卒業論文　　2000年度 (平成12年度)

ネットワーク運用支援ツールの設計と実装

慶應義塾大学 環境情報学部

氏名：海崎 良

指導教員

慶應義塾大学 環境情報学部

村井 純

卒業論文要旨 2000 年度 (平成 12 年度)

ネットワーク運用支援ツールの設計と実装

本研究では，ネットワーク運用を支援するトラフィックモニタリングツールの設計と実装を行い，その有用性を示した．

既存のトラフィックモニタリングツールは，ネットワーク上に流れる全てのパケットを視覚化するツールと予め設定したデータだけを視覚化するツールの 2 種類しか存在しない．そのためネットワークの詳細な情報を収集し，且つネットワーク変化を動的に検知することが出来ない．

本研究では，全てのパケットを収集し独自の集約手法を用いて，トラフィックの変化を動的に検知出来るツールを設計, 実装した．更に，収集したパケットの持つ多様な属性を視覚化することによって，多様な側面からネットワークを把握することを実現した．

本研究によって，ネットワーク上のトラフィック情報を印象値という独自の指標を用いて集約し，異なる時間粒度や異なる視点で視覚化することを実現した．

実際のネットワークでの測定実験を行い，本研究の実装が管理者がネットワークを運用していく上で非常に有用なツールであることを示した．

本研究の成果によって，ネットワーク運用を円滑に行われ，より安定したネットワークを提供することが出来る．

キーワード

1．ネットワーク運用 ,2．トラフィックモニタリング ,3．フローの動的検知 ,4．印象値，5．視覚化

慶應義塾大学 環境情報学部

海崎 良

Abstract of Bachelor's Thesis

Academic Year 2000

Design and Implementation of Monitoring Tool for Network Operation

This thesis proposes the design and implementation of a monitoring tool supporting fluent network operation.

There are two perspectives in types of network-monitoring tool.

1) Packet Capturing Type Network capturing tool capturing whole network packets.

2) Summary Type Monitoring tool supporting specific field or data.

In the field of Network management technology, there are no monitoring tool developed to support above.

New data aggrigation and visualization method is proposed in this thesis. As a result, dynamically monitoring of the network status has been made possible, and visualizing various characteristics of the network became easier.

Network monitoring testing is done at commodity network infrastructure. As a result, this tool is proven to be supportive for network operators. From this research, running network operation smoothly has been made possible.

Keywords :
1.network-operation, 2.traffic-monitoring, 3.auto-detect-flows, 4.impact-value, 5.visualize

Keio University , Faculty of Environmental Information

Ryo Kaizaki

目 次

第 1 章　序論 **1**

1.1 はじめに 1

1.2 本論文の目的 2

1.3 本論文の構成 2

第 2 章　ネットワーク運用とネットワーク情報 **3**

2.1 ネットワーク運用とネットワークモニタリング 3

2.2 ネットワークモニタリング技術の限界 4

2.2.1 ping 4

2.2.2 traceroute 5

2.2.3 SNMP 5

2.2.4 tcpdump 6

2.2.5 cflowd 7

2.3 本ツールの注目点 8

2.3.1 トラフィック量の多いフローの動的検知 8

2.3.2 異なるタイムスケールでネットワークを視覚化 9

2.3.3 異なる視点でネットワークを視覚化 11

第 3 章　本ツールの設計 **13**

3.1 トラフィックの多いフローの動的検知 13

3.1.1 データの収集 13

3.1.2 トラフィックの多いフローの抽出 18

3.2 異なるタイムスケールでネットワークを視覚化 23

3.3 異なる視点でネットワークを視覚化 27

3.3.1 収集したデータの階層化 27

3.3.2 異なる時間粒度によるサマリの作成 29

3.4 設計のまとめ 30

第 4 章　本ツールの実装 **31**

4.1 実装概説 31

4.2 データ抽出部 31

4.3 ツリー構造へのデータの保存 32

4.4 ツリー構造からのトラフィック量の多いフローの抽出 32

4.5 サマリデータの格納，更新 35
4.6 実装環境 36
4.6.1 ハードウェア環境 37
4.6.2 ソフトウェア環境 37

第 5 章 評価 **38**
5.1 測定環境 38
5.1.1 測定ネットワーク環境 38
5.1.2 測定時間帯 38
5.1.3 測定機環境 38
5.2 測定結果 39
5.2.1 定常的なネットワーク状態 40
5.2.2 特徴的なネットワーク状態 40
5.3 評価 41

第 6 章 結論 **43**
6.1 結論 43
6.2 今後の課題 43
6.3 本ツールの将来的展望 45
6.3.1 QoS ルーティングの支援 45
6.3.2 AS 間 SLA の締結を支援 45
6.3.3 ネットワーク設計の支援 46

図 目 次

2.1 ネットワーク運用のモデル図 3
2.2 ping 出力例 5
2.3 tracereoute 出力例 5
2.4 SNMP 概要図 6
2.5 tcpdump 出力例 7
2.6 cflowd 概要図 8
2.7 WIDE インターネット上のあるポイントにおけるトラフィック量 10
2.8 ポート別トラフィックデータ 12

3.1 MIB ツリー構造の一部 14
3.2 libpcap の機構概要 16
3.3 cflowd の export_data_format(version1) 17
3.4 データ収集によって構築された IP アドレスツリー 18
3.5 簡易ツリー 19
3.6 閾値に達しないリーフを集約して出来たツリー 19
3.7 閾値を越えたため分割されるツリー 20
3.8 印象度によって順位付けされたツリー 20
3.9 各ノードの値が近い場合 21
3.10 各ノードの値が遠い場合 22
3.11 各ノードの値が遠い場合の印象値 22
3.12 サーバ , クライアント間のサマリ更新 (手法 1) 24
3.13 サーバ , クライアント間のサマリ更新 (手法 2) 25
3.14 サーバ , クライアント間のサマリ更新 (手法 3) 26
3.15 IP アドレスツリーの階層化 28
3.16 プロトコル , ポートの階層化 28
3.17 異なる時間粒度のサマリ作成 30

4.1 libpcap を使ったデータ抽出 31
4.2 データ保持のための構造体 32
4.3 ツリー構造へのデータの保存 33
4.4 トラフィック量の多いフローの抽出 34
4.5 本実装において抽出される順番 35
4.6 サマリデータの格納 36

5.1 測定トポロジ概要 39
5.2 定常的なネットワーク状態 40
5.3 特徴的なネットワーク状態 41

表 目 次

2.1 本ツールと既存技術の比較 (動的なフロー検知) 9
2.2 トラフィック流量とネットワーク状態の検知 10

3.1 MIB アクセス項目数試算 15
3.2 フローの収集 17
3.3 トラフィックの多いフローの抽出 23
3.4 サマリ作成のタイミング相互比較 26
3.5 本ツールが作成する 8 つのツリー 29

4.1 ハードウェア環境 37
4.2 ソフトウェア環境 37

5.1 ハードウェア環境 39
5.2 測定機ソフトウェア環境 39

第1章　序論

1.1　はじめに

インターネットに接続されたネットワークを運用するには，管理者がネットワークの状態を把握し，ネットワークの現状に適した設定を行う必要がある．ネットワークの現状を把握するには，ネットワークを構成する機器に関する情報と，トラフィックの状態を把握する必要がある．ネットワークを物理的に構成する機器に関する情報は，設置された機器に関する情報を収集することで得られる．これらの情報は，例えばSNMP[1](Simple Network Management Protocol) などといった管理プロトコルにより取得することが可能である．しかし，ネットワーク上で流れているトラフィックを動的に把握することは，これらの自律した機器の情報を共有するという意味で非常に困難である．

現存するネットワークトラフィック解析ツールは以下の2種類に分類される．

- ネットワーク上に流れるパケット全てを見る
- ネットワーク上に流れるパケットの総量を見せる

tcpdump[2] に代表される前者のツールは，ネットワーク上の全パケットを収集，表示する．的確な状況判断を行うためには情報が詳細すぎるため非常に見にくく，情報が埋もれてしまう．また，多くの情報が出力されるため，通信の傾向を長時間に渡って調べるなどの用途には向かない．

MRTG[3](The Multi Router Traffic Grapher) に代表される後者のツールは，回線使用率が分かりやすく，ネットワーク全体のトラフィックの傾向を長時間に渡り観測可能である．しかし，全てのパケットが一つの数値として集約されるため，ネットワークに対する特定パケットの占有率といった種類の情報は得られない．MRTG によって得られる結果は，ある特定の回線上に流れたパケットの総数や，それにより消費された帯域だけである．

このように，現状のトラフィック解析ツールでは，詳細な情報を取得し，さらに長時間に渡るトラフィックの傾向を視覚化出来ない．

本研究では，以下に示される5つの成果を得られる．

本研究により作成されたトラフィック解析ツールを利用することで，1) ネットワークを占有しているトラフィックの傾向が分かる．また，2) トラフィックの傾向を把握することで，ネットワーク構築時の機器の構成や，セキュリティの向上が期待できる．

あわせて，3) ネットワークのどの部分にどのような機器を設定すれば効率的にスループットを向上させられるかが分かる．現状では，ネットワーク管理者がどのようにト

ラフィックが流れるているかを予想しながら機器の設定を行っている．本研究により作成されたツールを使用することで，4) 実際に流れているトラフィックを考慮したネットワーク機器の設定ができる．

本研究により作成したツールを利用することで，5) ネットワークセキュリティ対策の応用ツールとしての利用が可能である．現在，DoS(Denial of Service)[4] などの攻撃が増加しているが，実際に DoS 攻撃を受けているノード外での DoS 攻撃の検出は非常に困難である．本研究で作成したツールは，トラフィックの傾向なども即座に検出できるため，極端なトラフィックを被害者ノード以外で検出できる．それにより，DoS 攻撃を早期発見でき，被害を最小限に押さえられる．また，本ツールにより不審なトラフィックの増加も検出できるので，不正侵入なども事前に察知できる．

1.2 本論文の目的

本論文の目的はネットワーク運用を支援するツールを設計，実装することである．そのために，ネットワーク運用自体とネットワークを運用する際に必要となるネットワーク情報について議論する．特にネットワーク情報の収集，加工，ネットワーク運用への応用の 3 点に注目し， 既存のネットワークモニタリング技術では同時に満たすことの出来なかった以下に示す 3 つの条件を同時に満たすツールの実装，評価を行いネットワーク運用を支援する．

1. トラフィック量の多いフローを動的に検出
2. ネットワーク情報を異なるタイムスケールで視覚化
3. ネットワーク情報を異なる視点で視覚化

本論文で述べるネットワーク情報とは，ネットワーク上を流れるデータの特性や通信機器の情報と定義する．

また，本論文で述べるフローとは，あるホストからあるホストへ流れる一方向のパケットの集合，または同一の特徴をもった一方向に流れるパケットの集合と定義する．

1.3 本論文の構成

本論文は全 6 章で構成される．

第 2 章では，ネットワーク運用とネットワーク情報の関連について述べ，既存のネットワークモニタリング技術とその問題点について考察する．その上で，第 3 章では本論文で実装するツールの設計について，第 4 章では実装の手法について述べる．第 5 章では，本ツールの使用によってネットワーク運用にいかなる影響を与え得るか評価を行う．第 6 章では，本論文での結論とこれからの研究課題について論じる．

第2章　ネットワーク運用とネットワーク情報

本章ではネットワーク運用とネットワークから収集出来る情報の関連について述べる．その上で，既存技術の概要を解説し，本研究で特に注目した点に関して述べる．

2.1　ネットワーク運用とネットワークモニタリング

インターネット上には様々なユーザの嗜好に伴ったアプリケーションが送受信するデータの特性があり，ネットワーク構成要素の特徴を予測をすることが出来ない．例えば，Mosaic[5] から始まる WWW(World Wide Web) ブラウザの発達は，トラフィック全体に占める HTTP(Hyper Text Transfer Protocol) データの割合を急増させ，現在では全トラフィック量の 9 割に到達している [6]．

ユーザに対し快適で安定したネットワークを提供するためには，常にネットワーク全体の状況を把握した運用をする必要がある．しかし，ネットワーク全体の状況を把握するためには，トラフィック情報やフロー情報を統合的に収集，解析を行う必要があり，現在の解析手法では処理に要する時間的コストが大きい．実際のネットワーク運用の現場における解析データの利用は困難である．

図 2.1 に，ネットワーク運用時に行われる作業を段階的に分類しモデル化したものを示す．

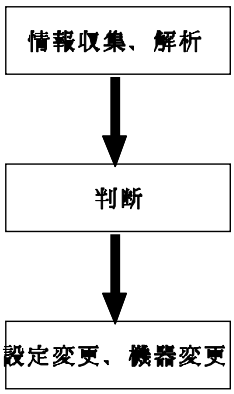

図 2.1: ネットワーク運用のモデル図

このようにネットワークの運用は，1)「情報収集，解析」2)「判断」3)「設定変更，

機器変更」の 3 段階に分類することが出来る．

障害の検知から回復までを考えた場合，障害を検知することが「情報収集，解析」に，障害の特定が「判断」に，障害回復作業が「設定変更，機器変更」にあたる．ネットワーク運用を行うことは，「情報収集，解析」「判断」「設定変更，機器変更」のサイクルを繰り返すことである．

つまり，ネットワークの現状に適した運用を行うには，このサイクルを早める必要がある．このサイクルを早める方法として，大きく分けて 3 つの手法が考えられる．

1. 「収集，解析するネットワーク情報」を出来るだけ詳細なものにし，且つ，直観的な理解を促す形で管理者に提供する
2. ネットワーク解析手法を確立することで「管理者によるネットワークに対する処置」を支援する
3. 「ネットワーク機器の設定変更」を抽象化する代理機構を構築する

本研究では，「情報収集，解析」に注目し，ネットワークの状況をいろいろな視点から捉える．その上で，ネットワーク管理者にとって直観的に理解しやすい形で提供することを目的とする．

2.2 ネットワークモニタリング技術の限界

ネットワークモニタリングとは，ネットワークから抽出された情報を見ることを指す．既存のネットワークモニタリング技術によって，ネットワークをさまざまな視点から見られる．この技術で得られた情報によって，管理者はネットワークに対する断片的な情報を知る．本項では，ping，traceroute，SNMP，tcpdump，cflowd について具体的な特徴を解説する．

2.2.1 ping

ping(Packet InterNet Groper)[7] を使うことによって，他のホストへ到達可能かどうかを調べられる．これは，ICMP(Internet Control Message Protocol) [9] エコー要求メッセージを対象となるホストに送信し，ICMP エコー応答が返ってくることを待つことによって実現されている．図 2.2 に ping の出力例を示す．

通常，あるホストから ping に対する応答が無い場合，そのホストへの到達性が無い．このことから，通常得られるサービスが受けられない場合に，ping を使うことによって何が問題であるかを知る出発点となる．また，あるホストまでの RTT(Round Trip Time:往復時間) を測定することによって，ネットワーク的な「距離」を知ることも出来る．

しかし，得られる情報が対象となるホストのインターフェース情報に限られているため，ping で得られた情報だけでネットワークを運用していくことは出来ない．

```
PING altavista.com (209.73.180.1): 56 data bytes
64 bytes from 209.73.180.1: icmp_seq=0 ttl=238 time=216.148 ms
64 bytes from 209.73.180.1: icmp_seq=1 ttl=238 time=226.340 ms
64 bytes from 209.73.180.1: icmp_seq=2 ttl=238 time=202.153 ms
— altavista.com ping statistics —
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 202.153/214.880/226.340/9.915 ms
```

図 2.2: ping 出力例

2.2.2 traceroute

traceroute[8] を使用することによって，あるホストから他のホストにパケットが流れる際の経路を確認することが出来る．traceroute は IP ヘッダの TTL フィールドを用いて実装されたツールである．traceroute は対象となるホストに対して TTL=1 のパケットを送信し ICMP の時間超過メッセージを期待する．次に TTL=2 のパケットを送信し，ICMP の時間超過メッセージを期待するという処理を繰り返す．その際，送信するパケットの宛先 UDP ポートに存在しない番号 (30，000 番以上) を選択し，目的ノードに到達した場合に ICMP のポート到達不可が帰って来るのをを期待する．traceroute はこの 2 種類の ICMP メッセージを用いて目的ホストまでの経路を確認する．図 2.3 に traceroute の出力例を示す．

```
traceroute to kame212.kame.net (203.178.141.212), 30 hops max, 40 byte pack-
ets
1 rg-gate (203.178.143.1) 0.529 ms 0.475 ms 0.555 ms
2 hitachi2.fujisawa.wide.ad.jp (203.178.137.70) 1.684 ms 1.476 ms 1.321 ms
3 gr2000.karigome.wide.ad.jp (203.178.141.193) 1.533 ms 1.664 ms 1.531 ms
4 kame212.kame.net (203.178.141.212) 1.139 ms 1.320 ms 1.304 ms
```

図 2.3: tracereoute 出力例

しかし，インターネットにおいてはネットワークが不安定な場合には，同じ発信元から同じ宛先に向かうパケットが同一ルートを経由するという保証はないため，ネットワーク状況によって結果が異なることがある．また，経路の変化をネットワーク管理者が検知した場合でも，その原因を traceroute だけでは判別出来ない．

2.2.3 SNMP

SNMP(Simple Network Management Protocol) は，ネットワークの情報管理プロトコルである．本プロトコルによってネットワーク機器それぞれが MIB(Managemanet

Infomation Base)[10] の形式に基づいて収集したデータを収集，管理出来る．図 2.4 に SNMP の概要図を示す．

SNMP は，各機器ごとにデータを収集，保存する機構を持っている．SNMP を使用し情報の要求を行った場合，各機器それぞれが MIB からデータを抽出し収集したデータを要求元に返信する．

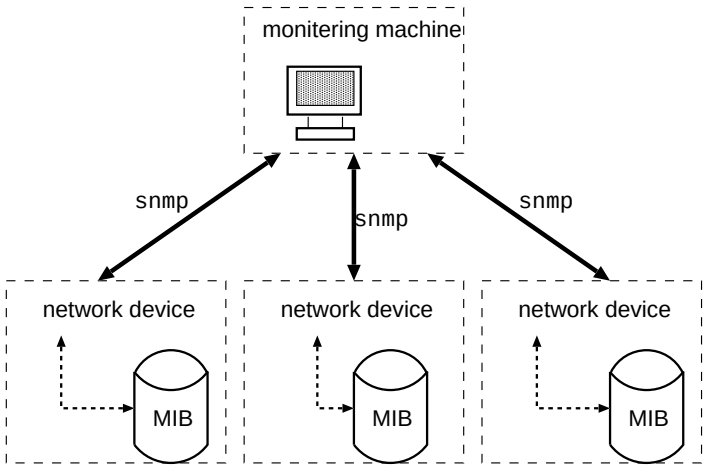

図 2.4: SNMP 概要図

SNMP を使用して分散した各ネットワーク機器が保持するデータを，管理者が一元的に収集することで情報の収集，解析を行える．そのため，動的にネットワーク機器の情報を収集することが容易となる．しかも，MRTG，RRDtool[11]，cricket[12] などに代表される SNMP を使用した可視化ツールが数多く存在しているため，それらのツールを使用して収集した情報を可視化することで，管理者が直観的にネットワークの状況を理解出来る．

しかし，予め設定しておいたデータしか収集することが出来ないため，管理者の予期しないネットワーク状態を検知出来ない．

2.2.4 tcpdump

tcpdump は libpcap[18] を使用したパケット収集ツールである．図 2.5 に tcpdump の出力図を示す．

tcpdump は，起動したホストが受け取れる全てのパケットを収集出来る．そのため, 送信先アドレス, 宛先アドレス，プロトコル，データ長などさまざまな情報を収集出来る．

しかし，収集出来る情報が多いにも関わらず，収集したの情報を集約する機能が付加されていない．そのため情報量が膨大になり，管理者がネットワークの状況を直観的に理解するのが困難である．

また，tcpdump によって収集されたデータを加工，分析することでネットワークの

```
17:47:57.645226 0800 62: 203.178.143.201.1967 > 203.178.140.130.21: S
192976724:192976724(0) win 8192 < mss 1460,nop,nop,sackOK > (DF)
17:47:57.652837 0800 60: 203.178.140.130.21 > 203.178.143.201.1967: S
2786713294:2786713294(0) ack 192976725 win 17520 < mss 1460 > (DF)
17:47:57.653074 0800 60: 203.178.143.201.1967 > 203.178.140.130.21: . ack 1
win 8760 (DF)
17:47:57.712136 0800 113: 203.178.140.130.21 > 203.178.143.201.1967: P
1:60(59)ack 1 win 17520 (DF) [tos 0x10]
```

図 2.5: tcpdump 出力例

状況の大部分を理解出来るが，どの視点で加工，分析するかという肝心な部分が管理者のネットワークに対する前提知識に強く依存してしまうため，ネットワークの運用という観点で動的なネットワークモニタリングツールでは無い．

2.2.5 cflowd

cflowd[13] は CAIDA[14] の開発したネットワークモニタリングツールである．cflowd は Cisco システムズ社 [15] 製ルータを通過したパケットをフロー毎に集約し，以下に示すデータを含んだネットワーク情報を生成する．しかも，これらのデータを多地点から収集，集約する機構も備えている．

cflowd において，フローは送信元アドレス，宛先アドレス，送信ポート，受信ポートが同一のデータの集合と定義されている．

図 2.6 に，cflowd の概要図 [13] を示す．

- フローの開始時間，終了時間
- パケット数
- 総データ長
- 送信元アドレス，宛先アドレス
- 送信ポート，受信ポート

しかし，flow 機能は Cisco システムズ社の特定の機器，特定の IOS[16] のみで動作が保証されており，さまざまなベンダの通信機器によって構成されるネットワークには対応出来ない．しかも，フローを集約するアルゴリズムなどが示されていないため，同一の機能のツールの開発が困難である．

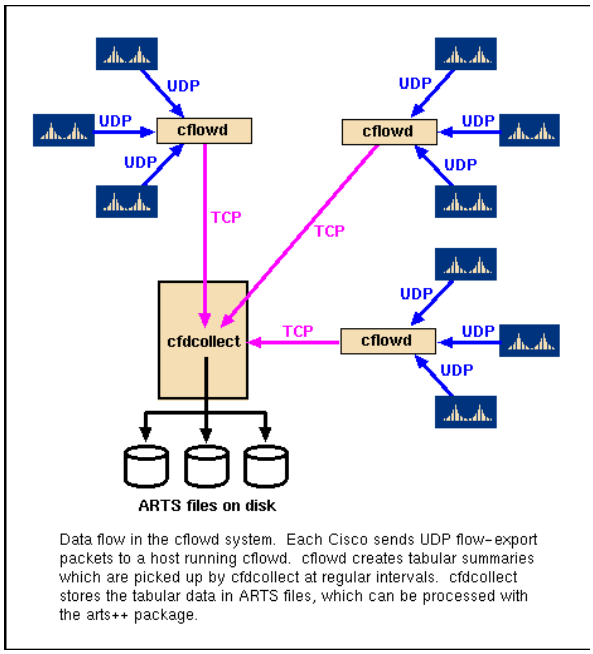

図 2.6: cflowd 概要図

2.3 本ツールの注目点

本論文ではネットワーク情報を収集し，ネットワークの状態をさまざまな視点から着目し，ネットワーク運用を支援する目的で加工することを目標としている．その中で，本論文の中で特に注目した 3 つの特徴を示す．

- トラフィック量の多いフローの動的検知
- 異なるタイムスケールでネットワークを視覚化
- 異なる視点でネットワークを視覚化

本節では，これら 3 つの特徴について個別に取り上げ論じ，この 3 つの特徴を満たすことで得られる長所をまとめる．

2.3.1 トラフィック量の多いフローの動的検知

運用者はフローを動的に検知することで，予期しないネットワーク状態を検知出来る．更に，ネットワーク運用者にとって，既に検知，予期出来るフローよりも，予期していないフローの方が情報としての価値が高い．

例えば，管理対象にあるネットワークのホストが DoS(Denial of Service) 攻撃を受けていた場合，既存技術ではこの状態を検知出来ない．

なぜなら，SNMP を使用してフローを検知する場合，予め設定したおいたフローしか検知出来ないからである．つまり，SNMP を使ってトラフィック量の多いフローを動

的に検知するためには常に全てのフローを監視し続ける必要がある．そのため，ネットワーク運用に役立てる手法としては現実的でない．

また，tcpdump に代表される技術を使用した場合，全てのネットワーク情報を予め収集し，加工を加える手法でもトラフィックの多いフローを検知することは出来る．しかし，これを動的に行う手法が確立されていないため，検知したネットワークの状態をネットワーク運用に反映させる時には，既にネットワークの状態が変化している可能性が高い．

表 2.1 に，これまでに述べた 3 つの手法の違いについてまとめる．

表 2.1: 本ツールと既存技術の比較 (動的なフロー検知)

	情報を収集する対象	情報更新のタイミング
本ツール	全てのフロー	動的
SNMP 型	予め設定したフローのみ	動的
tcpdump 型	全てのフロー	静的

表 2.1 中で使われている用語は以下の意味で用いている．

- SNMP 型
  SNMP を使用して，通信機器から予め設定した情報を定期的に収集するタイプのトラフィックモニタリング技術

- tcpdump 型
  ネットワークに流れるデータを全て収集して，静的に加工することの出来るタイプのトラフィックモニタリング技術

2.3.2 異なるタイムスケールでネットワークを視覚化

異なるタイムスケールでネットワークを視覚化することで，現在のネットワーク状況が定常的なものであるか，一時的な状況であるかを判断出来る．

例えば，次のページに示す図 2.7 は WIDE プロジェクト [17] が管理するネットワークのある地点のポイントにおける 1999 年 5 月 11 日から 5 月 15 日までのトラフィックを流量別にまとめたものである．この回線は 5 月 13 日 16 時に回線帯域が 3Mbps から 11Mbps に増強された．

図 2.7 の 5 月 13 日 0 時付近のトラフィック量を見ると回線帯域使用率がかなり高まっていることは分かる．しかし，回線帯域使用率の高い状態が定常的に続いているとは言えない．

この場合，長期的にトラフィック量を取得しつづけることによってこの回線の利用率が定常的に高いことが分かった．更に，5 月 13 日 16 時に回線帯域が増強された後は，一日の中でも時間によって帯域利用率が大幅に変動することが確認出来た．

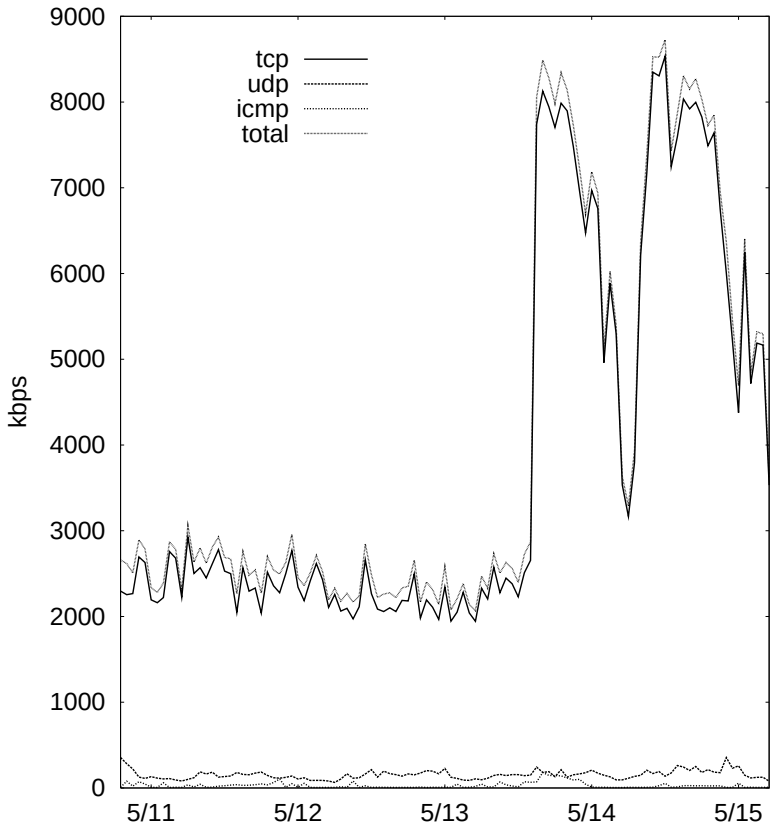

図 2.7: WIDE インターネット上のあるポイントにおけるトラフィック量

このように，トラフィック量を例に取った場合でも，異なるタイムスケールで視覚化することで，捉えられることの出来るネットワークの状況は異なる．表 2.2 に，トラフィック量を異なるタイムスケールで視覚化することで検知出来る主なネットワークの状態をまとめる．

表 2.2: トラフィック流量とネットワーク状態の検知

タイムスケール	検知出来るネットワーク状態
短期的 (1 時間程度まで)	攻撃，ネットワーク機器の故障，設定ミス
中期的 (日，週)	時間，曜日によるトラフィック量の増減傾向
長期的 (月，年)	ユーザ数の増減傾向

短期的にトラフィック量が激減した場合，そのネットワークの機器のどれかが正常に動作しなくなったと想像出来る．逆にトラフィック量が激増した場合，悪意のあるデータの流入である可能性を検知し，更に詳しい解析を行うきっかけとなる．

中期的なトラフィック傾向を視覚化することで，23 時を越えるとトラフィック量が増加するといった時間的なパターンを検知出来る．また，日曜日にトラフィック量が減少

するといった週の中でのパターンを検知出来る．
　長期的なトラフィック量の増減を視覚化することで，管理しているネットワークの回線増強や機器の設定，設置の時期を推測出来る．

2.3.3　異なる視点でネットワークを視覚化

　ネットワークに流れるデータフローは，以下に示すような属性を持っている．

- データの通過した時間
- パケット数
- データ長
- 送信元 IP アドレス
- 宛先 IP アドレス
- 送信ポート
- 受信ポート
- プロトコル
- IP version
- TTL

　ネットワークの状態は上記の属性によって多様に変化する．しかし，ネットワークを運用していく上で上記の全ての属性を常に監視し，ネットワーク運用に反映させていくことは困難である．
　図 2.8 に示したグラフから，以下のことが分かる．

- HTTP のデータが TCP のデータの大部分を占めている
- BGP のデータは時間によって流量の変化が少ない
- FTP データはデータ量が不規則且つ急激に変化する

　このように，データフローの一つの属性からだけでもネットワーク運用を行っていく上で有用な情報を抽出出来る．しかし，このような解析をリアルタイムにさまざまな属性に対して行っていくことは不可能である．
　そこで，本ツールでは短時間毎にデータ長に注目し流量の多いデータフローを選別する．その上で，選別したフローそれぞれに関して送信元アドレス，宛先アドレス，プロトコル，ポート番号の属性を参照出来るようにする．これは，ネットワークを運用していく上で，データ量の多いデータフローはネットワーク全体に与える影響が強いからである．以下に本ツールで注目したフロー特性とその主な理由を示す．

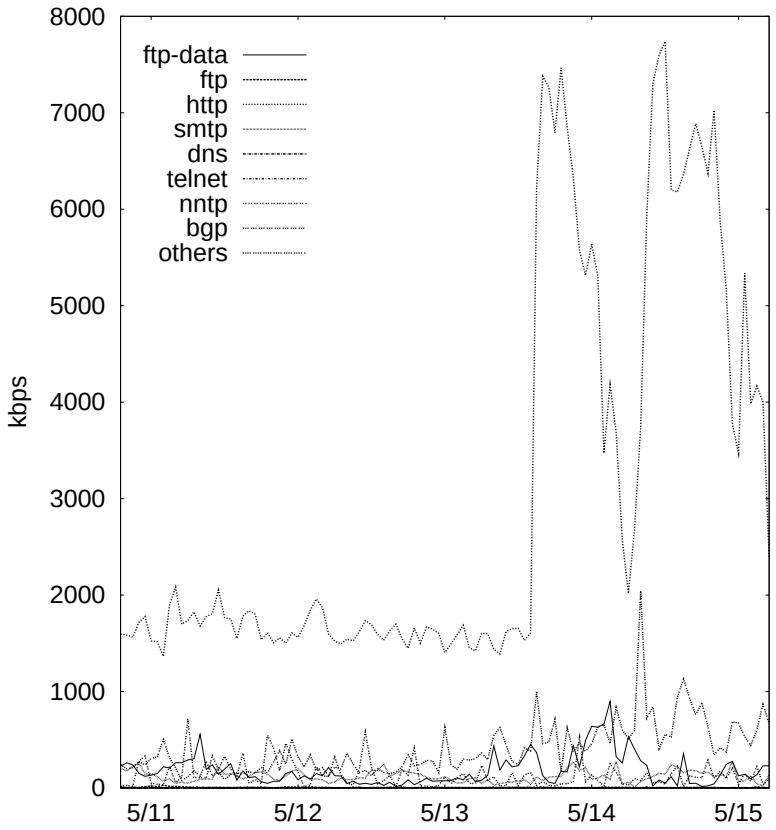

図 2.8: ポート別トラフィックデータ

1. 送信元アドレス

   短期的には攻撃元が分かる．中，長期的にはネットワーク機器の構成変更のための有効な情報となる．

2. 宛先アドレス

   短期的には攻撃対象が分かる．中，長期的にはネットワーク機器の構成変更のための有効な情報となる．

3. プロトコル

   短期的には輻輳制御 [19] を行っているデータの割合が分かる．長期的には通信機器の設計のための有効な情報となる．

4. port 番号

   短期的には DoS 攻撃の検知．中，長期的にはアプリケーションの流行が分かる．

第3章　本ツールの設計

本章では，本ツールの目的であるトラフィックの多いフローを動的に検知し，ネットワーク情報を異なるタイムスケールや異なる視点で視覚化する機能を実現する手法について議論を行う．

3.1　トラフィックの多いフローの動的検知

3.1.1　データの収集

本ツールの要求を満たすためには，それぞれのフローに関して以下に示す情報をネットワークから抽出出来なければならない．この論拠に関しては第 2.3.3 項で述べた．更に，実際のネットワーク運用の際に出来るだけ人的，時間的コストが少ない手法が良い．

- データの通過した時間
- パケット数
- 総データ長
- 送信元 IP アドレス
- 宛先 IP アドレス
- 送信ポート
- 宛先ポート
- IP プロトコル
- IP version

これらの情報を抽出出来る手法として 3 つ方法が考えられる．

1. SNMP
2. libpcap
3. cflowd

上記 3 つの概要については 2 章で述べた．

以下，これら 3 つの手法を比較して本ツールの要求をみたす一番良い手法を選択するための議論を行う．

SNMP を使用する場合

SNMP を使用して情報を収集する場合，通信機器の MIB と呼ばれるツリー型の構造を持ったデータベースにアクセスをしてカウンタ情報を収集する．図 3.1 に，このツリー型データベースの一部を示す．

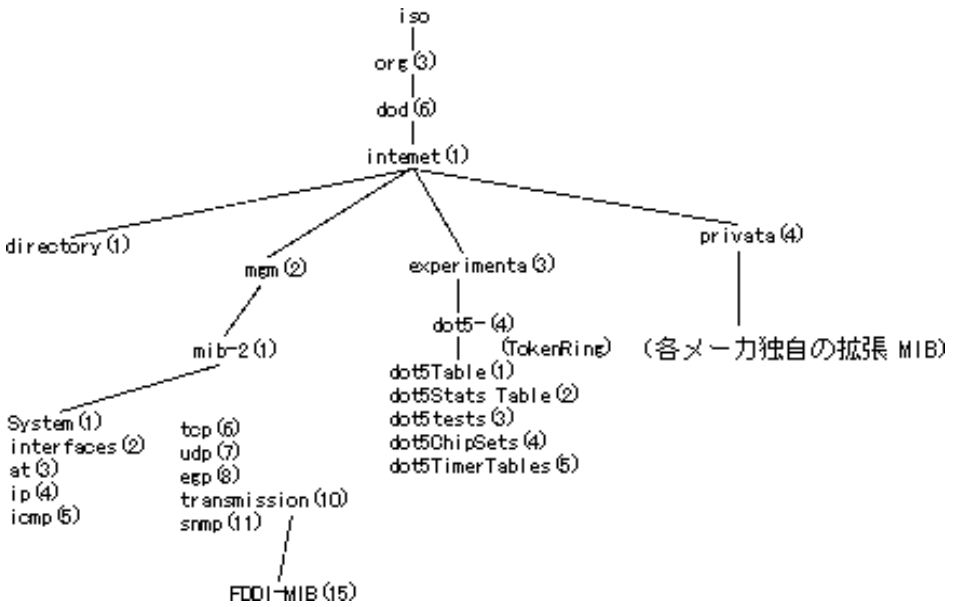

図 3.1: MIB ツリー構造の一部

図 3.1 のそれぞれの項目についている数字 (括弧内) は SNMP が MIB にアクセスする際の識別子として使用される．例えば，特定のインターフェースに流れるデータ量を取得するためには MIB の [1.3.6.1.2.1.2] にアクセスする．このように，本ツールが要求する情報はカウンタの形で取得出来る．しかし，本ツールは特定のフローだけではなく，全てのフローの情報を取得しなければならないので，ある特定時間に大量の MIB 項目にアクセスする必要がある．この概算を以下の表 3.1 に示す．

表 3.1 に示した項目を計算すると以下のようになる．

$$
\begin{aligned}
&Total \\
&= (Packet + Length) \times (protocol + src_addr + dst_addr + src_\text{ポート} + dst_port) \\
&= (1+1) \times (4 + 2^{32} + 2^{32} + 2^{16} + 2^{16}) \\
&= 2 \times (4 + 4294967296 + 4294967296 + 65536 + 65536) = 17180131336
\end{aligned}
$$

- Total： 総 MIB アクセス項目数 (IPv4 のみの場合)

表 3.1: MIB アクセス項目数試算

データ項目	項目数	説明
データの通過した時間	なし	データを取得した時間を保持するため必要なし
パケット数	1	
総データ長	1	
送信元 IP アドレス	2^{32}	IPv4 に限った場合
宛先 IP アドレス	2^{32}	IPv4 に限った場合
送信ポート	2^{16}	
宛先ポート	2^{16}	
IP プロトコル	4	最低でも tcp，udp，icmp，egp の 4 つは必要
IP version	1	IPv4 のみの試算のため

- Packet： パケット数
- Length： データ長
- protocol： プロトコル
- src_addr： 送信元 IP アドレス
- dst_addr： 宛先 IP アドレス
- src_port： 送信元ポート
- dst_port： 宛先ポート

上記の試算式に示したように SNMP を使ってトラフィックの情報を収集した場合，膨大なデータを収集しなければならない．これは，「値が 0」のデータも全て収集しなければならないためである．

SNMP を使った場合，SNMP 機能をサポートしている通信機器全てを対象にデータを収集出来る．また，単一のホストからの情報も SNMP のデーモンを立ち上げることでデータの収集が可能となる．

libpcap を使用する場合

libpcap はパケットを取り込むためのライブラリである．libpcap を使用することで OS が提供する下位 (データリンク層) のパケット取り込み機構を隠蔽出来る．図 3.2 に libpcap の概要を示す．

libpcap の機構の中で特に注目するべき点は，パケットを可変長でキャプチャー出来ることである．例えば Ethernet で IPv4 のデータを収集する場合，データリンクヘッダ (Ethernet ヘッダは 14 バイト)，IP ヘッダ (IPv4 の場合 20 バイト)，トランスポート

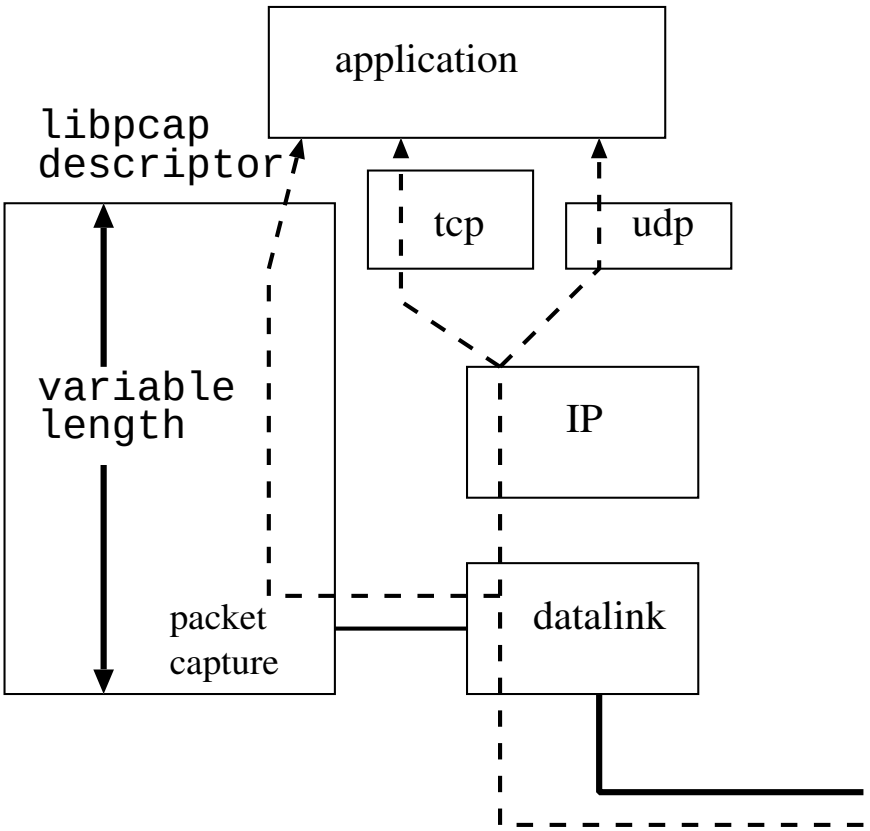

図 3.2: libpcap の機構概要

ヘッダ (TCP ヘッダは 20 バイト) の合計 54 バイト以上の長さでデータを収集すれば，本ツールが要求する情報を収集出来る．

libpcap は root 権限のある単一ホストで利用出来る．つまり，通信機器から直接収集することは出来ない．しかし，データを収集したい通信機器のポートミラーリングを行うことによって，データ収集の対象となる機器を通過するデータと全く同じものを収集出来る．

cflowd を使用する場合

cflowd が必要とする flow 機能は新しい IOS のバージョンの Cisco システムズ社製ルータに実装されている機能である．例えば cisco4500 の場合，IOS12.0(3) とそれ以降のバージョンの IOS でサポートされている．この機能は，フローを集約して定期的にモニタリングマシンにデータを送信する作業を実現している．図 3.3 に，cflowd が送信する情報のフォーマットを示す．

cflowd を使用した場合，本ツールが要求する情報は全て収集出来る．その際，収集される情報は既にフローとして集約されているため本ツールが後で加工する必要性が低い．

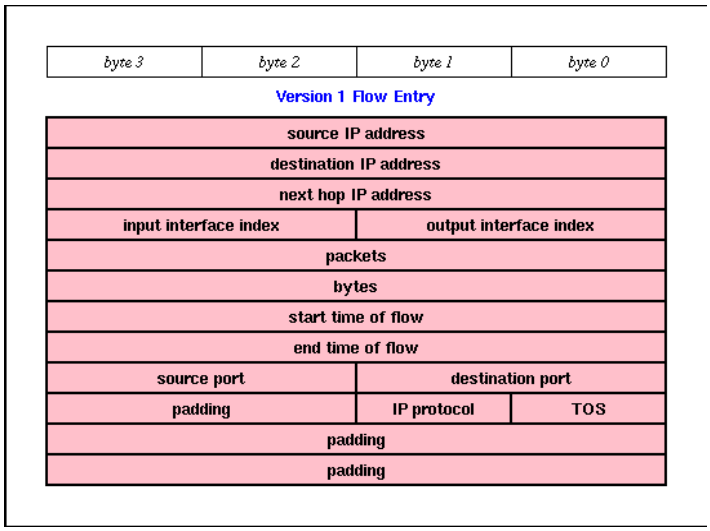

図 3.3: cflowd の export_data_format(version1)

また，cflowd は限られたベンダの限られた IOS でしか動作しない．しかし，大多数のネットワークはさまざまなベンダのさまざまな通信機器で構成されている．このことから，ネットワークの運用という立場で見た場合，cflowd を使用して本ツールを作成するのは最も良い選択肢では無い．

比較検討

この章では本ツールが情報を収集する際に選択すべき手法について議論してきた．表 3.2 に，議論の内容をまとめる．

表 3.2: フローの収集

	収集するデータ	情報収集可能な対象
SNMP	膨大な数の MIB 項目	SNMP をサポートしている通信機器，ホスト
libpcap	NIC で検知出来る全てのパケット	root 権限のあるホスト
cflowd	集約された全てのフロー	flow 機能をサポートした Cisco 社製 Router

上記の手法は，全て本ツールが要求する情報を提供出来る．しかし，現実的にネットワークの運用において使われる場合を想定してみると次の 2 つのことが言える．

1. SNMP を使用した場合，監視しなければならない対象が多くなり過ぎる
2. cflowd を使用した場合，多くのネットワークは cisco 製の機器への変更を迫られる

以上の論拠により，本ツールでは情報の収集手段として libpcap を使用する．

3.1.2 トラフィックの多いフローの抽出

本ツールではデータ収集の手段として libpcap を選択した．libpcap を使用することで機器のインターフェースに流れる全てのパケットを収集することが出来る．しかし，全てのパケットを収集しただけでは，トラフィックの多いフローを抽出したことにはならない．

本ツールでは送信元アドレス，宛先アドレスがツリー構造で形成されるアドレス空間の一部であることに注目して，複数のフローを集約したフローの傾向も抽出する．個々としては少ないデータ量の少ないフローであるが集合として見た場合に特徴を示すフローを検出するためである．

IP アドレスツリーを例にとって説明する．図 3.4 に「203.178.143.54」を送信元アドレスとする 80 バイトのパケットを収集した例を示す．

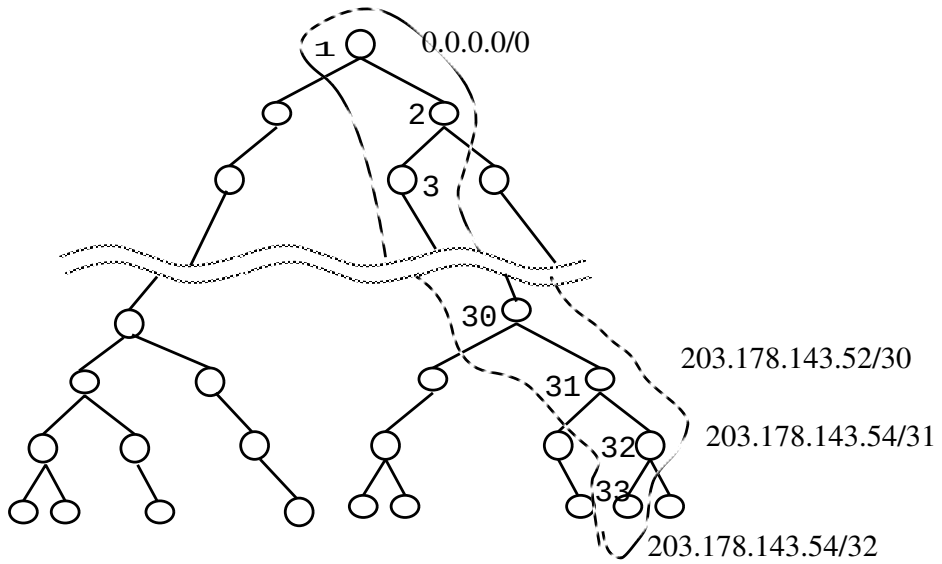

図 3.4: データ収集によって構築された IP アドレスツリー

1. 0.0.0.0/0 を送信元とするパケットが 80 バイト通過した
2. 128.0.0.0/1 を送信元とするパケットが 80 バイト通過した
3. :
4. :
:
31. 203.178.143.52/30 を送信元とするパケットが 80 バイト通過した
32. 203.178.143.54/31 を送信元とするパケットが 80 バイト通過した

33. 203.178.143.54/32 を送信元とするパケットが 80 バイト通過した

このように libpcap を使用して収集される情報を一定時間積み重ね，ツリー構造を用いて表す．この図 3.4 の斜線で囲まれた部分に，上の例で示したパケットは含まれる．

このツリー構造の中からトラフィックの多いフローを抽出するためには独自の加工が必要となる．

独自の加工には 3 つの手法が考えられる．これより，図 3.5 に示す簡易ツリーを用いて議論を行う．その上で，3 つの手法について述べ，本ツールが採用する手法とその論拠を述べる．

- ノード抽出手法 **1**：閾値に達しないフローを集約する手法
- ノード抽出手法 **2**：閾値を越えたフローを分割する手法
- ノード抽出手法 **3**：独自の指標によって順位付けする手法

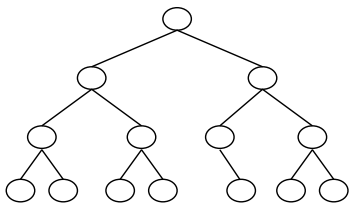

図 3.5: 簡易ツリー

ノード抽出手法 **1**：閾値に達しないフローを集約する手法

ノード抽出手法 **1** は収集されたデータに対し，ある閾値に達しないものは順次集約してゆく手法である．

図 3.6 に，この手法によって形成されるツリーの状態を示す．

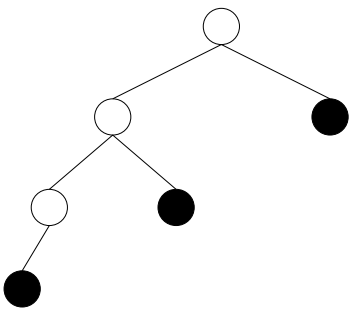

図 3.6: 閾値に達しないリーフを集約して出来たツリー

ノード抽出手法 **1** による特徴を示す 2 つ示す．

- トラフィック量の少ないフローが消失する
- 着目されるフローが互いに独立している

ノード抽出手法 **2** ： 閾値を越えたフローを分割する手法

ノード抽出手法 **2** は，完全なツリーを前提としない．ツリーを root ノードから作ってゆき，あるノードの値が閾値を越えた時に，子ノードを作成する．
図 3.7 に，この手法で形成されるツリーを示す．

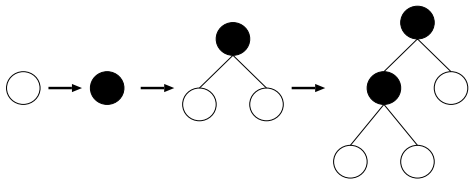

図 3.7: 閾値を越えたため分割されるツリー

ノード抽出手法 **2** を用いた場合，最終的に形成されるツリーはノード抽出手法 **1** を用いた場合と同様である．ノード抽出手法 **2** を用いた場合，手法 1 に較べてツリー自体が占有する記憶領域は少ない．しかし，ツリーを最終段階まで発展させるための計算回数が多い．

ノード抽出手法 **3** ： 独自の指標によって順位付けをする手法

ノード抽出手法 **3** は，ツリーを集約せず，ツリーの特徴的な部分だけを抽出する手法である．図 3.8 に，この手法で形成されるツリーを示す．

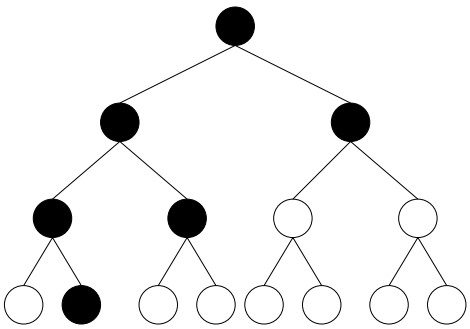

図 3.8: 印象度によって順位付けされたツリー

ノード抽出手法 **3** では，ツリーの特徴的なノードを抽出するために独自の指標を設ける．この指標を「印象値」と定義する．この印象度の高いもを順次抽出することでツリーの特徴的なノードを抽出する出来る．

この印象値の算出には以下の式 3.1 を用いた．

$$f(n) = T \times I^n \tag{3.1}$$

f(n): あるノードにおける印象値

T : あるノードにおけるデータ量，パケット数

I : 印象度

n : ノードの世代数 (ノードの親の数)

図 3.9，図 3.10，図 3.11 を例に取って式 3.1 の説明を行う．尚，図中に示した数値は各ノードが持つ値である．

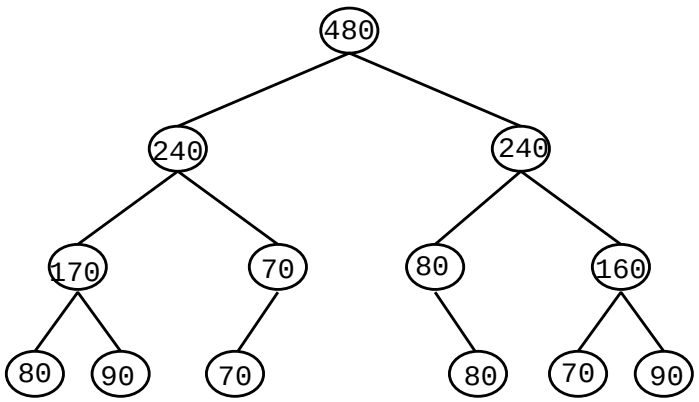

図 3.9: 各ノードの値が近い場合

図 3.9 に示したような，同じ階層のノードが持っている値が比較的近い場合，総データ量で各ノードを比較してツリーの特徴を抽出する手法が良い．

図 3.10 に示したような，特定のノード (この場合は a) が同じ階層の他のノードに較べて非常に大きい場合，このツリーの中で最も特徴的な部分はノード a である．しかし，総データ量で各ノードを比較した場合にはノード a に依存してデータ量の大きくなったノード b，ノード c，ノード d が抽出されてしまう．

この問題を解決する手法として，階層が低いノードは階層が高いのノードに較べてなんらかのアドバンテージを持たせるという方法を採用した．

この階層の上下一段階に存在するアドバンテージを印象度とし，この印象度とデータ量によって算出された印象値によって，各ノードの値が不均衡なツリーから特徴的なノードを抽出する．図 3.11 に，この印象値という独自の指標を用いて，ツリー中の

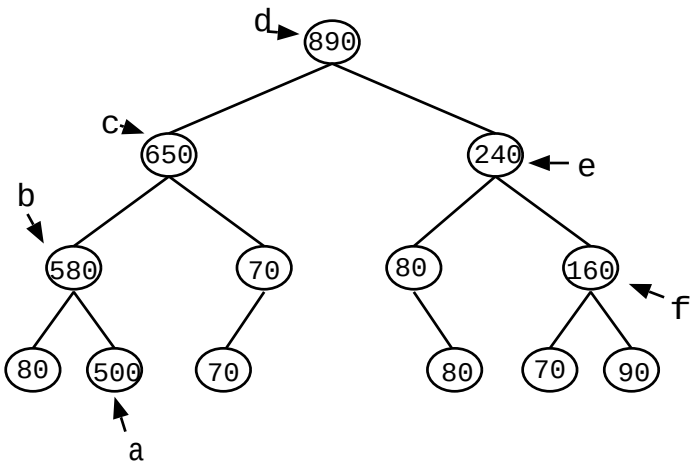

図 3.10: 各ノードの値が遠い場合

各ノードに重み付けした例を示す．この例のでは，印象度を「2」として印象値を算出した．

「印象値」は，ノードの保持する「データ量」に root から数えた世代数回「印象度」を掛けて算出する．

例えば，図 3.11 のノード a の印象度は「500(データ量)」に「2(印象度)」を「3(root ノードから数えた世代数)」回掛けた結果，「4000(印象値)」が算出される．

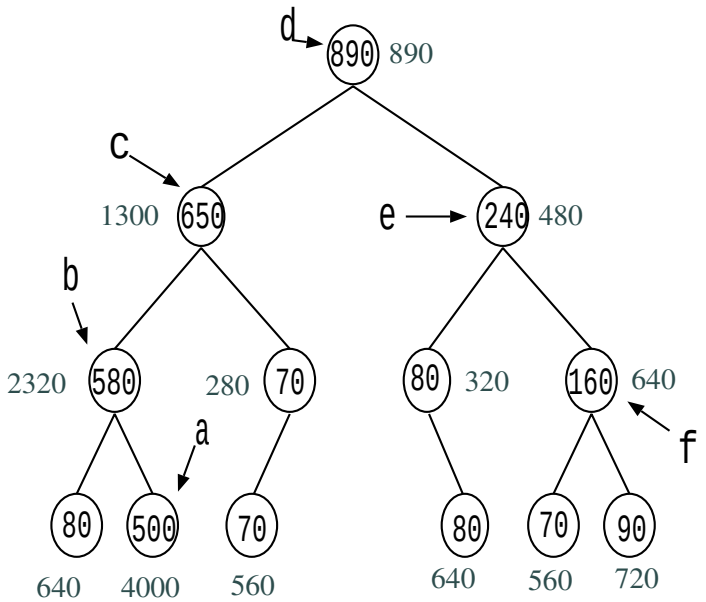

図 3.11: 各ノードの値が遠い場合の印象値

図 3.11 の各ノードの横に書かれた数字は，各ノードの印象値を示している．この各ノードの印象値を比較することで，ツリー中のデータ量の多いノードを抽出出来る．

この手法を用いることで，ノード b，ノード c，ノード d よりも値の小さいノード a に注目することが出来る．しかも，ノード e，ノード f の上下関係も保てる．

比較検討

本節では本ツールがトラフィックの多いフローを抽出する際に選択すべき手法について議論してきた．表 3.3 に，議論の内容をまとめる．

表 3.3: トラフィックの多いフローの抽出

	中間ノードの抽出	ツリー構成のための処理回数
ノード抽出手法 1	不可能 (リーフノードのみ)	少
ノード抽出手法 2	不可能 (リーフノードのみ)	多
ノード抽出手法 3	可能	多

上記の手法は，全てネットワークの特徴的な状況を抽出出来る手法である．しかし，ノード抽出手法 **1**，ノード抽出手法 **2** ではツリーの中の中間ノードを抽出出来ないため，トラフィックの多いフローを階層的に注目出来ない．

よって，本ツールではトラフィックの多いフローを抽出する手法としてノード抽出手法 **3** を採用する．

3.2 異なるタイムスケールでネットワークを視覚化

前節において，ネットワークの情報を収集しトラフィックの多いフローを抽出する手法について議論した．

この手法をある単位時間ごとに用いることで，その単位時間におけるネットワークの特徴を抽出出来る．本論文では，このネットワークの特徴を記述したものを「サマリ」と定義する．

このサマリが保持すべき情報と，サーバ，クライアント間でサマリを交換する方法を考えた場合，3 つのモデルが考えられる．

- サマリ更新手法 **1**：サーバが定期的にサマリを作成する
- サマリ更新手法 **2**：サーバはクライアントの要求通りにサマリを作成する
- サマリ更新手法 **3**：クライアントがサマリの作成を行う

本節では，ネットワーク運用を支援する観点から上記のモデルについて議論し，本ツールが採用した手法について述べる．

サマリ更新手法 1 : サーバが定期的にサマリを作成

サマリ更新手法 1 は，サーバがクライアントの有無に関わらず定期的にパケットを収集し，サーバ側の設定に従ってサマリを作成するモデルである．図 3.12 に，この手法の概要を示す．

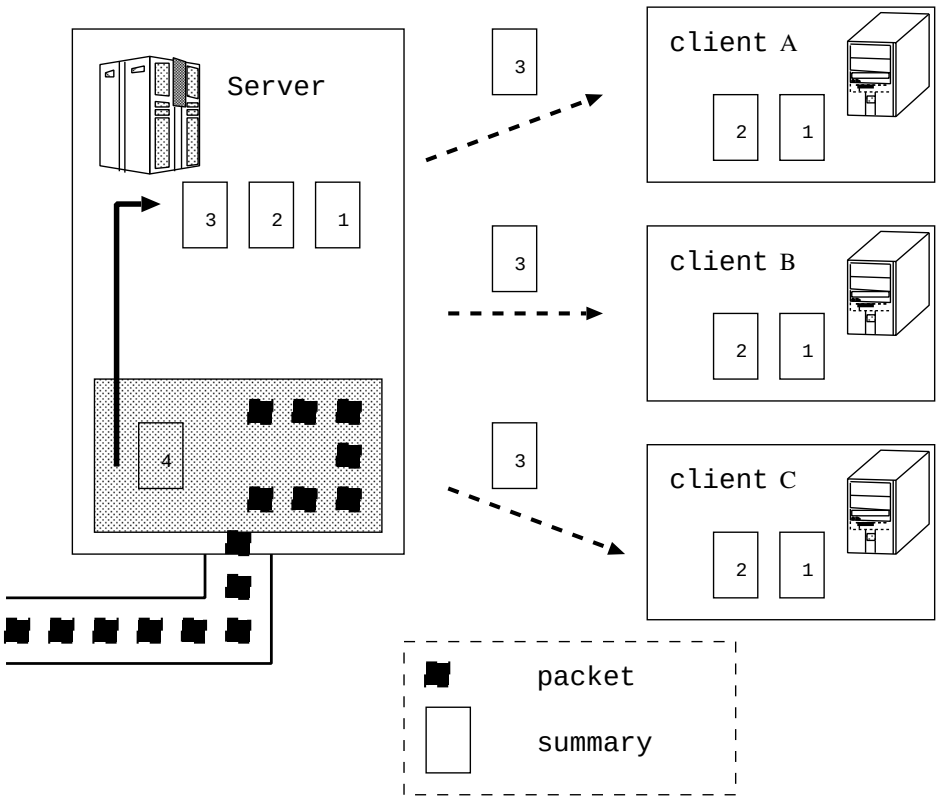

図 3.12: サーバ，クライアント間のサマリ更新 (手法 1)

サマリ更新手法 1 を用いた場合，クライアントがサーバに接続して来た時点で，サーバに保持していた過去のサマリを渡せる．この点は，ネットワークの不審な挙動を感じてからネットワークの状況を調べる場合にとって非常に有効である．

また，クライアントの数が増えた場合でも，サーバの中で保持するサマリの量は変化しないため，クライアントの増加に対して対応出来る．

しかし，サマリ更新手法 1 を用いた場合，サーバ側があらかじめ設定した形式でしかサマリを作成出来ないため，クライアント側の詳細な要求を満たせない．

サマリ更新手法 2 : サーバがクライアントの要求通りにサマリを作成

サマリ更新手法 2 は，クライアントの要求する設定に従ってサーバ側がパケットを収集しサマリを作成するモデルである．図 3.13 に，この手法の概要を示す．

サマリ更新手法 2 を用いた場合，クライアントがサーバに接続して来た時点で，サー

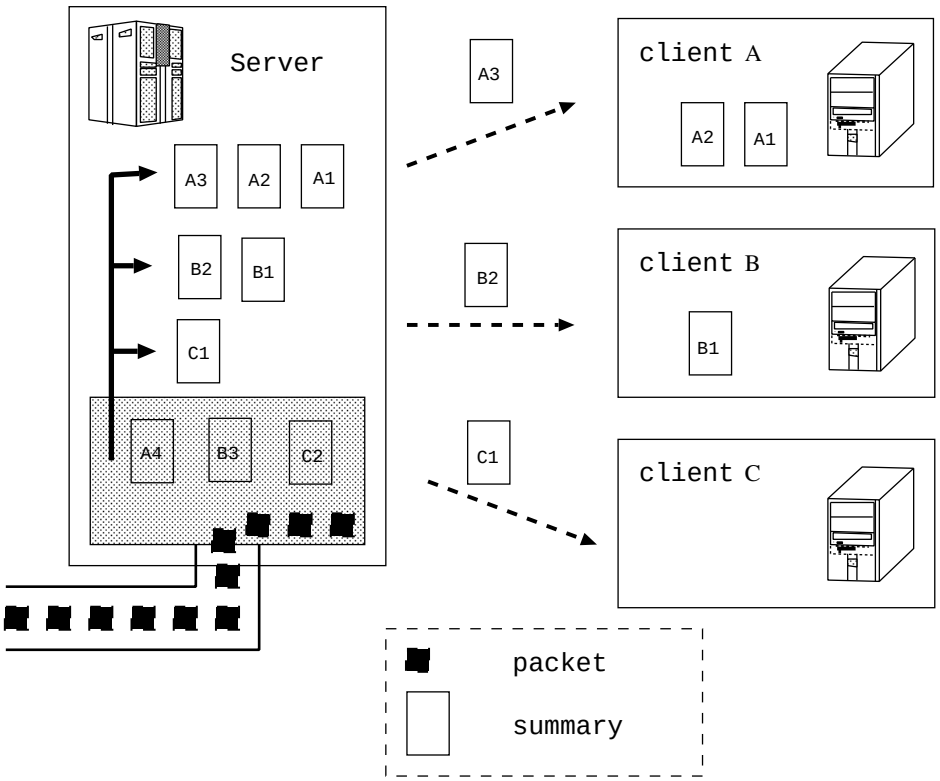

図 3.13: サーバ，クライアント間のサマリ更新 (手法 2)

バはそれぞれのクライアント毎の設定に応じてサマリを作成し送信する．これによって，クライアント側では独自の要求に適したサマリを取得出来る．

しかし，クライアントの数が増加した場合，サーバ側ではそれに応じて作成するサマリの量が増加する．サーバにかかる負荷が大きくならないように，クライアントの数を限定する必要性が生じる．

サマリ更新手法 3 : クライアントがサマリを作成

サマリ更新手法 3 は，サーバ側で収集したパケットのヘッダ部分，もしくはサマリ作成に必要な属性だけを抽出して作成した独自のパケットをクライアント側に送信するモデルである．図 3.14 に，この手法の概要を示す．

サマリ更新手法 3 を用いた場合，クライアントがサーバに接続して来た時点で，サーバはそれぞれのクライアントの IP アドレスだけを保持する．そして，それぞれのクライアントに対して全てのパケット数だけ独自パケットを送信する．これによって，クライアント側では独自の要求に適したサマリを作成出来る．

しかし，クライアントの数とサーバが収集するパケットの数に応じてサーバ，クライアント間で渡されるデータ量が増加してしまうため，クライアントの増加に対応することが難しい．また，サーバ，クライアント間で渡されるデータが多いため，対象

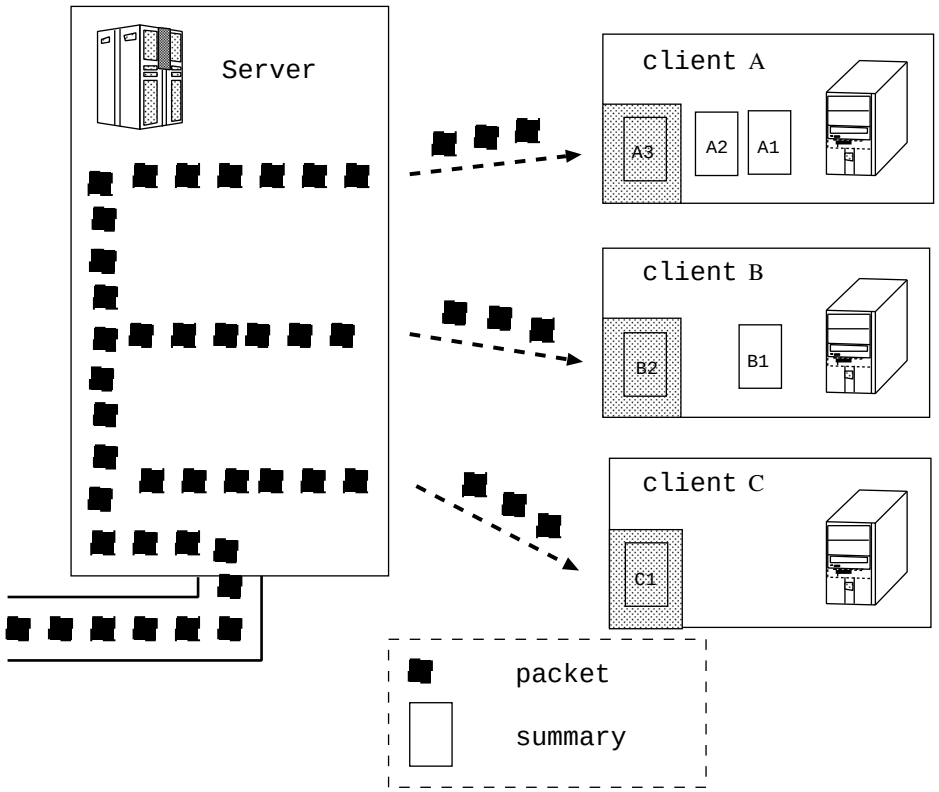

図 3.14: サーバ，クライアント間のサマリ更新 (手法 3)

となるネットワーク自身の状況を変化させてしまう可能性が高い．

比較検討

表 3.4 に，これらの手法をまとめる．

表 3.4: サマリ作成のタイミング相互比較

	評価項目 1	評価項目 2	評価項目 3	評価項目 4
サマリ更新手法 1	取得可能	不可能	一定	少
サマリ更新手法 2	取得不可能	可能	クライアント数と比例	少
サマリ更新手法 3	取得不可能	可能	一定	多

- 評価項目 1 : クライアント起動前のネットワーク情報
- 評価項目 2 : クライアント毎の設定
- 評価項目 3 : サーバが保持するサマリ量
- 評価項目 4 : サーバ，クライアント間のデータ量

本ツールはネットワーク運用を支援することを目標としている．この目標を前提とした場合，以下のことが言える．

- クライアント起動前のネットワーク状況を知ることが出来るのはサマリ更新手法 1 だけである．つまり，ネットワークの挙動がおかしいと感じた時点で過去のネットワーク状況を知ることが出来るのはサマリ更新手法 1 だけである．
- ネットワーク管理者が複数であることを考えると，クライアント数の増加に耐え得るシステムである必要がある．その点で，サマリ更新手法手法 2，サマリ更新手法 3 はクライアント数の増加に対応出来ない．
- 対象となるネットワーク状況を正確に捉えるためには，本ツールのサーバ，クライアント間で渡されるデータ量は少ない方がよい．その点でサマリ更新手法 3 は本ツールの手法として不適切である．

以上の論拠により，本ツールではサマリ更新手法 1 を採用する．

3.3 異なる視点でネットワークを視覚化

本節では，収集したデータを視覚化するにあたって採用するデータ構造について議論する．

3.3.1 収集したデータの階層化

前節において，ツーリの中からデータ量の多いノードを抽出する手法について議論した．この手法を用いて以下に示す 7 つの属性を視覚化するためには，属性を前節で述べた「ノードが保持する値」と「階層化要素」に分類する必要がある．

- パケット数
- データ量
- 送信元 IP アドレス
- 宛先 IP アドレス
- プロトコル
- 送信ポート
- 宛先ポート

本ツールでは図 3.15 のように，既に深いツリー構造になっている IP アドレスに関しては単一のツリーとして扱う．また，図 3.16 のようにツリー構造にはなっていないプロトコルとポート番号を階層化させる．これによって，「プロトコル」と「ポート番号」という 2 つの属性を関連付けることが出来る．そのため，ネットワークを流れるトラフィックの傾向をより詳細に把握することが出来る．

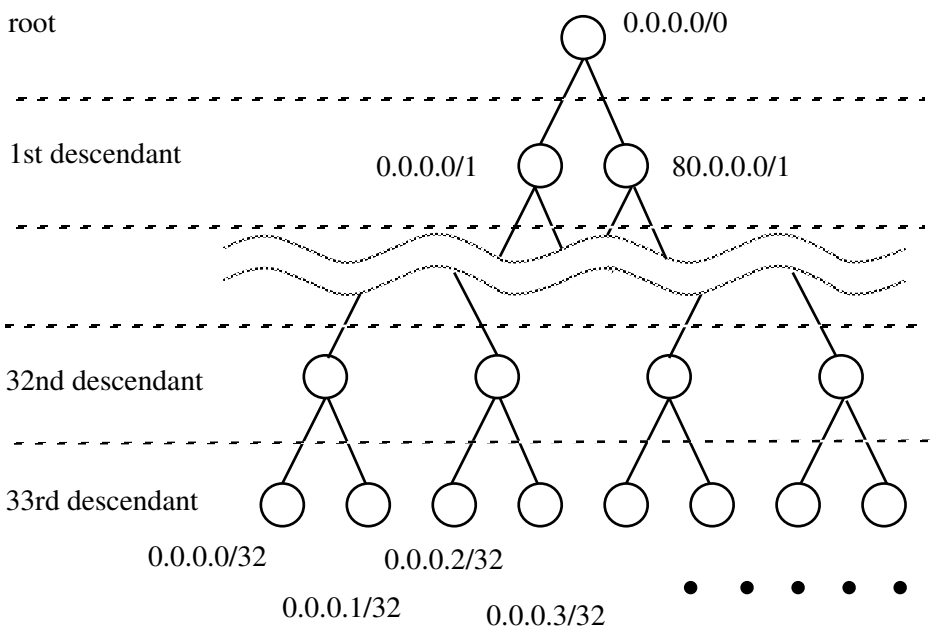

図 3.15: IP アドレスツリーの階層化

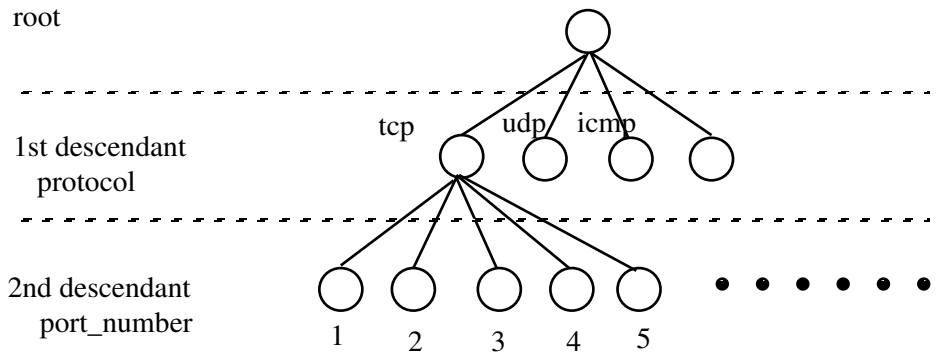

図 3.16: プロトコル，ポートの階層化

図 3.15 に示した IP アドレスツリーを送信元 IP アドレス，宛先 IP アドレス両方に対して構築する．さらに，この 2 つのツリーに対して「ノードの保持する値」をトラフィック量，パケット数それぞれに関して構築する．

図 3.16 に示したツリーを送信ポート，宛先ポートそれぞれに対して構築する．さらに，この 2 つのツリーに対して「ノードの保持する値」をトラフィック量，パケット数それぞれに関して構築する．

表 3.5 に本ツールが構築する 8 通りのツリーをまとめる．

表 3.5: 本ツールが作成する 8 つのツリー

	ノードが保持する値	階層化要素
ツリー 1	トラフィック量	送信元 IP アドレス
ツリー 2	トラフィック量	宛先 IP アドレス
ツリー 3	トラフィック量	プロトコル，送信ポート
ツリー 4	トラフィック量	プロトコル，宛先ポート
ツリー 5	パケット数	送信元 IP アドレス
ツリー 6	パケット数	宛先 IP アドレス
ツリー 7	パケット数	プロトコル，送信ポート
ツリー 8	パケット数	プロトコル，宛先ポート

表 3.5 に示したように，パケットから収集出来る並列な情報を階層化させることで，異なる視点からネットワークの状況を捉えることを可能にする．

3.3.2 異なる時間粒度によるサマリの作成

これまで述べてきたように，本ツールでは収集したデータを 8 種類のツリーに蓄積し，ある一定時間毎にツリー中のデータ量の多いノードを抽出しサマリを作成する．

本ツールでは，このサマリを更に時間的に集約することで異なる時間粒度でのネットワーク状況を捉えることを可能にする．

図 3.17 を用いて，サマリが更新される時間を「5 分」と仮定した場合の時間粒度の議論を行う．

図 3.17 中のサマリ A は最近 1 時間のネットワーク状況を表すサマリである．このサマリ A は，5 分間のサマリである，サマリ 1 からサマリ 12 まで 12 個のサマリから作成される．この際用いられる手法は 5 分間のサマリを作成した手法と同様である．

同様の手法によって，1 時間のサマリを 24 個集約し 1 日分のサマリを作成する．1 ヵ月間のサマリ，1 年間のサマリも同様に作成する．

このように時間粒度の異なるサマリを作成することで，表 2.2 に示したように，短期的なネットワークの状況から長期的なネットワークの状況まで把握出来る．

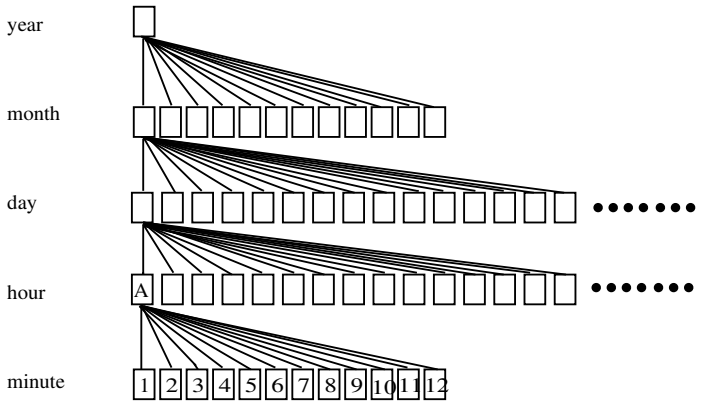

図 3.17: 異なる時間粒度のサマリ作成

3.4 設計のまとめ

この章では，本ツールの目的を実現するための手法について議論を行った．その議論の結果，本ツールで採用された手法をまとめる．

- トラフィックの多いフローを動的に検出

  libpcap を使用してネットワークの情報を収集する．

  ツリー形式に収集された情報から「印象値」という独自の指標を用いてトラフィック量の多いノードを抽出する．ここで抽出されたノードの情報のみを記述して保存したものを「サマリ」と定義した．

- ネットワーク情報を異なるタイムスケールで視覚化

  サーバ，クライアント間のサマリ更新の手法として，サーバ側があらかじめ設定した様式にしたがってサマリを作成し，クライアントへ配信するモデルを採用した．

- ネットワーク情報を異なる視点で視覚化

  ネットワークから収集するパケットの「トラフィック量」，「パケット数」「送信元 IP アドレス」，「宛先 IP アドレス」，「プロトコル」，「送信ポート」，「宛先ポート」の 7 つの属性を 8 つのツリー構造の記憶領域に蓄積しサマリを作成する．

  更に，サマリを時間的に集約することによって異なる時間粒度でネットワークの状況を把握することを可能とした．

第4章　本ツールの実装

4.1　実装概説

本章では，3 章で述べた設計の実装例として，プロトタイプ実装の詳細を示す．このプロトタイプは，「トラフィック量」と「送信元 IP アドレス」，「宛先 IP アドレス」を要素とする実装である．

4.2　データ抽出部

図 4.1 に libpcap を使用したパケットの抽出部を示す．

```
    /* setting and compiling packet filter at &fcode*/
    if (pcap_compile(pd, &fcode,cmd, 1, netmask) < 0) {
fprintf(stderr, "can't compile fileter\n");
exit(1);
    }
    if (pcap_setfilter(pd, &fcode) < 0) {
fprintf(stderr, "can't set filter\n");
exit(1);
    }
    /* set call back function for output */
    callback = draw_data;
    while(pcap_flag == 1){
      if (pcap_loop(pd, 1 , callback, NULL) < 0) {
(void)fprintf(stderr, "pcap_loop: error occurred\n");
exit(-1);
      }
    }
```

図 4.1: libpcap を使ったデータ抽出

図 4.1 の"cmd" に記述されたフィルタに従ってパケットを抽出することが出来る．本プロトタイプでは，「ip」と記述し IP パケットのみの抽出を行った．pcap_loop() で

は ,”call_back” 変数に代入した関数を繰り返す .本プロトタイプでは ,この変数に”draw_data” という関数を代入し抽出された全てのパケットをツリー構造の記憶領域に保存した .

4.3 ツリー構造へのデータの保存

図 4.2 に , libpcap を用いて抽出したデータを示す . 図 4.3 に , 抽出したデータをツリー構造の記憶領域に保存する構造体を示す .

```
struct tree {

  unsigned int address; /* IP address */
  int  slash;           /* length of host part */
  int length;           /* data length */
  mpz_t impact;         /* original indicator for drawing nodes */
  struct tree *left;    /* child node */
  struct tree *right;   /* child node */

};
```

図 4.2: データ保持のための構造体

図 4.2 の”mpz_t” という変数の型は GMP (GNU Multipule Precision)[20] ライブラリで宣言されている型である . GMP を用いることで , ”unsigned int” では表現出来ない 2^{32} 以上の数を表現出来る .

本プロトタイプでは , 独自の指標である印象値の算出の際に 2^{32} 以上の値が出る可能性が高いため GMP を用いた .

図 4.3 では , 抽出したデータをツリー型の記憶領域に保存している . この際 , 抽出したパケットの IP アドレスにマスクをかけ 32bit のアドレスの先頭から順に見ていく .

IP アドレスは「0」と「1」の 32bit のバイト列であるので , 2 分木によって表現出来る .

もし , 既にノードがあればそのノードにデータを加算し , ノードが無ければ新しくノードを作成している . この作業を一つのパケットにつき 32 回繰り返すことによって抽出したパケットから , ツリー構造の記憶領域にデータを全て保存出来る .

4.4 ツリー構造からのトラフィック量の多いフローの抽出

図 4.4 にツリー構造の記憶領域に保存したデータの中からトラフィック量の多いフローを抽出する手法の実装を示す .

```
void addtree(struct tree *root_pt,unsigned int ip_addr,int len){
  struct tree *ptn;
  double impact_level = IMPACT_LEVEL;
  int i;
  unsigned int next_bit;
  mpz_t big_impact,buf;
  ptn = root_pt;
   /* go to target point */
  for(i = 0 ; i <= 32 ; i++ ){
[抽出したデータの値を加算]
    /* search next */
    next_bit = get_bit(ip_addr,i+1); /* ホスト部の値を 1 増加 */
    if(next_bit > 0 ){ /* thus next bit == 1 */
      if(ptn->left == NULL && i != 32){
  /* add new node */
ptn->left = (struct tree *)malloc(sizeof(struct tree));
ptn = ptn->left;
 [新しく作ったノードを初期化] }
      else{ptn = ptn->left;}
    }
     else if(next_bit == 0 ){ /* thus next bit == 0 */
      if(ptn->right == NULL && i != 32){
/* add new node */
ptn->right = (struct tree *)malloc(sizeof(struct tree));
ptn = ptn->right;
 [新しく作ったノードを初期化] }
      else{ptn = ptn->right;}
    }
    else{printf("error next bit == negative\n");
         exit(-1);}
  }
}
```

図 4.3: ツリー構造へのデータの保存

図 4.4 では，形成されたノードを全て辿って印象値によって順位づけされたリスト構造にデータを挿入する．本実装では，以下のアルゴリズムでノードを検索する．

1. 子ノードが無ければデータを抽出し，そのノードを消しスコープを親ノードに移す．この際，親ノードはスタックで管理する．

```
void tree_to_list(struct tree *pts){
  struct tree *root,*stack[128];
  struct list *current_pt;
  int i = 0;
  root = pts;
   current_pt = (struct list *)malloc(sizeof(struct list));
    mpz_init(current_pt->impact);
  while( !(root->left == NULL && root->right == NULL)){
    /* 子ノードに行けるだけ行く */
    while(!( pts->left == NULL && pts->right == NULL)){
      if(pts->left != NULL){
stack[i] = pts;
i++;
pts = pts->left;}
      else {   /* thus pts->right != NULL */
stack[i] = pts;
i++;
pts = pts->right;}
    }
[ノードからデータを抽出し，印象値によって順位づけ
        されたリスト構造へ代入する]
    if(root == head_src){sort_list(current_pt,1);}
    else if(root == head_dst){sort_list(current_pt,2);}
    else{printf("make list error\n");
         break;}
    i--;
    pts = stack[i];
    if(pts->left != NULL){
      free(pts->left);
      pts->left = NULL;}
    else if(pts->right != NULL){
      free(pts->right);
      pts->right = NULL;}
  }
 }
```

図 4.4: トラフィック量の多いフローの抽出

2. 左の子ノードがあれば，スタックに現在のノードを積み，スコープを左の子ノードに移す．

3. 右の子ノードがあれば，スタックに現在のノードを積み，スコープを右の子ノードに移す．

このアルゴリズムを使用することによって，完全な 2 分木構造 [21] (親ノードが必ず子ノードを 2 つ持つ木構造) ではない記憶領域の全てのノードを辿ることが出来る．

以下，図 4.5 に具体例として本実装を使った場合のノードの検索順を示す．

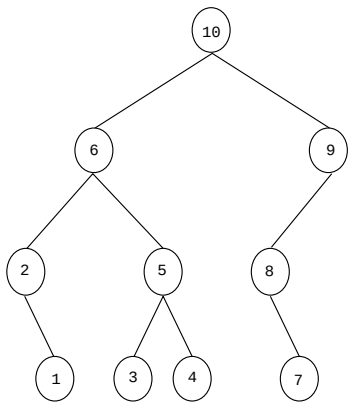

図 4.5: 本実装において抽出される順番

図 4.5 に示したツリー構造を用いて説明すると，スコープは最初ツリー構造の root である「10」の位置にある．ノード 10 は左に子ノードを持っているので，スタックに「10」を積み，スコープを「6」に移す．

ノード 6 は左に子ノードがあるので，スタックに「6」を積み，スコープを「2」に移す．

ノード 2 は左に子ノードを持っておらず，右に子ノードを持っているので，スタックに「2」を積んでスコープを「1」に移す．

ノード 1 は左にも右にも子ノードを持っていないので，値を抽出しスコープをスタックの一番上に積んである「2」に移す．

本実装では，この様な順序でツリー構造を辿ってゆくことで 2 分木ではないツリー構造中のデータを全て検索することが出来る．

4.5 サマリデータの格納，更新

図 4.6 に作成したサマリデータの格納部を示す．

本実装では，サーバが起動した時点からのサマリを蓄積することにより，クライアントがサーバに接続する以前のネットワーク状態を知ることが出来る．そのためには，サマリが作成された時間をサーバが保持している必要がある．

```
/* write current_sammary to file with txt format  */
void write_sammary(){
  int i;
  FILE *file;
  char name[64];
  char *filedir;
  char filename[128];
  time_t t;
  filedir = DATA_DIR;
  t = time(NULL);
  sprintf(name,"%d",t);
  strcpy(filename,filedir);
  strcat(filename,name);
  file = fopen(filename,"wb");
  if (file < 0) {
    printf("error file_create\n");
    exit(-1);
  }
  for(i = 0; i < MAX_SAM_NUM; i++){
    fprintf(file,"%0x/%d\t%d\t%0x/%d\t%d\n",src_pt[i]->address, \
   src_pt[i]->slash,src_pt[i]->length,dst_pt[i]->address,  \
   dst_pt[i]->slash,dst_pt[i]->length);
   }
  fclose(file);
  }
 }
}
```

図 4.6: サマリデータの格納

そこで本実装では time 関数 [22] を用いてサマリのファイル名が一意になるように工夫している．更に，クライアント側に保持している最新サマリの時間をサーバに通知することによって，ネットワークが不安定な状態でも，常に差分のサマリだけを受け取れる．

4.6 実装環境

本章では実装を行ったハードウェア環境，ソフトウェア環境について述べる．

4.6.1 ハードウェア環境

表 4.1 に本実装を行ったハードウェア環境を示す．

表 4.1: ハードウェア環境

機種	ThinkPad600E
プロセッサ	PentiumII366MHz
メモリ	256MB
ハードディスク	6.1GB
NIC	10baseT

4.6.2 ソフトウェア環境

表 4.2 に本実装を行ったソフトウェア環境を示す．

表 4.2: ソフトウェア環境

OS	FreeBSD3.5-RELEASE[23]
使用言語	C
コンパイラ	gcc2.7.2.3
libpcap	libpcap0.5
gmp	gmp3.1.1

第5章　評価

本章では，実装を行った本ツールのプロトタイプについて評価する．本ツールを用いて実際のネットワークを測定し，本ツールの出力によってネットワーク状態の変化を動的に把握出来るかどうかを評価する．

5.1　測定環境

測定は以下に示す環境で行った．

5.1.1　測定ネットワーク環境

測定は研究室のネットワークで行った．このネットワークには定常的に実データの転送が行われている．

図 5.1 に，測定を行ったネットワーク環境を示す．

図 5.1 中の「accelar1250」でポートミラーリング行い，「rgate」の「203.178.143.1/24」のネットワークインターフェースに転送中のパケットと同一のパケットを「measurement-machine(測定機)」が収集出来るように設定した．

この設定によって「203.178.143.0/24」の IP アドレス空間から送受信されたパケットを全て収集することを可能にした．

5.1.2　測定時間帯

- 測定開始時刻 ： 2001 年 1 月 26 日 金曜日 20 時 30 分
- 測定終了時刻 ： 2001 年 1 月 30 日 火曜日 12 時 30 分

5.1.3　測定機環境

表 5.1 に測定機のハードウェア環境を示す．

表 5.2 に測定機のハードウェア環境を示す．

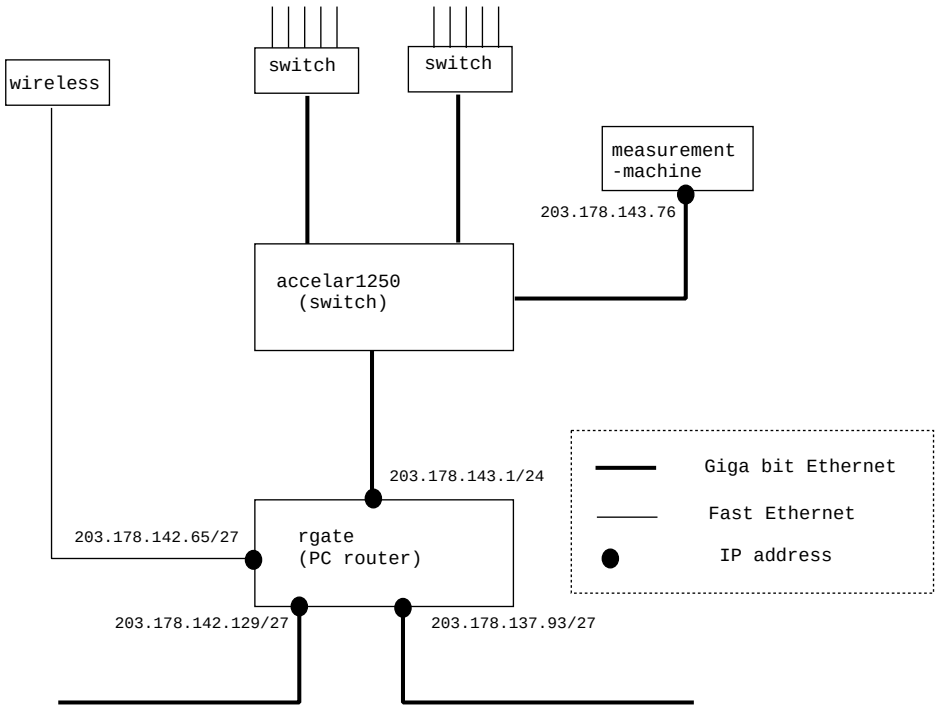

図 5.1: 測定トポロジ概要

表 5.1: ハードウェア環境

機種	PC 互換機
プロセッサ	PentiumIII450MHz
メモリ	256MB
ハードディスク	8.7GB
NIC	1000baseSX

5.2 測定結果

測定結果の表示には gnuplot[24] という関数描画アプリケーションを使用した．そのため，本ツールで出力したサマリを gnuplot が読むことの出来る形式に変換するプログ

表 5.2: 測定機ソフトウェア環境

OS	FreeBSD4.1-RELEASE
使用言語	C
コンパイラ	gcc2.7.2.3
libpcap	libpcap0.5
gmp	gmp3.1.1

ラムを作成した．

gnupolt の描画する折れ線グラフは，6 種類の色と 6 種類の線種の組合せで最大 36 本の折れ線グラフを出力することが出来る．しかし，本論文が黒白で出版されることを考慮し，以下に示す測定結果の表示には 6 つの線種の折れ線グラフを出力した．

5.2.1 定常的なネットワーク状態

図 5.2 に定常的なネットワーク状態の出力図を示す．図中の y 座標は kbytes 単位であり，5 分間のトラフィック量を示している．例えば，2 時 25 分の総トラフィック量は約 1Gbytes である．この値を計算すると約 26.7Mbps となる．

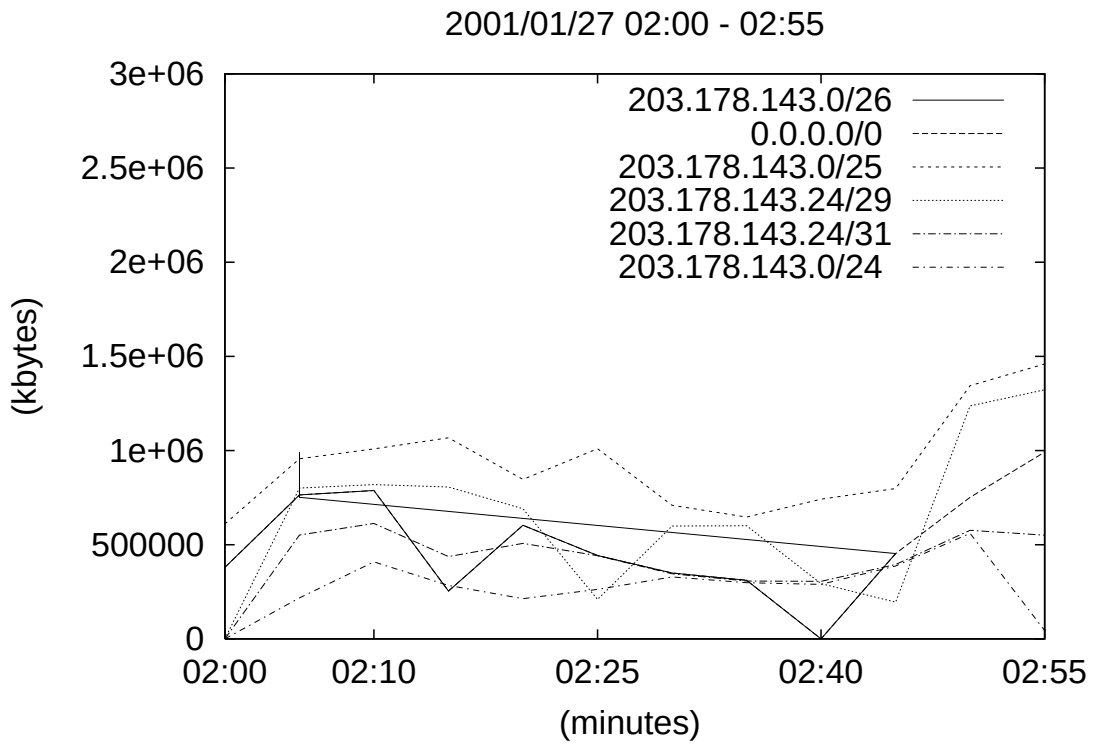

図 5.2: 定常的なネットワーク状態

図 5.2 は，1 月 27 日 2 時から 3 時まで 1 時間のパケットを収集し，送信 IP アドレスに関して出力したものである．

この図によって複数のフローを集約して視覚化することを確認出来た．

5.2.2 特徴的なネットワーク状態

図 5.3 に，特徴的なネットワーク状態の出力を示す．

この測定では，故意に特徴的なネットワーク状態を作り出すために，5 時 30 分から 5

時 40 分の間に，測定機のパケット収集ポイントを通過させるように，DVTS[25](Digital Video Transport System) を用いて約 31Mbps のデータを送信した．

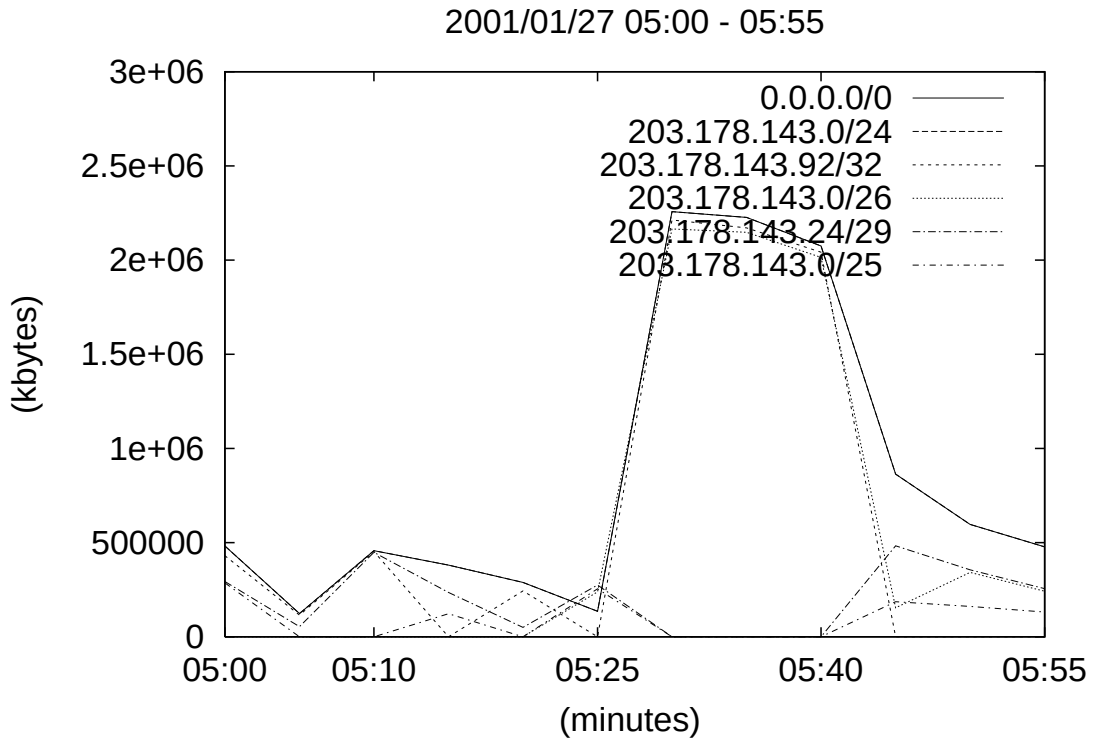

図 5.3: 特徴的なネットワーク状態

図 5.3 には，図 5.2 では現れなかった特定のホスト (この場合は 203.178.143.92/32) の IP アドレスが出現している．

また，全体のトラッフィク量の急激な増加が，特定のホストの送信するトラフィックによって引き起こされていることをから容易に判断出来た．

5.3 評価

本モニタリングにより，以下に示すことが確認された．

- フローを集約して視覚化出来る

  図 5.2 に見られるように，大量のデータを送出するホストが存在しない場合は大きな IP アドレス空間に複数のフローを集約することが出来た．また，その集約されたフローを視覚化出来た．

- トラフィック全体を構成しているフローを検出し視覚化出来る

  図 5.3 に見られるように，全体のトラッフィク量が急激に増加した場合，その原因となっているフローを視覚化出来た．

- 動的にトラフィック量の多い特定のホストを検出することが出来る

  図 5.3 に見られるように，あるホストが送出するデータ量が多い場合，予め設定をすることなく， そのホストを特定出来た．

第6章　結論

本章では，本研究の結論を述べ，その後に今後の課題と本研究の将来的展望を述べる．

6.1　結論

本研究では，ネットワーク運用を支援するツールの設計と実装を行った．

ネットワーク運用を行うことは，ネットワークの現状に適した機器の設定を行うことである．そのためには，管理者がネットワークの現状を把握しなければならない．しかし，現状のネットワークモニタリングツールでは，詳細な情報を取得し且つ長期間に渡ってトラフィックの傾向を把握出来ない．

本研究では以下の 3 項目を満たすネットワークモニタリングツールを設計し，そのプロトタイプとして「トラフィック量」「送信元 IP アドレス」「宛先 IP アドレス」を要素とする実装を行った．

1. トラフィック量の多いフローの動的検知
2. 異なるタイムスケールでのネットワークの視覚化
3. 異なる視点でのネットワークの視覚化

実装において，収集したパケットから「データ長」「送信元 IP アドレス」「宛先 IP アドレス」を抽出し，ツリー構造の記憶領域に保存した．このツリー構造中のノード全てに「印象値」という独自の指標を付加し，その指標に従ってツリー中から特徴的なデータを抽出した．

更に，本研究で実装したプロトタイプを用いて実ネットワークにおいてデータを収集し評価を行った．

その結果，本研究の実装は既存のネットワークモニタリングツールでは検知出来なかったデータ送受信量の多いホストの特定を動的に検知出来た．更に，特にトラフィック量の多いフローが無い場合は，幾つかのフローを集約することを確認した．これにより，本研究の実装によってネットワーク運用を支援出来ることが検証された．

6.2　今後の課題

本研究における今後の課題には，以下のようなものが挙げられる．

IPv6，パケット数，プロトコル，送信ポート，宛先ポートへの対応

本研究では，「IPv4」「トラフィック量」「送信元IPアドレス」「宛先IPアドレス」を要素とするプロトタイプ実装を行った．しかし，ネットワークの現状をより詳しく把握するためには，「IPv6」「パケット数」「プロトコル」「送信ポート」「宛先ポート」も収集する必要がある．

ツリー集約アルゴリズムの改変

本研究では，ツリー構造の記憶領域からデータ量の多いノードを抽出する手法として3つの手法を述べ，「印象値」という独自の指標を用いてノードを抽出する手法を採用した．

しかし，この手法は不要な中間ノードが作成される欠点を持っている．そこで，「パトリシア木」というアルゴリズムを用いて本実装を改良する予定である．このアルゴリズムはラディックスツリーの形成 [26] にも用いられているものである．

サーバ側の設定項目を増やす

本研究の実装では，サーバ側の設定項目として「情報を収集するインターフェース」と「サマリを作成する時間間隔」の2つがある．

しかし，印象値を算出するために用いられる「印象度」や，サーバが作成する「サマリの項目数」は固定値として実装の中に組み込んでいる．これらの設定項目をサーバ側で自由に設定したり，過去の履歴から最適値を算出出来れば，よりネットワーク管理者の要求にあったネットワーク状態を見られるようになる．

多地点での収集したデータの集約への対応

本実装を用いた場合，ネットワークの一地点でデータ収集出来る．

しかし，ネットワークはさまざまな回線や通信機器によって構成されているためネットワークの一地点でデータを収集するだけではネットワークの全体像を掴むことは難しい．

そこで，多地点で収集したデータを集約する機能を付加する必要がある．その際，多地点で収集したデータを集約することの意義や手法について議論する必要がある．

トラフィック量の急激な増減の検知と警告

本研究ではトラフィック量の多いフローに注目した．しかし，実際のネットワーク運用時に使われることを考えた場合，本ツールでは，定常的にトラフィック量が多かったフローが突然無くなる事態を検知することが出来ない．更にトラフィック量が多いフローを検知した場合も，そのフローが定常的にトラフィック量が多いのかどうかを知ることが出来ない．

この問題を解決するためには，サマリ情報を時間的に比較して管理者に見せる機構の充実を図る必要がある．また，これらの急激なトラフィック量の増減を見せるだけでなく，システム自体が判断する機構を付加することで管理者への警告を可能にする．

6.3 本ツールの将来的展望

ストリーミングアプリケーションに代表される，低遅延，広帯域をネットワークに要求するアプリケーションが増加している．限り有るネットワーク資源の中でこれらのアプリケーションの要求する品質を保証するためには，最新のネットワーク技術を適切に用いたネットワーク運用を行う必要がある．

本ツールは，以下に示す 3 点でネットワーク運用を支援出来る可能性を持っている．

- QoS(Quality of Service)[27] ルーティング
- AS[31] 間 SLA(Service Level Agreement)[32] の締結
- ネットワーク設計

6.3.1 QoS ルーティングの支援

広帯域，低遅延のネットワークを実現する技術として，ATM[27] や RSVP[28] などの Intserv(Integrated service)[29] 型のシステムを用いてフローごとに帯域を確保する方法が用いられている．更に，Diffserv(Differentiated services)[30] 型システムを用いてフローを幾つかのサービスクラスに分類して帯域制御を行う方法が注目されている．しかし，既存のトラフィックモニタリングツールを使用した場合，動的なフローの検出が難しい為，ネットワークの現状に即した帯域確保，帯域制限を行うのは非常に困難である．

本ツールでは，ネットワーク上を流れるトラフィックのフローを動的に検知するため，ネットワーク管理者がネットワークの状況を詳細に知っていない場合でも各フローに応じた帯域確保を設定出来る．また，Diffserv 型システムを用いた場合でも適切なサービスクラスの分類を行うための大きな示唆となる．

6.3.2 AS 間 SLA の締結を支援

インターネット上における情報配信の品質は，配信経路上の各 AS (Autonomous System) の性能に大きく依存する．ここで，AS とはインターネットにおけるルーティングの管理単位である．しかし，現状では AS 間の SLA(Service Level Agreement) を定義するためのネットワーク情報が不足しているため，情報配信の品質まで考慮した AS 間の通信は不可能である．

本ツールを使用した場合，AS の端でデータを収集，集約することで管理している AS をトランジットの為だけに通過するデータを検出出来る．このデータは，各 AS 間での SLA を定義する際の大きな指針となり AS をまたがった情報配信が円滑に行われる．

6.3.3 ネットワーク設計の支援

管理しているネットワーク内においてボトルネックとなり得る経路を検知した場合，ネットワーク管理者はその回線の増強を行う．しかし，状況によっては回線の増強を行わなくてもネットワーク機器を設置，置換するだけで問題が解決する場合も多い．更に，ネットワークの状態を熟知しているネットワーク管理者ならば，既存のネットワーク機器の配線や設定を変更するだけで問題が解決される場合もある．

本ツールをボトルネックをなっている回線に設置した場合，回線を通過しているトラフィックのポート番号，宛先，送信元を時間的経緯と共に大まかに知ることが出来る．そのため，回線を圧迫している原因を絞りこみやすくなり，原因に即した解決策を取ることが出来る．

謝辞

本研究を進めるにあたり，御指導を頂きました，慶應義塾大学環境情報学部教授の徳田英幸博士と村井純博士に感謝致します．また，絶えず御助言と御指導を頂きました，同学部の中村修博士，楠本博之博士，SonyCSL の長健二朗博士に深い感謝の意を表します．

杉浦一徳氏，小川晃通氏をはじめとする慶應義塾大学政策メディア研究科博士課程の方々，本論文中の図を数多く描いてくれた小畠元氏，並びに同大学環境情報学部の徳田・村井・中村・楠本研究室の諸氏には，本論文執筆の上で様々な議論をして頂き，また耐えざる励ましとご援助を頂きました．ここに，深い感謝の意を表します．

また，重近範行氏，土本康生氏，今泉英明氏，小原泰弘氏をはじめとする WIDE インターネットの管理者の方々には，ネットワークの運用に関して絶えず御指導頂きました．ここに感謝の意を表します．

最後に，４年間の長きに渡って学費を払い，辛抱強く私を信じて見守って下さった母に感謝いたします．

以上を持って謝辞と致します．

参考文献

[1] J.Case,M.Fedor,M.Schoffstall,J.Davin "A Simple Network Management Protocol (SNMP)", RFC1157, May 1990

[2] S.McCanne,V.Jacobson, "The BSD Packet Filter: A New Architecture for User-Level Packet Capture", '93 USENIX Conference, January 1993

[3] Oetiker,Tobias, "MRTG - The Multi Router Traffic Grapher", USENIX System Administration Conference, December 1998

[4] R.Needham "Denail of Service: An Example", Communications of the CACM volume 37, November 1994

[5] "NCSA Mosaic Documentation", http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html, January 1997

[6] "WIDE 報告書", 1999

[7] G.Kessler,S.Shepard "A Primer On Internet and TCP/IP Tools" chapter 2.2, RFC1739, December 1994

[8] G.Kessler,S.Shepard "A Primer On Internet and TCP/IP Tools" chapter 2.3, RFC1739, December 1994

[9] J.Postel "Internet Control Message Protocol", RFC792 September 1981

[10] K.McCloghrie "Management Information Base for Network Management of TCP/IP-based internets : MIB-II", RFC1213 March 1991

[11] T.Oetiker, "Round Robin Database Tool", http://www.caida.org/Tools/RRDtool/

[12] J.Allen, "Cricket Monitoring System", http://www.munitions.com/ jra/cricket/

[13] "netflow documentation", http://www.cisco.com/univercd/td/doc/product/rtrmgmt/nfc/nfc_3_0/nfc_ug/nfcover.htm

[14] "Cooperative Association for Internet Data Analysis", http://www.caida.org

[15] "Cisco Systems", http://www.cisco.com

[16] "Cisco IOS Software Configuration", http://www.cisco.com/univercd/cc/td/doc/product/software/index.htm

[17] "WIDE プロジェクト", http://www.wide.ad.jp

[18] S.McCanne,V.Jacobson, "The BSD Packet Filter: A New Architecture for User-Level Packet Capture", '93 USENIX Conference, January 1993

[19] M.Allman,V.Paxson,W.Stevens "TCP Congestion Control", RFC2581, April 1999

[20] "GNU Multiple Precision", http://www.swox.com/gmp

[21] 近藤嘉雪, 「C プログラマのためのアルゴリズムとデータ構造」, Soft Bank Publishing, 1998 年 3 月

[22] "API C Language Real-Time Extensions", 9945-2:1993 ISO/IEC June 6, 1993

[23] "FreeBSD HomePage", http://www.freebsd.org/

[24] "GNUplot", http://www.ucc.ie/gnuplot

[25] Akimichi Ogawa, "Digital Video Transport System", http://www.sfc.wide.ad.jp/DVTS

[26] J.Bentley,R.Sedgewick "Fast Algorithms for Sorting and Searching Strings", ACM-SIAM Symposium, January 1997

[27] R.Cole,D.Shur,C.Villamizar "IP over ATM : A Framework Document", RFC1932, April 1996

[28] R.Braden L.Zhang S.Berson S.Herzog S.Jamin, "ResourceReSerVation Protocol (RSVP)", RFC2205, September 1997

[29] R.Barden D.Clark S.Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC1633, June 1994

[30] S.Blake D.Black M.Carlson E.Davies Z.Wang W.Weiss, "An Architecture for Differentiated Services", RFC2475, December 1998

[31] J.Hawkinson,T.Bates, "Guidelines for creation selection and registration of an Autonomous System(AS)", RFC1930, March 1996

[32] K.White, "Definitions of Managed Objects for Service Level Agreements Performance Monitoring", RFC2758 February 2000

[33] Almquist, "Type of Service in the Internet Protocol Suite", RFC1349, July 1992