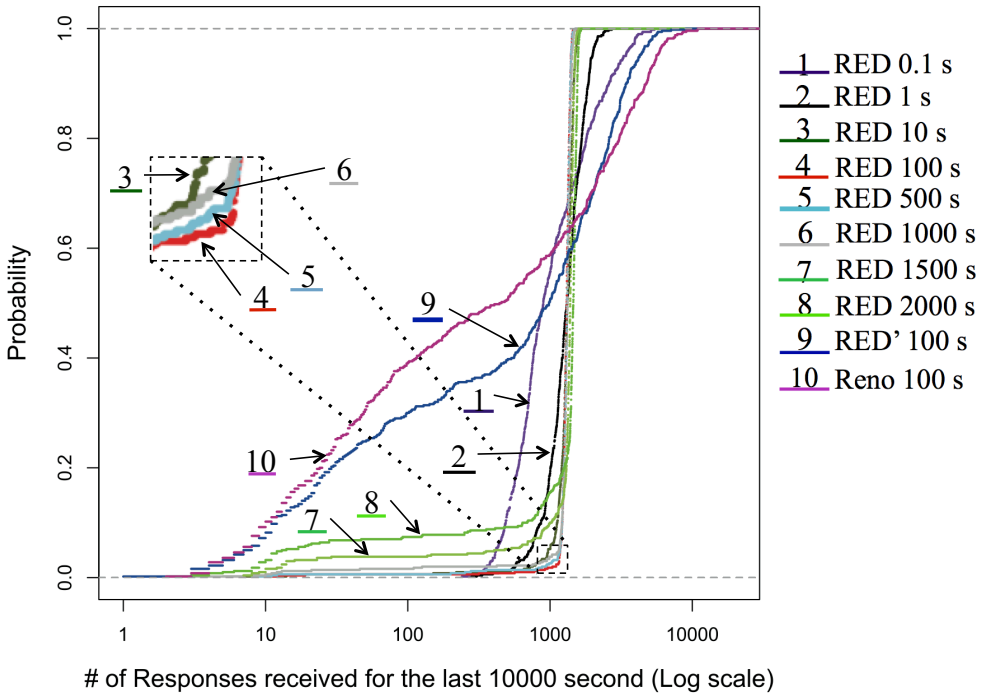


図 5.3: 異なる RTO に対する SNMP GetResponse 数の累積分布

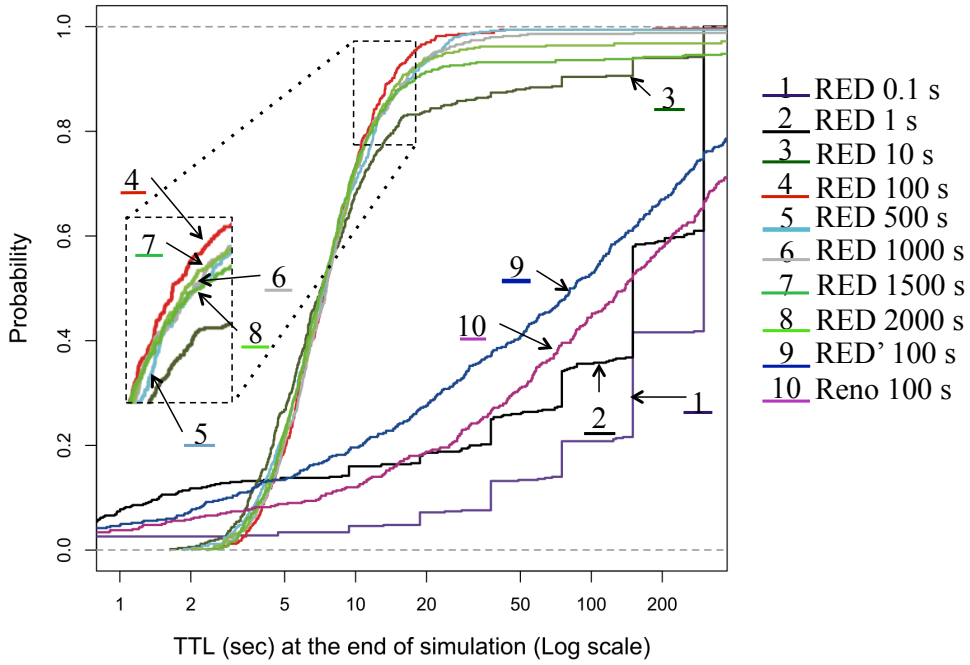


図 5.4: 異なる RTO に対する TTL の累積分布

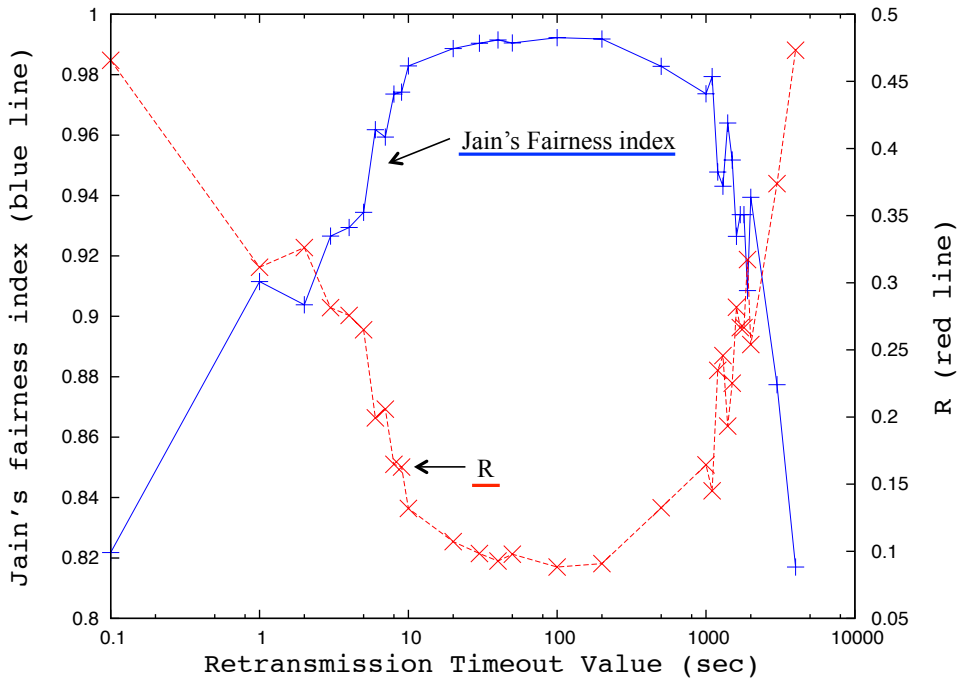


図 5.5: 2 つの公平性指標への RTO の影響

10,000 秒間の平均である．例えば，CPU 使用率の制御性能が 5 の場合，CPU 使用率が TH より平均 5%乖離していることを意味する．そのため，収束時に公平，かつ CPU 使用率の制御性能の高い（小さい）TH_{min} を選択する必要がある．

このため，HTTP Get request 間隔 $\overline{m}$ を 0.1 秒，RTO は 100 秒，TH を 20 秒に設定し，TH_{min} を 15 秒から 19 秒まで変えながら 3 万秒間，10 回のシミュレーションを行った．最後の 10,000 秒間の R と CPU 使用率の制御性能を測定し，10 回分の平均を示したのが図 5.6 である．図 5.6 では TH_{min} が大きいほど R が悪化（大きくなる）し，CPU 使用率の制御性能は改善（小さくなる）する結果を示している．TH が 20%であるため，CPU 使用率の制御性能を $TH(20\%)$ の一割程度 (2%) に保つには TH_{min} を 19%以上にする必要があるが，R は TH_{min} が大きくなるほど急速に悪化する．TH_{min} の選択は，運用者の制御ポリシーにも依存するが，本論文では，R が 0.1 ，CPU 使用率の制御性能が 3.2%となる TH_{min} =18%の場合が最善と判断した．

5.5　条件 3 の評価

実際のモニタリング環境では様々な間隔の HTTP Get request が共存する．本節では図 3.2 に示すように，同一の HTTP Get request 間隔 $\overline{m}$ を持つ NMS と OID の組からなるグループを L 個用意し，各グループ間の HTTP Get request 間隔 $\overline{m}$ が異なる混合環境におけるグ ループ間の公平性を論じる．まず，シミュレーションの設定について説明する．一つのグループが 25 対の NMS と OID からなる 20(= L) 組のグループを作り，各グループの HTTP Get request 間隔 $\overline{m}$ を，それぞれ 0.1, 0.3, 0.5, 0.8, 1, 3 ,5 ,8 ,10 ,30 ,50 ,80 ,100 ,130 ,150 , 180 ,200 ,230 ,260 ,300 秒に設定した．TH や TH_{min} など他の設定は 5.3 節と同じである．以上の状況での最適な RTO を探す

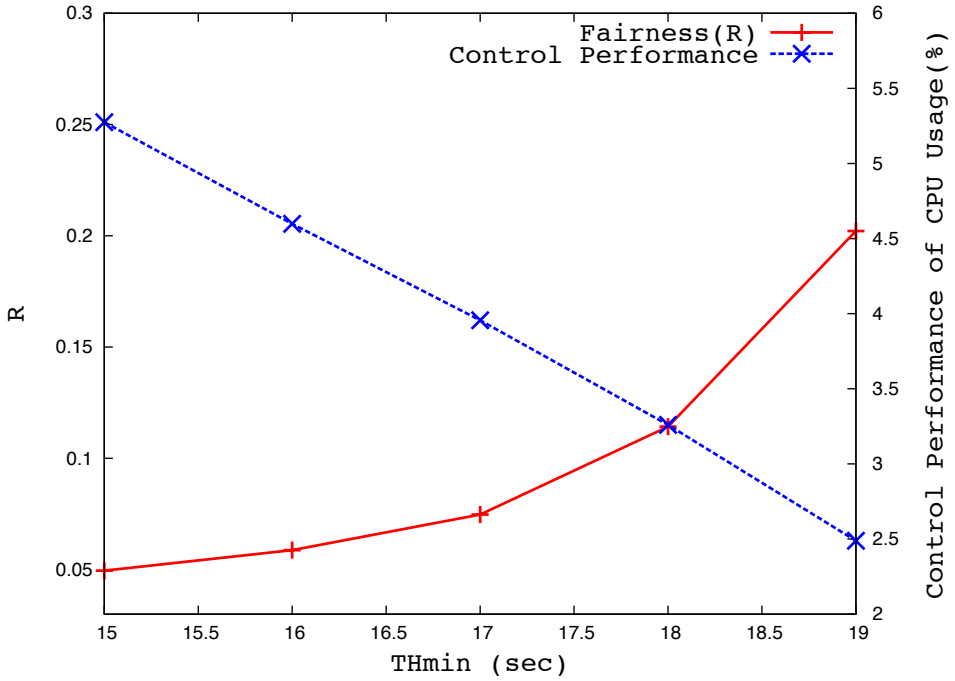


図 5.6: TH_{min} に対する R 値と CPU 利用率の制御性能

ため，RTO を 1, 5, 10, 50, 100 秒に変え，30,000 秒間のシミュレーションの最後の 10,000 秒の間各 cache が取得した SNMP GetResponse の数（x）から各グループの R（式 (3.2)）と SNMP ratio(%) を求めた．図 5.7 と 5.8 に以上を 10 回行って得た平均と誤差を示した．

R は 20 組ある各グループ内の NMS 間で取得した x のバラツキを示し，R が小さいほどグループ内のバラツキは小さいことを意味する．図 5.7 の結果では，$\overline{m}$ が 0.1 秒から RTO へ長くなるに連れ R が大きくなり，$\overline{m}$ が一定以上長くなると R が小さくなることが分かる．例えば，RTO が 1 秒の場合，$\overline{m}$ が 0.1 秒から 8 秒まで R が大きくなるが，10 秒以上ではまた下がる．この理由は，$\overline{m}$ が RTO と近い，または長いグループの場合，次の HTTP Get reqeust が到着する前に RTO がタイムアウトにより図 4.2 の (9) に示す初期化が行われる cache がグループ内に混ざっている可能性が高いため，バラツキが大きくなる．また，$\overline{m}$ が RTO よりずっと長くなる組は，すべての cache が RTO タイムアウトされ，バラツキが小さくなるため R が小さくなる．そのため，複数の HTTP Get request 間隔 $\overline{m}$ からなるグループが存在した場合の最適 RTO は $\overline{m}$ の最大値に合わせれば良いことが分かる．

SNMP ratio は HTTP Get request が輻輳制御により drop（図 4.3 のパス (5)-(6)-(7)-(8)-(9)）される割合が高くなるほど小さくなる．図 5.8 は異なる RTO に対する，各 HTTP Get request group の SNMP ratio を示している．条件 3 により，HTTP Get request 間隔 $\overline{m}$ が小さく，多く HTTP Get request を出すグループ程 SNMP ratio が小さくなり，公平と判断できる．各 RTO に対する結果を見ると，RTO 100 秒のみ $\overline{m}$ と SNMP ratio の関係が線形 ($SNMP\_ratio = 2.79 \times \overline{m} + 1.06$, $\overline{m} < 1$) であり，リクエスト間隔の短いものほど抑制されている．

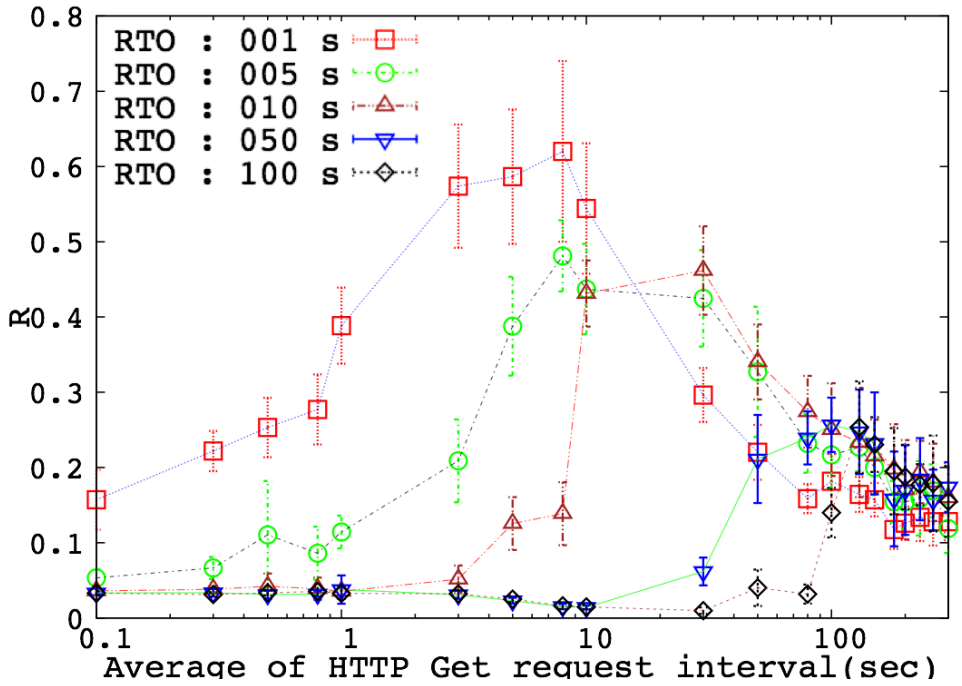


図 5.7: 混合環境における HTTP Get request 間隔 $\overline{m}$ と R 値

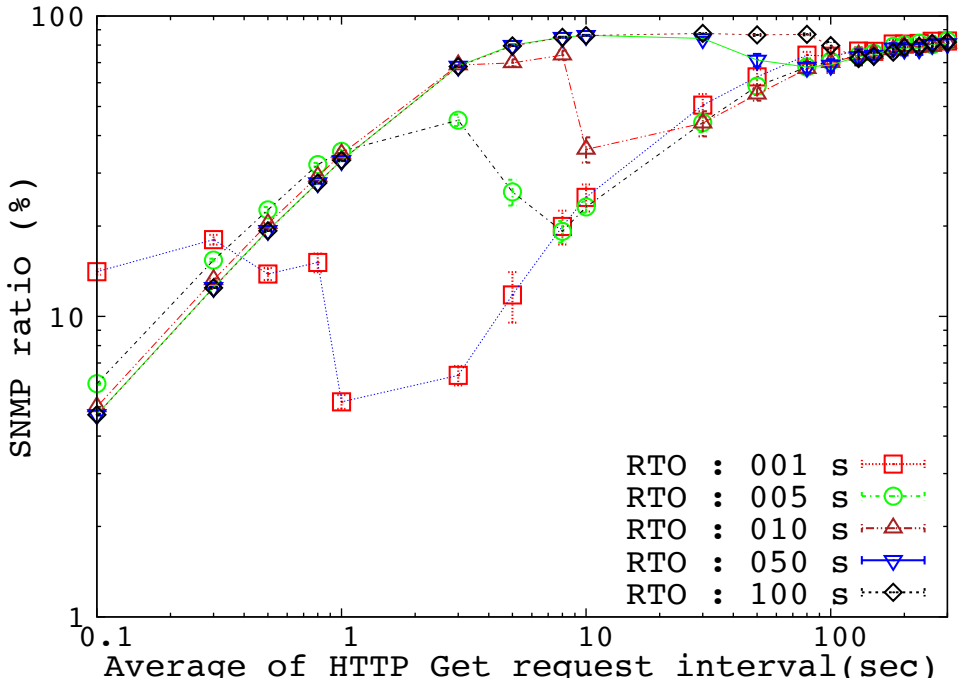


図 5.8: 混合環境における HTTP Get request 間隔 $\overline{m}$ と SNMP ratio

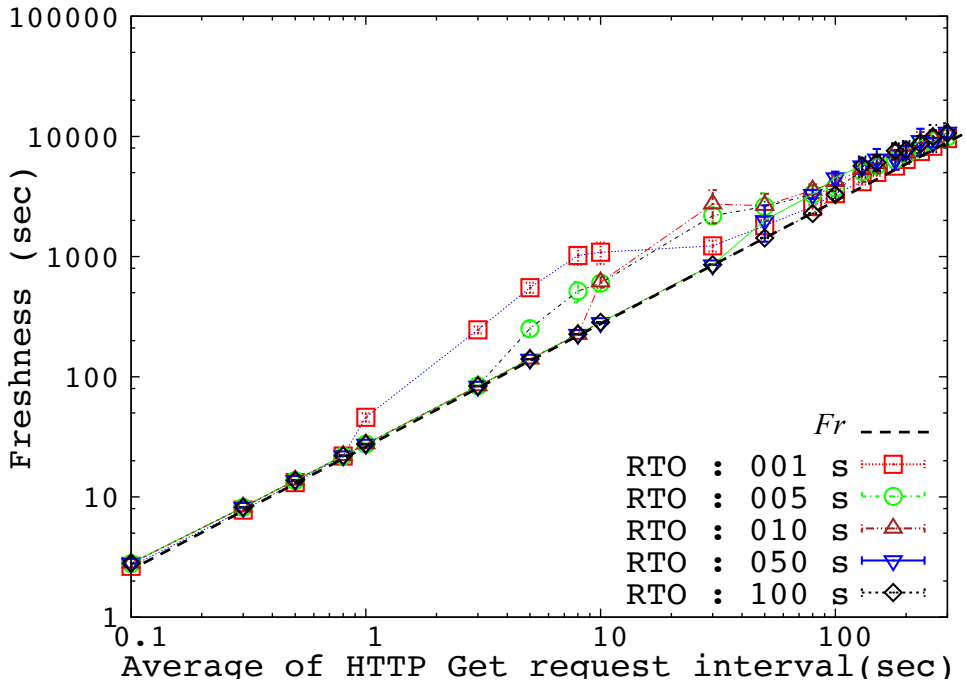


図 5.9: 異なる HTTP Get request 間隔 $\overline{m}$ に対する鮮度

5.6　条件 4 の評価

5.5 節では，HTTP Get request 間隔 $\overline{m}$ が短く多くのリクエストを出す NMS へのペナルティに関して論じたが，短い $\overline{m}$ を持つモニタリングの精度が犠牲になるのは好ましくない．5.5 節と同じ条件で，20 グループ各々の鮮度 (Fr) を式 (3.3) にしたがって求めたものを図 5.9 に示す．式 (3.3) の鮮度は，各グループの cache が新しい SNMP GetResponse を得るまでの遅延の平均である．図 5.9 は，RTO=100 秒の結果のみ，HTTP Get request 間隔 $\overline{m}$ と鮮度 (Fr) がほぼ線形関係になり，$Fr = 29.48 \times \overline{m}$ と近似できることが分かった．新しい SNMP GetResponse を得るまでの平均待ち時間 (Fr) が，NMS からの HTTP Get request 間隔 $\overline{m}$ に比例していることを意味し，$\overline{m}$ の短いモニタリングほど短い遅延時間になるという望ましい特長があることが分かった．

5.7　まとめ

本章では，ns-2 で構成したシミュレーションの基に 4 章の提案が 3 章の条件を満足するかを評価した．まず，条件 1，管理トラヒックによる Agent の CPU 使用率増加を一定値 (TH) 以下に保つことは，4.3 節で提案した Reno による制御で，Agent の CPU 使用率が TH 以下に保てることを確認した．500 個の NMS から 0.1 秒の間隔で HTTP Get request を Tambourine に向けて送信した場合，既存の手法では CPU 使用率が 100%まで高くなったことに対して，Reno では TH である 20%近傍に収束することを示した．次に，条件 2，同一リクエスト間隔の NMS 間の公平性は，NMS からの HTTP Get request 間隔を 0.1 秒に設定し，異なる RTO と 4.4 節で提案した RED ’や RED そして Reno のキャッシュ制御手法を組み合わせて，Agent から受信する SNMP GetResponse の

数を測定し CDF で示した．CDF の結果から，RTO は 100 秒，キャッシュの制御は RED の組み合わせが，同一リクエスト間隔の NMS 間の公平性の面で一番優れていることが分かった．条件 3, 異なるリクエスト間隔の NMS グループ間の公平性の評価では，20 組の HTTP Get request 間隔の NMS グルームが取得する SNMP GetResponse の数から各グループの R と SNMP ratio(%) を求めた．HTTP Get request と SNMP GetRequest との割合である SNMP ratio の場合，HTTP Get request 間隔が小さく，多く HTTP Get request を出すグループ程 SNMP ratio が小さくなったため，公平と判断した．HTTP Get request 間隔が RTO より長くなると，次の HTTP Get reqeust が到着する前に RTO のタイムアウトにより図 4.2 の (9) に示す初期化が行われる cache が現れるためバラツキが大きくなることが，R 値により確認された．そのため，条件 3 を満足させるためには，想定する HTTP Get request 間隔の最大値に RTO を設定すれば良いことが分かった．最後に，条件 4，頻繁な問い合わせる NMS に最大限応じられることは，20 組の異なる HTTP Get request 間隔の NMS のグループに対し各グループが新しい SNMP GetResponse を得るまでの平均待ち時間 (Fr) を求めた．その結果，NMS からの HTTP Get request 間隔と Fr が比例することが分かった．これは短いモニタリングほど短い遅延時間で Agent からのレスポンスを得ていることを意味し，条件 4 を満足していることを意味する．

第6章

考察

6.1　はじめに

本章では，手法の有効性を前提とする大規模ネットワークの規模を明確にした上で，評価実験結果を大局的観点から考察する．特に，SNMP モニタによるネットワーク機器の CPU 負荷を 20%以下に抑えるという仮定が，現実のネットワーク運用環境から考えて妥当なものであること，方式を適用する際に必要となるネットワーク機器の SNMP 負荷特性を把握する作業コストも比較的軽微なものと予想されることを示す．新たな技術観点から懸念されるリバースプロキシ Tambourine 自体の負荷分散に関しては，既存技術での実現方法を示し，大規模ネットワーク運用環境に導入可能な技術であることを示す．

6.2　大規模ネットワーク

大規模ネットワークと呼ぶ場合，企業網や LAN では管理対象が 1000 台を越える場合が一般的である [76] [77]．データセンタ関係では，ネットワーク内のルータ・スイッチの総ポート数が数千ポート規模のものを大規模と言及することがある [78]．Cisco 社が大規模ネットワーク管理用に開発した Meraki では，数百の企業網の一括管理や，100 箇所の大規模のネットワーク内にある数千台の無線 LAN アクセスポイントの一括管理をしている [79]．また，運用管理に関しては，アジアの中規模系のキャリアの VPN サービスで，数百社から数千社を対象に 100 人規模の管理者がそれぞれ指定された企業網の管理に従事する事例がある．

このような現在のネットワーク管理は，物理ルータ時代のものと考えるべきであり，今後，SDN や NFV (Network Functions Virtualization) が普及し Virtual Router の管理までが対象になると，管理対象となるネットワーク機器の台数はさらに多くなり，さらに複雑な管理が要求される．このため，ネットワーク機器の管理をプロキシと呼ばれることの多い管理サーバ経由とすることで，ネットワーク管理システム全体の機器ベンダ依存性を低減させることが一般的になっている．今回対象としているリバースプロキシ Tambourine は，ネットワーク管理の頻度と粒度を現状で一般的な 5 分間から飛躍的に高める（例えば 10 秒間隔）ために，このようなプロキシが持つべき機能として位置づけている．

一般ユーザ向けインターネット接続サービスでは，OKI MileStar BM1410 [80] のような OLT(Optical Line Terminal) で FTTH(Fiber To The Home) を実現し，5000 万世帯を収容した場合，管理対象は更に増え，単純計算で 10 万台の規模になる．しかし，現状ではベストエフォートサービスが主体であり，この論文が目指すようなモニタ間隔の短い精緻な管理はエンド・ツー・エンドの VPN サービスが一般ユーザまで浸透するまでは必要ないと考えている．

6.3　有効性評価実験のモデルと適用可能性

6.3.1　TCP 型制御による公平性の実現

TCP の分散制御で公平性を実現することは一般的に難しい．TCP の RED では全セッションの状況を把握することが難しいため，ランダムにパケットを drop しているが，この方式で TCP のセッションに公平性を実現するのは難しい．本論文のアーキテクチャではプロキシ Tambourine は全セッションの状況を把握可能なため，ランダムにリクエストを落とすのではなく，頻度の高い OID リクエスト中でラウンドロビンで決定論的に落とす制御の方が公平性を実現しやすいとも考えられる．しかし，TCP 制御の枠内でも，頻度の高いリクエストに非線形のペナルティを加えて

drop することにより同様の公平性を実現できることを示し，そのような公平性を実現できる非線形確率密度関数の一例を示した．

6.3.2　シミュレーションのネットワーク構成の妥当性

想定する大規模ネットワークでは，6.2 節の第 1 の想定から，VPN サービスのコアルータには数千社から数万社のトラヒックが流れ，数百人規模の運用者がこのルータの複数の OID にアクセスしているものとした．第 2 の想定により，災害などでネットワークが輻輳したときは，これらの運用者が連携を十分取れずに一斉にアクセスすると考えられ，実験で軽負荷と定義した 100 リクエスト/秒を超える管理トラヒックが発生する可能性が十分にある．有効性評価実験のモデルでは，500 台の NMS から 500 個の OID に対して管理する構成での特性を求めたが，SNMP セッション間の公平性の評価を易しくするため，この構成を採用した．100 台の NMS が 1 台当たり 5 個の OID をモニタするという，より一般的場合の特性も，この実験で尽くされている．

6.3.3　DDoS 攻撃への耐性

また，運用者間の統制が取りにくいという第 2 の想定は，各 NMS が独立に管理トラヒックを発生するというモデルで近似できたと考える．さらに，この手法では，外部ネットワークとプロキシ Tambourine をつなぐことになるため，DDoS 攻撃も想定する必要がある．SNMP トラヒックを軽負荷の範囲内に保ち，かつモニタ間隔を可能な限り短くしたいという提案手法は，SNMP 問い合わせ負荷が軽負荷と一般的に許容される 20%を上限にした場合，DDoS を回避し，運用者間の公平性を保ち，取得情報の鮮度を改善し，かつ制限以下の CPU 使用率に抑えられることを実験結果から示すことができている．

ただし，図 5.9 から頻繁に問い合わせる NMS への追従性が良いため，DoS 攻撃者へ大きなリソースが割り当てられる欠点は免れない．この状況下でも，DoS 対象以外の OID のモニタリングの追従性は定量的に保証される．プロキシ Tambourine 自体での DoS 攻撃に対しては，Tambourine の負荷分散で回避できている．

6.3.4　SNMP モニタがネットワーク機器へ与える負荷の妥当性について

本論文では，SNMP では 1 分間の CPU 使用率しか観測できないネットワーク機器を対象に，同一負荷で 1 分間以上（論文では 30 分間）の連続負荷実験を行い，30 回の SNMP 応答の示す CPU 使用率の平均値を，その負荷に対する 1 秒間単位の CPU 使用率とみなした．このように，予備実験として，種々の負荷に対して CPU 使用率を測定し，SNMP 問い合わせ負荷と CPU 使用率間の換算式を作成することが方式適用の必要条件である．さらに，機器の CPU 使用率をプロキシが問い合わせ負荷から計算可能なことを利用して，問い合わせ負荷から管理対象の CPU 負荷を推定してネットワーク管理をするという構成を採用している．このため，ルータにおける IP ヘッダのオプションヘッダの処理やルーチングテーブルの更新などの処理（以下主要処理と呼ぶ）のような SNMP 問い合わせ以外の負荷がある場合は，推定値より実際の CPU 使用率の方が一般に高くなる．したがって，主要処理に影響が出ないように SNMP の問い合わせ制御することが求められ，問い合わせ処理の負荷の上限は軽負荷の範囲にする必要がある．これに対しては，Cisco のマニュアルに CPU 使用率が高いほうで 80%を超えた場合と，低いほうで 20%を切った場合にトラップを出す

ように設定したものがあることを考え，20%以下を軽負荷の基準とした．問い合わせによる CPU 使用率の上限 TH を 20%以下に設定しておけば，現実の運用では通常問題ないと考えられる．もちろん，主要処理で実際の CPU 使用率が 80%近くのときに，さらに SNMP で上限の問い合わせを行うと当然過負荷になる．このような場合は対象機器からトラップを出すように設定しておけば，その受信時に問い合わせに伴うトラヒック量を半減，またはそれ以下にすればよい．ただし，自明な処理と考え，本論文では分析の対象にはしていない．

SNMP 問い合わせと CPU 負荷の関係の一般性に関しては，軽負荷の範囲での近似式が問題となる．許容される CPU 使用率の上限を 20%以下とした場合，Cisco 社のルータ・スイッチや Alaxala 社の機器に対しては良い直線近似が得られている．また，一般的に CPU 使用率は SNMP 問い合わせ負荷に対して単調増加関数になると考えられるため，実際の運用では，許容される CPU 使用率の上限に対する SNMP 問い合わせ間隔が得られていれば，その点と原点との間を図 4.1(b) のように線形近似して運用しても，SNMP 問い合わせ負荷を上限以下に抑えるという目的達成には十分である．

6.3.5　ネットワーク機器の SNMP 負荷特性取得コストについて

現実の複合機器で構成されるネットワークに適用する場合には，管理対象機器に対し予め SNMP 問い合わせと CPU 負荷の関係を求める作業が必要になる．測定に当たっては，実運用に入っている機器が大半であるため，それと同種の機器を使って測定する作業になる．1 種類当たり許容される CPU 使用率の上限を求めるためには，複数の問い合わせ間隔でそれぞれ 1 分間計測し，それにより計算した上限の CPU 使用率に対応する問い合わせ間隔でさらに 1 分間計測を数回ほど行って確認実施する必要がある．このため，1 種類あたり 30 分程度の作業時間がかかると予測され，管理対象機器の種類に対して作業コストが比例するという欠点がある．6.4 節に示すように Tambourine の負荷分散が可能であるため，提案手法の適用上の制約は，管理対象となるネットワーク機器の台数ではなく，むしろ種類に依存する．換言すれば，管理対象機器の CPU の負荷特性を求める事前測定作業のコストが適用限界を決める．ただし，一般に企業は設備の購入に当たって，購入費用の低減を図るため，一時に大量の機器を購入することが多いと考えられる．このため，管理対象が数千台あっても，全て異なる種類であるような場合は一般に少なく，機器の種類は現実に測定可能な範囲内にあることが多いと考える．

6.4　負荷分散

多数の NMS からの多数のリクエストがある場合，Tambourine 自体がボトルネックになる恐れがある．このような場合，多数の NMS からの負荷をロートバランサで，複数台の Tambourine へ分散させることが一般に考えられる．しかし，CPU 使用率を計算するには，SNMP 機器への SNMP GetRequest の総数を知っていなければならないため，この情報をシェアできていない限り，実現は困難である．

6.4.1　WCCP による Tambourine の負荷分散

一つの解として，図 6.1 で示すように Tambourine を 1 台とし Web Cache Communication Protocol (WCCP) [81] 対応ルータと複数のキャッシュサーバを Tambourine と NMS の間に配置して負

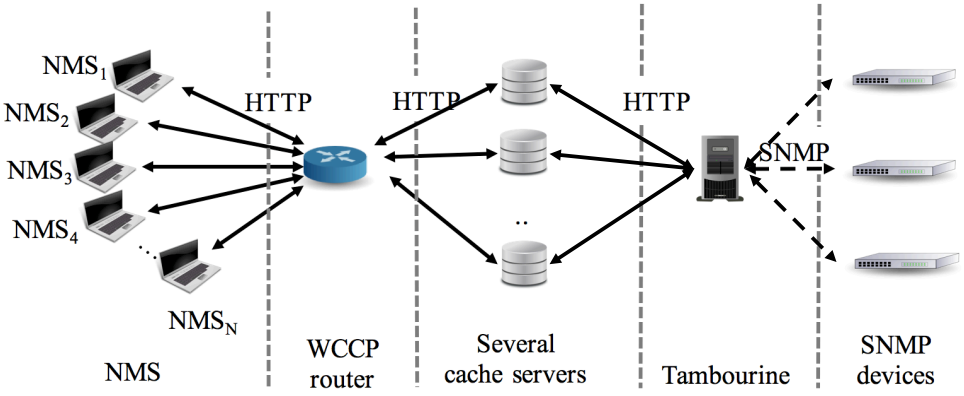


図 6.1: WCCP による Tambourine の負荷分散

荷分散することが考えられる．WCCP ルータは，ソース (NMS) の IP アドレスとポートでハッシュし，NMS からの HTTP リクエストをリダイレクトするキャッシュサーバを選択する．キャッシュサーバは受信した HTTP リクエストを Tambourine へ転送し，Tambourine から受信したレスポンス (MIB 情報) をキャッシュし，NMS へ返す．この結果，頻繁に問い合わせされる OID の MIB 情報を多数のキャッシュサーバへ分散しキャッシュできる．WCCP のキャッシュサーバは HTTP 1.1 のキャッシュ標準に対応している [82] ため， Tambourine が HTTP response を返す際，HTTP ヘッダの Cache-Control header の max-age argument に MIB 情報の TTL 値（単位は秒）を，Expire header field に timeout 時刻を「現時刻＋ TTL」で示すことにより，各キャッシュサーバが保存した MIB 情報を Tambourine に合わせて更新することができる．

この手法の利点は，Cache-Control header にキャッシュの有効期限を追加した HTTP レスポンスを返すだけで，負荷分散が可能になることである．しかし，頻繁に問い合わせされる OID の MIB 情報が複数のキャッシュサーバへ分散されると，多数のキャッシュサーバからのキャッシュの更新で Tambourine の負荷が増加する可能性が依然として残る．

6.4.2　Dispatcher による Tambourine の負荷分散

次に，図 6.1 の WCCP router を dispatcher で置換した dispatcher ベース負荷分散手法を検討する [83] [84]．WCCP ルータの場合，ソースの IP アドレスとポートによってキャッシュサーバを選択するが，dispatcher は予め各キャッシュサーバ毎に，担当する OID でテーブルを作り，HTTP リクエストの URL から取得した OID がテーブル内の OID とマッチした場合，HTTP Get request を当該キャッシュサーバにフォーワーディングする点が違っている．NMS が dispatcher に向けて送った HTTP Get request を dispatcher が受信すると，dispatcher は HTTP Get request の destination IP アドレスを担当するキャッシュサーバの IP アドレスに書き換える．また，キャッシュサーバからのレスポンスは，source IP アドレスをキャッシュサーバから dispatcher の IP アドレスに書き換える必要がある．

6.5　まとめ

手法の有効性を前提とする大規模ネットワークの規模は，現実のネットワークビジネスの運用を踏まえたもので，数千社から数万社のトラヒックが流れる VPN サービスで数百人規模の運用者がルータの複数の OID にアクセスしているものである．SNMP モニタによるネットワーク機器の CPU 負荷を 20%以下に抑えるという仮定も，現実のネットワーク機器の運用環境から導き出されたものである．さらに，方式を適用する際にネットワーク機器の SNMP 負荷特性を予め測定する必要があるのが本方式のスケーラビリティを考える上での欠点であるが，現実のネットワークに導入される機器の種類を考えると，一般に比較的軽微なものなると予想される．技術的にはリバースプロキシ Tambourine 自体のスケーラビリティが問題になるが，Tambourine の負荷分散に関しては，既存技術での実現方法を示した．上記を総合し，Tambourine は大規模ネットワーク運用環境に導入可能な技術であると考える．

第7章

結論

7.1　結論

本論文では，SDN の導入により，大規模かつ中央集中化しつつるあるネットワーク管理において，既存のネットワーク機器の管理手法である SNMP を共存させるためのプロキシとして提案されていた Tambourine に新たなキャッシュ管理手法を提案した．そのため，既存の Tambourine が持つ問題を示し，提案を構成する各通信エンティティの目的要件を提示し，それらを満足するための設計方法を示し，シミュレーションにより各条件を満足できることを示した．

プロキシによるネットワーク管理は，管理者の端末である NMS，被管理対象のネットワーク機器である Agent，そして NMS と Agent の間の中継の役割をするキャッシュサーバである Tambourine から構成されている．NMS は Agent の状態をなるべく高い粒度でモニタリングすること，Agent は SNMP 機器本来の作業に忠実しながら NMS のモニタリング要求に応じること，そして Tambourine はキャッシュで NMS と Agent の要求のバランスを取る目的を持っている．既存の Tambourine のキャッシュ管理手法は，Agent の状態情報の値が変化する頻度に合わせてキャッシュの更新周期を変えることが骨子であったが，Agent の負荷を考慮してないため，頻繁に値が変化する状態情報への問い合わせを過度に送る方法で DDoS 攻撃が行なわれる恐れがあった．

以上の目的と問題を解決するため，次の 4 つの要求条件と評価尺度を提案した．まず条件 1 として，Tambourine は，Agent の CPU 使用率増加が一定値 (TH) 以下に保って管理トラヒックを Agent へ送らなけてはならない．このため，Tambourine は管理トラヒックによる Agent の CPU 使用率を計算しキャッシュの TTL を制御する必要がある．条件 2 は，同一リクエスト間隔の NMS 間で情報取得の公平性を実現することである．公平性の尺度として NMS からの HTTP リクエストに対し Tambourine が得た SNMP レスポンスの数の標準偏差を平均値の相対値として見た変動係数 R を使い同じ粒度で管理トラヒックを出す NMS 間での公平性を確保する．条件 3 として，異なるリクエスト間隔の NMS グループの間でも情報取得の公平性の実現を掲げ，評価尺度は，やはり変動係数 R が使え，R が同じであれば，リクエスト間隔に依らず，NMS グループ間の相対的なばらつきが同じであることを意味する．最後に，条件 4 として，頻繁な問い合わせる NMS に対しては，上記の条件を満足する範囲内で，最大限応じなければならないとした．評価尺度としては，Tambourine のキャッシュにより NMS から Tambourine への HTTP リクエストと Agent から実際に得た OID 情報の間には，取得遅れが生ずるため，取得遅れの平均値を鮮度 (Fr) として定義し評価した．

提示手法は，Tambourine から Agent への SNMP リクエスト/秒と Agent の CPU 使用率間の関係を事前実験により求め，Agent への SNMP リクエスト/秒から Agent を CPU 使用率を推定することを基本とする．このため，PC から Alaxala3640s と Catalyst 2950 に SNMP リクエストを任意の一定間隔で送り，各機器の過去 1 分間の CPU 使用率を測定し近似式 (4.1) を求めた．これは，Tambourine が Agent へ送り出す SNMP リクエストの総数 N を把握しているため，近似式 (4.1) に N を代入することにより，その時点での SNMP リクエストによる CPU 使用率の上昇分を計算できることを用いている．

条件 1 に対しては，NMS からの問い合わせに対し，輻輳の状態にはキャッシュから値を返し，他の場合は TCP Reno に類似したキャッシュの更新間隔 (TTL) を調整する手法を提案した．ここで，CPU 使用率増加が予め指定した閾値 (TH) 以上になることを輻輳と定義した．また，TCP の RED に類似したリクエストの非線形手法を提案した．その際，リクエスト送信頻度の高い NMS ほど，CPU 使用率の計算の際，TH 以上の値を得る確率を上げることにより，NMS 間で公平に SNMP 機器からレスポンスを取得でき，条件 2 と 3 を満足させることができることを示した．一般に TCP 型の分散制御で公平性を実現するのは難しい．本論文のアーキテクチャではプロキシ Tambourine

は全セッションの状況を把握可能なため，ランダムにリクエストを落とすのではなく，頻度の高いものをラウンドロビンで決定論的に落とす制御も可能と考えられる．しかし，TCP 制御の枠内でも，頻度の高いリクエストに非線形のペナルティを加えることにより公平性を実現できることを示せた．本論文では，そのような公平性を実現できる非線形関数の一例を示している．最後に，条件４の評価尺度として，取得データの遅延を評価する指標 (Fr) を定義し，NMS の問い合わせ間隔と取得データの遅延との関係を評価した．

評価では，ns-2 で構成したシミュレーションの基に提案が条件を満足するかを評価した．条件１は Reno に類似したキャッシュの制御で，Agent の CPU 使用率が TH である 20%近傍に収束することで満足されること示した．次に，条件 2 は，HTTP リクエスト間隔が同じである複数の NMS に対し，RED でキャッシュ制御手法行うことにより，NMS 間で公平性に Agent から SNMP レスポンスを得ることができることを示した．また，条件 3 は NMS からの HTTP リクエスト間隔の最大値より大きい RTO を選択し RED でキャッシュの制御をすることで，異なるリクエスト間隔の NMS グループ間でも公平性が改善されることを示した．その際，通常の RED と異なりリクエスト間隔の短いものに非線形のペナルティを与えることが有効であることを示し，それを満足する確率密度関数の一例を示した．最後の条件４に対しては，提案手法により NMS からの HTTP リクエスト間隔と Fr が比例することが分かった．これは短いモニタリングほど短い遅延時間で Agent からのレスポンスを得ていることを意味し，条件 4 を満足していることを意味する．

最後に，本方式適用上のボトルネックになるリバースプロキシ Tambourine のスケーラビリティに関して，Tambourine の負荷分散は既存技術で容易に実現であり，その一例を示した．本方式の導入による欠点は，本来 SNMP 応答を受信できる NMS がキャッシュにより旧い情報を入手する点にある．その評価は SNMP ratio により評価した．SNMP ratio の理論限界とシミュレーションの精度の見極めは残ってはいるが，今回は簡単な実験により，RTO 100 秒のみ平均 HTTP リクエスト間隔 $\overline{m}$ と SNMP ratio の関係が線形になり，リクエスト間隔の短いものほど抑制されていることを示せた．上記を総合し，Tambourine は大規模ネットワーク運用環境に導入可能な技術である．

7.2　将来研究

SDN や NFV は，現在では，主として企業ネットワークを対象に提供されている技術である．一方，HEMS (Home Energy Management System) が普及するとスマートハウスもこのような通信サービスの顧客になる可能性がある．本論文では一般ユーザによるネットワーク管理や設定が可能になることをネットワークのパーソナル化と呼ぶ．筆者は，以前，ユーザが持っているコンテンツを SNS 上の友達と P2P（Peer-to-Peer）共有するために DAS（DLNA agents for SNS）という Proxy を提案していた [2]．DAS はユーザ間で高画質な動画をストリーミングして鑑賞するため，エンドツーエンドで十分なネットワーク帯域が確保されていることを前提にしていた．例えば， H.264/Advanced Video Coding (AVC) でエンコーディングされた一時間分の 720p 画質の動画の場合，13 GB の大きさになり，スムーズな受信のためには 29.58Mbps 以上の帯域を必要とする [85]．しかし，2013 年の Akamai のサーバに対する世界各国の平均接続速度を鑑みると [86]，1,2 位である韓国と日本でもそれぞれ 22.1Mbps と 13.3Mbps であり，必要とされる速度には達していない．また，今後インターネットの通信速度が改善されたとしても，帯域共有型のベストエフォートサービスでは常にスムーズな再生を期待することはできない．したがって，QoS（Quality of Service）制御は必須であるが，ISP がネットワークの設定を動的に変更することはコスト的に難しく，QoS を確保したサービス提供はできていない．SDN およびその後継技術には，DAS のよ

うなユーザ間の高画質動画コンテンツの共有のため，ユーザの必要に応じでネットワークサービス内容を柔軟に変更できるような機能の実現を期待したい．以下が，想定アプリケーションである．

7.2.1　想定アプリケーション：DAS（DLNA agents for SNS）

ネットワーク管理は，いかに大規模になったとはいえ，ホームネットワークの管理を実現するまでには，現在は至っていない．ここでは，具体的なアプリケーション事例を基に検討し，ニーズがあることを示し，将来の大規模ネットワーク管理のスケール感を示したい．

最近の写真や動画などのコンテンツ共有は Web 上のサービスを介して行うのが一般的になっている．たとえば，コンテンツ共有を専門とする Youtube や Flickr だけではなく，Facebook のような SNS（Social Network Service）や Dropbox のようなクラウドサービスを利用しても簡単にコンテンツ共有はできる．Web 上のサービスを介する共有の基本的な手順は，サーバへコンテンツを転送し，共有する相手の範囲を決めるのが一般的である．サービス提供者が運営するサーバを中心にコンテンツ共有を行うため，ユーザはサーバやネットワークを管理する必要がなく，簡単かつ安く共有ができるメリットがある．しかも，スマートフォンの普及により，撮影した写真をその場で簡単にサーバに転送し共有設定できるようになっている．

しかし，このような Web を基板とする情報/コンテンツ共有は基本的にユーザの個人情報が含まれているコンテンツを全てサービス提供者のサーバへ転送することを前提としているため，プライバシ侵害，情報漏洩の恐れがある．また，大量のコンテンツの共有を望む場合，共有できる動画サイズや画質に制限があるだけではなく，コンテンツをサーバへ転送し，共有できるフォーマットへ変更するための待ち時間が大きい．このため，ユーザが持っている DLNA コンテンツを SNS 上の友達と P2P（Peer-to-Peer) で共有するための Proxy である DAS（DLNA agents for SNS）を提案した [2].

DLNA（Digital Living Network Alliance）は，家電やパソコンのメーカを中心に 2004 年 6 月発足した団体で，ホームネットワーク上の機器間のマルチメディアの共有のためガイドラインの作成と認証を行っている．DLNA ガイドライン（以下 DLNA）で使われている機器は大きく DMS (Digital Media Server) と DMP (Digital Media Player) に分類される．DMS はハードディスクレコーダの発展系で，コンテンツを持ち，ホームネットワーク内に配布する役割をする．DMP は DMS により送られるコンテンツを再生するテレビの発展系である．DLNA では機器の発見に Multicast を利用するため，ホームネットワーク内の仕様が前提で，インターネットを跨いての共有には対応していない．

図 7.1 に DAS により，SNS 上の友達と家庭内の DLNA コンテンツを共有するため必要な構成要素を示した．ユーザは PC を利用し SNS サーバへアクセスし，共有相手とコンテンツの管理を行う．共有コンテンツの追加や再生は DLNA 機器を利用して行う．DAS は DLNA 機器と SNS サーバの間で共有に必要な情報のやりとりを中継する．SNS 上の友達とインターネットを介しての DLNA コンテンツを共有するためには，次の設計要件を満足する必要がある．

1 リモートの DLNA 機器とのコンテンツ共有：DLNA では Multicast で機器の発見を行うため，ルータを超えての共有ができない．

2 セキュリティ：家庭間のコンテンツ転送において暗号化せずに HTTP 転送するのはプライバシの側面での問題がある．

3 SNS と DLNA 機器間の連携：SNS サーバ上で DLNA コンテンツを共有設定するためのコンテンツリストを DLNA 機器間でやりとりする機能が必要である．

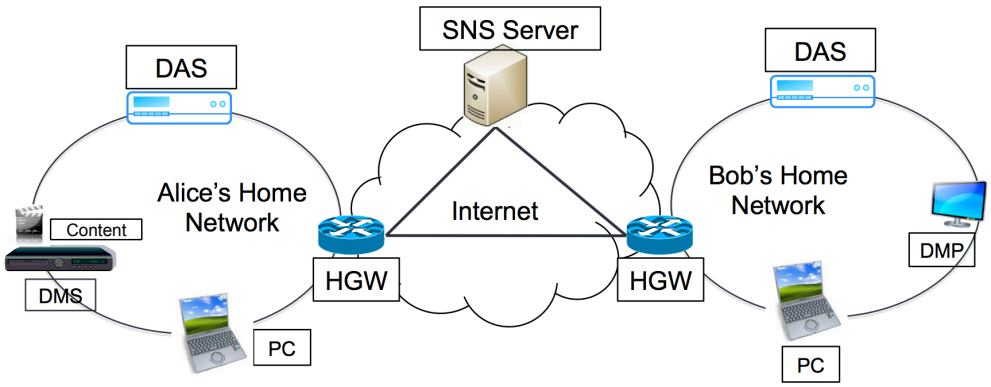


図 7.1: DAS システムの概念図

DAS は以上の問題を解決するため図 7.2 に示す 4 つのモジュールで構成されていた．まず，DLNA Proxy は DMS が持っている DIDL（Digital Item Declaration Language）表記の全コンテンツ名を DLNA プロトコルを使って取得し SNS Agent 経由で SNS サーバへ転送し，また DLNA Proxy の Media Server は SNS Agent を介して SNS サーバから得たコンテンツリストを用いて仮想 DMS（図 7.2 の Media Server）を立ち上げる．仮想 DMS は，マルチキャストで送られる機器探索のための DLNA の M-SEARCH リクエストを終端し，DLNA のコンテンツ再生リクエスト（HTTP リクエスト）を Reverse Proxy に転送する．Reverse Proxy はこの HTTP リクエストの Destination IP アドレスをリモートの Reverse Proxy の IP アドレスに変換，あるいは逆にリモートから受信した HTTP パケットの Destination IP アドレスをローカルの DMP と DMS の IP アドレスへ変換し，転送する役割を担っている．ここでも Reverse Proxy が Tambourine と同様，HTTP リクエストの加工のために利用されている．ただし，Tambourine は ISP 網内に設置されていたが，この Reverse Proxy は家庭内ネットワークに設置されている点が異なる．

次に，VPN Box は，VPN を自動設定するための設定ファイルを作成し，SNS サーバを介して共有する相手の VPN Box へ送る．ホームネットワークはプライベートアドレスなので，接続する両ネットワークが同一のネットワークアドレスの可能性がある．このため，リモートの DMS が自分のネットワークの機器と IP アドレスが競合しないよう，ホームネットワークでの IP アドレスをトンネル用の IP アドレスに変換することも役割の一つである．最後に，受信側の SNS Agent は，コンテンツリストと VPN 接続情報を SNS サーバから受信する．

利用シナリオとして，Alice が Bob に DMS 上のコンテンツを共有しようとする場合，次のような流れになる．まず，DLNA Proxy が Alice の DMS 内のコンテンツリストを取得し，SNS Agent が VPN Box から得た VPN 接続情報と一緒に SNS サーバへ転送する．SNS サーバ上で Alice が Bob に対し共有設定を行うと，SNS サーバは Bob の SNS Agent へコンテンツリストと VPN 接続情報を送る．Bob の VPN Box は VPN 接続情報を利用して Alice の VPN Box と VPN を張り，Alice の VPN トンネルに対し IP を割り当てる．Bob の DLNA Proxy は SNS Agent 経由で入手した Alice のコンテンツの URL を Alice の VPN トンネルの IP へ修正し仮想 DMS へ追加する．Bob が TV（DMP）を利用して 仮想 DMS へアクセスしコンテンツの再生を要求すると，Reverse Proxy は，コンテンツの IP を参照し VPN 経由で Alice の DLNA Proxy へ転送する．Alice の DLNA Proxy はこのリクエストをターゲットの DMS へ転送する．この結果，Bob の DMP からの DLNA 再生要求が Alice の DMS まで伝送される．Alice の DMS からレスポンスとしてコンテンツが上記の逆

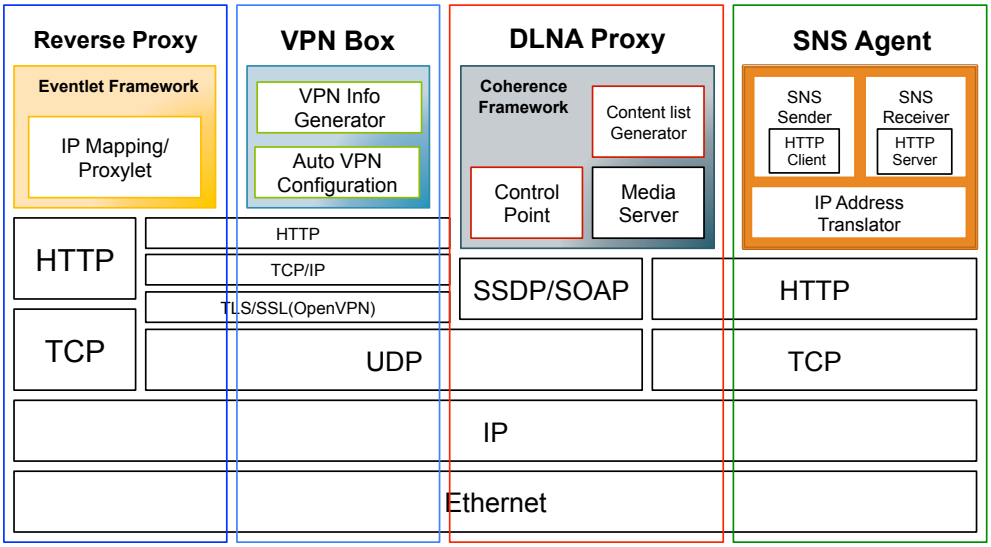


図 7.2: DAS の実装図

順で Bob の DMP まで伝送され，コンテンツ再生が可能になる．

7.2.2　DAS の限界

DAS のメリットとしては，コンテンツ本体を第 3 者へ預ける必要がないため，プライバシや著作権の面で安心できることと，共有できるコンテンツの容量や画質には制限がなく，エンドシステムの仕様で決まることである．従って，ユーザ間のネットワークが高画質な動画をストリーミングしながら鑑賞できるほど十分な帯域が確保されていることが基本的な制約条件である．再生が行われる間は帯域を確保する QoS 制御が必要になるが，ISP はネットワークの設定を動的に変更しなければならず，現行の ISP の設備運用体制では困難である．

また，DLNA では著作権保護技術として DTCP-IP [87] を採用している．DTCP（Digital Transmission Content Protection）は，DTLA（Digital Transmission Licensing Administrator）がライセンスする著作権保護技術のことである．DTCP は USB や IEEE1394 など様々なインタフェース上に規定されており，IP インタフェースのためのものを DTCP-IP と呼ぶ．DTCP-IP では鍵交換をする際に，TTL を 3 以下，RTT を 7ms 以下にする制約を設けて，機器同士が同じ家庭内のホームネットワーク上にあることを確認している．DAS ではユーザが作成したコンテンツのみの共有を前提しているが，DTCP-IP の制限を受ける商用コンテンツを共有するには，各ユーザの DLNA 機器間の TTL や RTT が上記の基準を満足する必要がある．DAS では VPN を用いて TTL の増加を回避したが， ベストエフォートの現行の ISP の設備運用体制では RTT の保証は困難である．

7.2.3　パーソナル NaaS API

広域ネットワーク経由で高画質 DLNA コンテンツを円滑に再生できるようにするためには，前節の VPN Box の欠点を補う，次の設定が NaaS（Network as a Service） API でユーザに提供さ

れていることが望まれる．本節では，一般ユーザが NaaS を利用し，次のネットワークリソースの設定ができることをネットワークのパーソナル化と呼ぶ．

1 QoS 設定：共有コンテンツの画質にあった帯域の確保

2 エンド・ツー・エンド間の遅延の制御：DTCP-IP の制御を受けるコンテンツの共有を可能にするため RTT 未満の転送遅延の保証

3 VLAN 設定：離れたホームネットワークを VLAN で同じサブネットに構成することで，DTCP-IP での TTL 制限や DLNA 機器の発見問題を解決する．

ホームネットワークは，プライベートネットワークであり，DLNA の場合，IPv4 であれば DHCP でダイナミックに割り当てられたアドレスが，IPv6 であればリンクローカルアドレス（fe80::10）またはサイトローカルアドレス（fd00::8）が割り当てられている [88]．また，ホームエネルギー管理で使われている ECHONET Lite の場合は，リンクローカルアドレスのみが使われている．このため，課題 3 に関しては，以下の問題を解決する必要がある．

1 IPv4 アドレスの場合は，同一の IP アドレスを使っている場合があり，両端のネットワークアドレスを未使用のネットワークアドレスを介して，相互変換 (Network Address Translation) する必要がある．

2 IPv6 アドレスの場合も，サイトローカルアドレス（fd00::8）は外部へ広告しないという原則を例外処理する必要がある．アドレス競合の確率は 0 ではないため，やはり Network Address Translation するほうが望ましい．

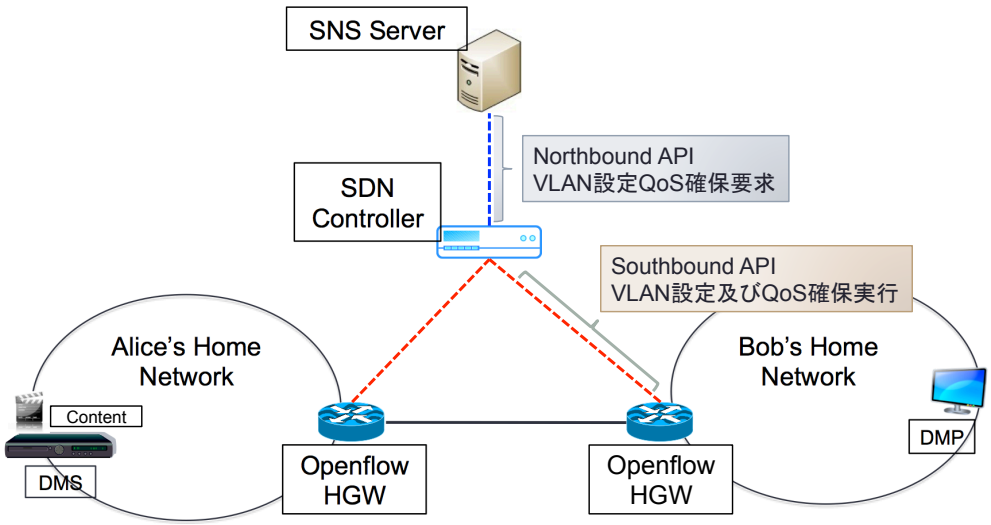


図 7.3: パーソナル NaaS API を利用したシングルドメインの DAS の構成

図 7.3 に，ホームゲートウェイ HDW の VPN Box の部分を SDN ベースに置換し，一つの SDN Controller で制御するシングルドメイン構成を示す．SDN Controller は SNS サーバの NaaS API で制御する．HGW 間は ISP 内に閉じたパスで， SDN Controller も ISP 内の特殊なセグメントに存在する．このため，インターネット上の SNS サーバが NaaS API を利用して SDN コントローラを制御するには，ファイヤーウォール越えて HTTP などを使った相互接続を実現する必要がある．

HGW の入出力にはベストエフォートのインターネットトラヒックと QoS 設定が必要な P2P のトラヒックが共存し，SDN コントローラが Southbound API を使ってフローの管理をする必要がある．図 7.1 の場合，P2P トラヒックは IP トンネルのフローであり，容易に判別可能である．ただし，図 7.1 ではアドレス変換機能を DAS に内蔵していたが，VPN を含め Openflow HGW に標準装備させる必要が生じる．

7.2.4　期待と展望

本節では，企業網を越えてさらにネットワーク管理を大規模化した場合，どのようなサービスが新たに可能になるかを考察した．提案したネットワークのパーソナル化を可能にするパーソナル NaaS API が実現できれば，ユーザ間のコンテンツ共有だけではなく，オンデマンドで広帯域コンテンツサービスを利用することにも適用可能である．例えば，狭帯域を利用するユーザが，一時的に YouTube から広帯域で高画質のコンテンツの提供を受ける場合，Youtube 側がパーソナル NaaS API を利用してユーザのネットワークリソースのステータスと QoS 確保の可能性を把握し，可能であれば，付加サービスとして販売することが考えられる．このように，パーソナル NaaS API をコンテンツサービスで活用することで，ネットワークリソースの小売が可能になり，ISP やコンテンツサービス提供者に新たなサービス提供手段となり得る．ユーザにとっても，簡単に必要な時に必要なネットワークリソースを必要なだけ購入及び利用できるため利便性が改善される利点がある．ただし，この種の Intserv [89] 型のサービスは古くから考えられてきたが，コストパフォーマンスのあるサービスモデルを提案することができず普及しなかった経緯がある．このため，仕様を大きくせずに，コスト，ユーザ利便性，スケーラビリティを折衷したサービスを組み立てる必要があるが，リバースプロキシとキャッシュの導入による管理コストの削減が可能であることが本論文で示せたため，再度チャレンジする意味があると考える．

この論文では，SDN に代表されるインターネットの大規模管理に伴う既存機器と SDN 機器との共存技術にターゲットを絞って検討したが，次なる研究展開は，企業網からホームネットワークにサービス対象をさらに拡大したときに生まれる新しいサービスの創造にあり，一つの可能性として，DAS を紹介した．

謝辞

本研究および博士論文の執筆にあたり, ご多忙を極めるなか常に親身にご指導いただいた浅見徹教授に心から感謝いたします．さらに本研究へのご指導をはじめ, その他研究生活全般において, 多大な労力をかけてさまざまな場面でご指導いただいた川原圭博准教授に心からお礼を申し上げます．また, 本研究へ貴重なアドバイスを下さった博士課程の Abdul Hamid Ahmad Kamil 氏, 日々の研究室生活でお世話になりました浅見研究室の秘書の方々ならびに先輩, 後輩のみなさまに感謝いたします．又, 一生懸命に日本での生活の支えになってくれた妻と，いつもの応援してくれる家族にも深く感謝いたします.

2015 年 12 月 1 日

参考文献

[1] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, “ Simple network management protocol (SNMP), ”RFC 1157, 1990.

[2] R. Enns, “ Network Configuration Protocol (NETCONF), ”RFC 6241, June 2011. https://tools.ietf.org /html/rfc6241

[3] ONF, “ Software-defined networking: The new norm for networks, ”White Papers, pp.3-6, ONF, 2012

[4] ETSI, “ Network Functions Virtualization - Introductory White Paper, ” http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[5] N.M.M.K. Chowdhury and R. Boutaba, “ Network virtualization: state of the art and research challenges, ”Communications Magazine, IEEE, vol-94, pp.20-26, 2009.

[6] K. Tutschku, T. Zinner, A. Nakao, and P. Tran-Gia, “ Network virtualization: Implementation steps towards the future internet, ”Electronic Communications of the EASST, vol-17, pp.1-14, 2009.

[7] ERICSSON, “ Managed services, ”2012. http://www. ericsson.com/ourportfolio/telecom-operators/managedservices

[8] T. Ylone, T. Kivinen,M. Saarinen,T. Rinne, and S. Lehtinen, “ SSH Transport Layer Protocol, ”draft-ietf- secsh-transport-16 (work in progress), July 2003.

[9] Tim Bray, et al. ”Extensible markup language (XML).” World Wide Web Consortium Recommendation, RECxml19980210, 1998.

[10] J. Schoenwaelder, Ed., ”Common YANG Data Types,” RFC 6991, 2013.

[11] McKeown, Nick, et al, ”OpenFlow: enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, 38.2, 69-74, 2008.

[12] Pradeep Sindhu, “ ChalkTalk on Software Defined Networks (SDN), ”Juniper’s Financial Analyst Day 2012, https://www.youtube.com/watch?v=VrOaZh7BZuI

[13] Phil Goldstein, “ Network outsourcing: Why the U.S. is an exception, ”FierceWireless, http://www.fiercewireless.com/special-reports/networking-outsourcing-why-u-s-exception

[14] MarketsandMarkets, “ Network Management Market by Network Performance Monitoring and Management, IP Address Management, Network Traffic Management, Network Device Management, Network Configuration Management, Network Security Management - Global Forecasts up to 2019, ”2015. http://www.rnrmarketresearch.com/global-network-management-market-performance-monitoring-and-management-ip-address-management-traffic-management-device-management-configuration-management-security-management-worldwide-market-market-report.html

[15] A.K.A. HAMID, Y. Kawahara, and T. Asami, “ Web cache design and implementation for efficient snmp monitoring towards internet-scale network management, ”IEICE Transactions on Communications, vol-94, no.10, pp.2817-2827, 2001.

[16] L. Richardson, and S. Ruby, “ RESTful web services, ”O’Reilly Media, Inc., 2008.

[17] Chang, Yanan, et al. ”Design and implementation of NETCONF-based network management system,” Future Generation Communication and Networking, 2008. FGCN’08. Second International Conference on. Vol. 1. IEEE, 2008.

[18] Cisco, “ Using the Command-Line Interface, ”2010. http://www.cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c/fcf001.html

[19] Internetworking, ”show running-config – show running-config isakmp.” http://www.cisco.com/c/en/us/td/docs/security/asa/asa82/command/reference/cmd_ref/s5.html

[20] Juniper NETWORKS, ”show configuration.” http://www.juniper.net/documentation/en_US/junos14.1/topics/reference/command-summary/show-configuration.html

[21] Internetworking, ”Cisco/Juniper Commands.” http://networking.ringofsaturn.com/Cisco/ciscojuniper.php

[22] R. Presuhn, J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, “ Management Information Base (MIB) for the Simple Network Management Protocol (SNMP), ”RFC 3418, 2002.

[23] K. McCloghrie, and M. Rose, “ Management Information Base for Network Management of TCP/IP-based internets:MIB-II, ”RFC 1213, 1991.

[24] ORACLE, “ SNMP エージェント MIB リファレンス, ” http://docs.oracle.com/cd/E23846_01/tuxedo/docs11gr1/snmpmref/1tmib.html

[25] ITU-T, “ Changing from ASN.1:1988 to ASN.1:2002, ” http://www.itu.int/ITU-T/studygroups/com17/changing-ASN/

[26] Wikipedia, “ Abstract Syntax Notation One, ” https://ja.wikipedia.org/wiki/Abstract_Syntax_Notation_One

[27] ITU-T, “ Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), ”X.209, 1988.

[28] ITU-T, “ Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), ”X.690, 2008.

[29] Microsoft, “ ASN.1 および BER の簡単な紹介, ” https://support.microsoft.com/ja-jp/kb/252648

[30] 緒方亮, 鈴木暢, and 矢野ミチル, “ マスタリング TCP/IP SNMP 編, ”株式会社 オーム社, 2005.

[31] M. Rose, and K. McCloghrie, “ Structure and Identification of Management Information for TCP/IP-based Internets, ”RFC 1155, 1990.

[32] K. McCloghrie, D. Perkins, and J. Schoenwaelder, “ Structure of Management Information Version 2 (SMIv2), ”RFC 2578, 1999.

[33] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser “ Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2), ”rfc1905, 1996/

[34] W. Stallings, “ Snmpv3: A security enhancement for snmp, ”Communications Surveys and Tutorials, IEEE, vol-1, no.1, pp.2-17, 1998.

[35] “ Weblogs.com ping server, ”http://www.weblogs.com/api.html

[36] T. Zitello, D. Williams, and P. Weber, “ HP OpenView System Administration Handbook, ”vol.1, 1 edition, Prentice Hall, 2004.

[37] “ HP Leads Convergent Mediation Market with HP OpenView IUM Software, ” http://www8.hp.com/us/en/hpnews/pressrelease.html ?id=302004

[38] “ HP Leads Industry in Revenue, Market Share of Worldwide Distributed Performance and Availability Management Software, ”http://m.hp.com/us/en/news/details.do ?id=170624&articletype=news_release

[39] J. Schoenwaelder, ”Overview of the 2002 IAB Network Management Workshop, ” RFC 3535, Internet Engineering Task Force-IETF, 2003.

[40] J. Schonwalder, A. Pras, J.P. Martin-Flatin, “ On the Future of Internet Management Technologies, ”IEEE Communications Magazine, October 2003, pp.90 97.

[41] Enns, Rob. ”NETCONF configuration protocol, ” RFC4741, Std. Track, 2006.

[42] Cisco, ”Using the XML Management Interface,” http://www.cisco.com/c/en/us/td/docs/switches/datacenter/sw/nx-os/xml/user/guide/nxos_xml_interface/using.html

[43] H. Xu, and D Xiao, ”Data modeling for NETCONF-based network management: XML schema or YANG,” Communication Technology, 2008. ICCT 2008. 11th IEEE International Conference on. IEEE, 2008.

[44] H, Cui, et al. ”Contrast Analysis of NETCONF Modeling Languages: XML Schema, Relax NG and YANG,” Communication Software and Networks, 2009. ICCSN’09. International Conference on. IEEE, 2009.

[45] C. Lonvick, ”The Secure Shell (SSH) Authentication Protocol,” RFC4252 (2006).

[46] M. Wasserman, “ Using the NETCONF Configuration Protocol over Secure Shell(SSH) ”, draft-ietf-netconf-ssh-00 (work in progress), Oct 2003.

[47] M. Rose, “ The Blocks Extensible Exchange Protocol Core ”, RFC 3080, March 2001.

[48] E. Lear, and K. Crozier, ”Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP),” RFC 4744, December 2006.

[49] T. Dierks and E. Rescorla, ”The transport layer security (TLS) protocol version 1.2,” RFC 5246, 2008.

[50] John G. Myers, ”Simple authentication and security layer (SASL),” 1997.

[51] Mein, Gunnar, et al. ”Simple object access protocol.” U.S. Patent No. 6,457,066. 24 Sep. 2002.

[52] A. Devlic, et al. “ A use-case based analysis of network management functions in the ONF SDN model. In Software Defined Networking (EWSDN), ”2012 European Workshop on, pp. 85-90, IEEE, October, 2012.

[53] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, “ On scalability of software-defined networking, ”Communications Magazine, IEEE, 51(2):136-141, February.

[54] A. Tootoonchian et al., “ On Controller Performance in Software-Defined Networks, ” Proc. USENIX Hot-ICE ’12, 2012, pp. 10-10

[55] M. Yu et al., “ Scalable Flow-Based Networking with DIFANE, ”Proc. ACM SIGCOMM 2010 Conf., 2010, pp. 351-62.

[56] A. R. Curtis et al., “ DevoFlow: Scaling Flow Manage- ment for High-Performance Networks, ”Proc. ACM SIG- COMM ’11, 2011, pp. 254-65.

[57] T. Koponen et al., “ Onix: A Distributed Control Platform for Large-Scale Production Networks, ”Proc. 9th USENIX OSDI Conf., 2010, pp. 1-6.

[58] 土屋 太二，山崎 里仁, “ 第９回　これからこうなるＳＤＮの技術と標準 SDN JAPAN 2013 報告（後編）, ”日経コミュニケーション，2014，01 号, pp. 68-75.

[59] ONF, “ OF-Config 1.2, ”TS-016, 2014

[60] Geoff Huston, “ Web Caching,” Cisco, The Internet Protocol Journal, Vol 2, No. 3, 2009.

[61] V. Jacobson, ”Congestion avoidance and control,” ACM SIGCOMM computer communication review, Vol.18, No.4, ACM, 1988.

[62] K. Kurata, G. Hasegawa, and M. Murata, “ Fairness comparisons between TCP Reno and TCP Vegas for future deployment of TCP Vegas, ”IEICE Technical Report. SSE, Exchange system, vol.99, no.428, pp.67-72, 1999.

[63] W.R. Stevens and K. Fall, “ TCP/IP Illustrated: The Protocols, vol.1, 2 edition, ” Addison-Wesley Publishing Company, 2011.

[64] L.S. Brakmo, S.W. O ’Malley, and L.L. Peterson, “ TCP Vegas: New techniques for congestion detection and avoidance, ”Proceedings, 1994 SIGCOMM Conference, pp.24-35, London, UK, Aug. 31st Sept. 2nd 1994.

[65] R. Jain, ”Congestion control in computer networks: Issues and trends,” IEEE Network, 4.3,24-30, 1990.

[66] M. Allman, V. Paxson, ”TCP Congestion Control,” rfc5681, 2009.

[67] S. Floyd, and V. Jacobson, “ Random early detection gateways for congestion avoidance, ”Networking, IEEE/ACM Transactions on, 1(4),pp.397-413,1993.

[68] Kevin Wallace, “ Cisco IP Telephony Flash Cards: Weighted Random Early Detection (WRED), ”Cisco Press, 2004.

[69] D. Chiu, and R. Jain, “ Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, ”Computer Networks and ISDN Systems, 17, 1-14, 1989.

[70] “ Network Simulator, ns-2, ”http://www.isi.edu/nsnam/ns

[71] NEC Unified Solutions, Inc, “ IP Requirements, ”pp.1, 2009. http://www.necdsx.com/docs/files/ip_overview/ip_requirements.pdf

[72] S. Sundaresan, W Donato, N. Feamster, R. Teixeira, and S. Crawford, “ Broadband internet performance: a view from the gateway, ”SIGCOMM, pp.134-145, ACM, 2011.

[73] NTT Communications, “ サービス品質保証制度 (SLA) 実測値, ”2015. http://www.ntt.net/service/sla_ps.cfm

[74] T. Oetiker, and D. Rand, “ MRTG: The Multi Router Traffic Grapher, ”In LISA, 141-148, 1998.

[75] Cisco, “ CPU しきい値通知の設定, ”2012. http://www.cisco.com/cisco/web/support/JP/docs/SW/WANSWT/BPX_IGX_IPXWANSW/RN/003/nm_cpu_thresh_notif.html?bid=0900e4b1825ae5d7

[76] KEL, “ 常に業界をリードする IT システム本社・有人店舗，自動契約機を結ぶ大規模ネットワーク網, ”http://www.kel.co.jp/case_introduction/network/promiss.html

[77] NEC, “ LAN ガイド: 大規模 LAN : LAN の構築モデル, ” http://www.nec.co.jp/octpower/seminar/languide/02_003.html

[78] B. Martin, A. Al-Shabibi, S. M. Batraneanu, M. D. Ciobotaru, G. L. Darlea, M. Ivanovici, “ Advanced monitoring techniques for a large-scale data-processing network, ” Campus-Wide Information Systems, 25(5), pp.287-300, 2008.

[79] Cisco, “ Solution Guide Managing Large-Scale Network Deployments, ”White paper, 2013. https://meraki.cisco.com/lib/pdf/meraki_whitepaper_large_scale_deployments.pdf

[80] 榊正彦, and 川口和穂, “ 光アクセス GE-PON システム MileStar (ネットワーク特集), ”沖テクニカルレビュー, 71.1, pp.84-87, 2004.

[81] Cisco, “ Web Cache Communication Protocol Version 2, ”2012. http://www.cisco.com/cisco/web/support/JP/docs/ANS/WideAreaAppServs/AppContentNWingSystem_ACN/CG/002/C_4070appb.html?bid0900e4b1825ae625

[82] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “ Hypertext Transfer Protocol – HTTP/1.1, ”RFC 2616, 1999.

[83] V. Cardellini, M. Cola janni, and P.S. Yu, “ Dynamic load balancing on web-server systems, ”Internet Computing, IEEE, vol.3, no.3, pp.28-39, 1999.

[84] 上谷一, 今野徹, and F5 ネットワークス, “ ロードバランサの本質, ”@ IT, 2003. http://www.atmarkit.co.jp/ait/articles/0302/05/news001.html

[85] VIDEO SPACE CALCULATOR, http://www.digitalrebellion.com/webapps/video_calc.html

[86] Akamai, “ The State of the Internet, “ http://www.akamai.com/dl/akamai/akamai-soti-q313.pdf?WT.mc_id=soti_Q313

[87] Digital Transmission Licensing Administrator, “ DTCP Volume 1 Supplement E Revision 1.2 (Informational Version), ”http://www.dtcp.com/, 2007.

[88] UPnP Device Architecture V1.1, http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1AnnexA.pdf

[89] R. Braden, D. Clark, S. Shenker “ Integrated Services in the Internet Architecture: an Overview, ”IETF, RFC-1633, June 1994.

外部発表論文

国内論文誌

[1] 宋泰永, 川原圭博, 浅見徹, “ 大規模ネットワーク管理のための HTTP/SNMP リバースプロキシの制御の公平性について, ”Vol.J98-B,No.12,pp.1238-1254,Dec. 2015.

国際会議

[2] T. Y. Song, Y. Kawahara, and T. Asami, “ Using SNS as Access Control Mechanism for DLNA Content Sharing System, ”Proc. 6th IEEE Conference on Consumer Communications and Networking Conference (CCNC 2009), pp.1-2, Las Vegas, Nevada, USA, Jan. 2009 (demo).

[3] 宋泰永, 川原圭博, 浅見徹, “ Cache Management Algorithm of Load Balancer for Large-scale SNMP Monitoring System, ”Globecom Workshops (GC Wkshps), IEEE, Atlanta, USA, 901-905, Dec. 2013.

国内研究会

[4] 宋泰永, 川原圭博, 浅見徹, “ ネットワークのパーソナル化と NaaS への期待’, ”信学技報, vol. 114, no. 7, IN2014-3, pp. 11-16, 2014 年 4 月.

[5] 宋泰永, 川原圭博, 浅見徹, “ 大規模ネットワーク管理のための HTTP/SNMP リバースプロキシの制御の公平性について’, ”情報ネットワーク研究会, 9 月, 2014.

全国大会

[6] 宋泰永, 川原圭博, 浅見徹, “ 大規模 SNMP モニタリングのための負荷分散システムの設計, ”信学総大, B-7-15, March 2013.

[7] 宋泰永, 川原圭博, 浅見徹, “ 大規模 SNMP モニタリングのための負荷分散システムの設計とキャッシュ管理アルゴリズム, ”信学総大, B-7-33, Sept. 2013.

[8] 宋泰永, 川原圭博, 浅見徹, “ パケットシェーパを用いた大規模 SNMP モニタリング の TCP 型トラ ヒック制御, ”全国大会, B-6-26, 3 月, March 2014.