第IV部

ネットワークトラフィック統計情報の収集と解析

第4部
ネットワークトラフィック統計情報の収集と解析

第1章　MAWI ワーキンググループ

MAWI (Measurement and Analysis on the WIDE Internet) ワーキンググループは、トラフィックデータの収集と解析、また、データの保存と利用に関する活動を行っている。MAWI WG では WIDE プロジェクトの特徴を活かした研究をするため、「広域」「多地点」「長期的」の三つの項目に重点を置いたトラフィックの計測・解析を行ってる。

1. 広域で行う
   インターネットの最大の特徴は、大規模な広域ネットワークにある。しかし、トラフィック情報には組織の機密保持やプライバシー保護の問題が伴うため、特に不特定多数のトラフィックを含む広域データを第三者が入手するのは困難である。ひとつの企業や組織内といった狭い範囲でデータを取ることは各組織で出来るが、広域バックボーンでのデータ収集はバックボーンを持っている WIDE だからできる。
2. 多地点で見る
   インターネットのもう一つの大きな特徴は、自律したネットワークが相互接続して経路制御を行い、エンド・エンドで通信制御を行う分散システムにある。したがって、ある地点でトラフィックを観測しても、ネットワーク全体の状態を捉えることは不可能である。観測者はあくまでその観測点から見たインターネット像が得られるだけで、別の観測点からはまったく別の世界が見えているかも知れない。MAWI WG では、多地点で観測したデータを照らし合わせることによって、より広い範囲のネットワークの状態を把握する手法や、それを俯瞰で可視化することによって直観的に分かりやすく観測する手法について研究を行っている。
3. 長期間行う
   ネットワークのトラフィックの挙動は、TCP の特定のアルゴリズムが関係するようなミクロなものから、過去 10 年間のトラフィック量の推移のようなマクロなものまで、幅広い時間スケールに渡っている。また、ミクロな挙動についても、インターネットのマクロなレベルの発展に伴って、次第に変わっていく。したがって、一日や一週間といった短期間の計測も重要だが、何年間という長いスパンでデータを取り続けることが非常に重要になる。しかし、長期的にデータを収集し、その蓄積を持つことは、ある日誰かが思い付いてできるものではない。 そこで、ワーキンググループとしてメンバーが協力して継続的なデータ収集を行っていくことが必要である。

MAWI WG では、バックボーンから得られたデータをホームページ上で公開している。公開されているデータは、プライバシーの保護を考慮し、必要に応じて解析に不要な部分を削除する、あるいは、アドレスをスクランブルするなどの加工を行っている。また、データ公開のためのルールや、そのために開発したツールも公開している。これらのデータは、研究目的であれば、WIDE 外の研究者でも自由に利用できる。

インターネット技術の多様化や、ネットワークの複雑化により、トラフィックの特性も多様化している。もはや、一年前のデーターに基づいて、一年後をプロジェクションするのは困難である。このような状況だからこそ、新しい計測や解析の手法が求められており、インターネットにおける計測や解析の重要性が見直されている。しかしながら、根気がいる割に成果が難しい分野であるため、組織的にバックアップして盛り上げて行かなければいけない。

今回の報告書では、まず、第 2 章で、現在開発を進めている集約型トラフィックプロファイラについて報告する。このツールは、WIDE バックボーンのトラフィックをリアルタイムかつ長期的にモニタリングする目的で作られている。また、急増する分散型 DoS アタックの早期検出の可能性も持つ。

第 3 章では、サーバシステムの負荷を計測する手法として、新しいカーネルモニタリングシステムを設計・実装し、WWW サーバーで行った評価を報告する。従来のカーネルモニタリング手法はサーバシステムに負荷を与える問題があったが、本提案は性能に対する影響が少ないのが特徴である。

第 4 章では、tcpdump およびダンプデータに関する話題として、1) プライバシー情報の保護、2) 全二重回線の観測を取り上げる。それぞれを実現するため開発したツールを紹介すると共に、その他 tcpdump に関係の深い一連のツールを取り上げる。

第 2 章　集約型トラフィックプロファイラの設計と実装

2.1 はじめに

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、ネットワーク状態に適した設定を行う必要がある。現存するトラフィックモニタリングツールは以下の 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せる
- ネットワーク上に流れるパケットの総量を見せる

tcpdump[160] 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。しかし、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 (The Multi Router Traffic Grapher)[119] 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詳細な情報は得られない。

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、かつ長時間に渡るトラフィックの傾向を視覚化することが出来ない。

2.2 目的

安定したネットワーク運用にはネットワークの現状把握が必要である。しかし、そのためには、トラフィック情報やフロー情報を統合的に収集、解析する必要があり、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。

そこで、本研究では 1) トラフィック中の特徴的なフロー傾向を残しつつ、2) 短期間から長期間に渡って利用可能なトラフィックモニタリングツールの設計、実装を目的とする。このツールを AGURI (Aggregation-based Traffic Profiler) とし、以下、本報告書では AGURI と呼ぶ。

本研究では、以下に示される 2 つの成果を期待できる。

1. 実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
   現在、ネットワークの再構成や機器設定は、ネットワーク管理者の予想や時間をかけて解析された情報に基づいて行われている。AGURI を使用することで、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。それにより、実トラフィックに基づいてネットワーク機器の設定を行うことができる。
2. ネットワークセキュリティ対策に応用できる
   現在、DoS (Denial of Service)[111] 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。AGURI を用いた場合、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から検出できる。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、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可能性をもっている。

2.3 設計

本章では、本研究の目的から必要とされる機能をまとめた上でトラフィックモニタリングツール AGURI の設計について述べる。第 2.2 で述べたように、AGURI の目的は 2 つあり、ここでは以下の

ように定義した。

- トラフィック中の特徴的なフロー傾向を抽出する
  収集したトラフィックを複数の属性から捉え、その属性毎に流量の多いデータフローを抽出する。これは、ネットワークを運用していく上で、流量の多い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。
- 短期間から長期間に渡って利用可能である
  短期間の利用には、詳細なネットワーク現状が動的に検知できる必要がある。長時間の利用には、過去のデータが容易に参照可能であることが必要である。更に、長期のデータ保存を考慮した場合、保存するデータサイズを小さく保つ必要がある。

AGURI に要求される機能には、(1) フローの動的検知 (2) 複数の属性からのフロー検知 (3) 過去データの参照 (4) 小さいデータサイズ等があり、これらの機能を満たした設計を行う必要がある。

2.3.1 データ収集部

トラフィックの多いフローには以下の 2 種類が存在する。

- 1 台のホストのトラフィックが多い
  単一のホストが多くのデータを送受信している。
- サブネットとしてトラフィックが多い
  個々のホストのデータ送受信量はさほど多くないが、個々のホストの集合であるサブネットのデータ送受信量が多い。

つまり、トラフィックの多いフローを抽出するには IP アドレスだけではなく、その IP アドレスが含まれる IP アドレス空間も考慮する必要がある。そこで AGURI では IP アドレスがツリー構造で形成される IP 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。個々としてはデータ量の少ないフローであっても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。

図 2.1 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。この場合、203.178.143.54 を送信元とする 80 バイトの

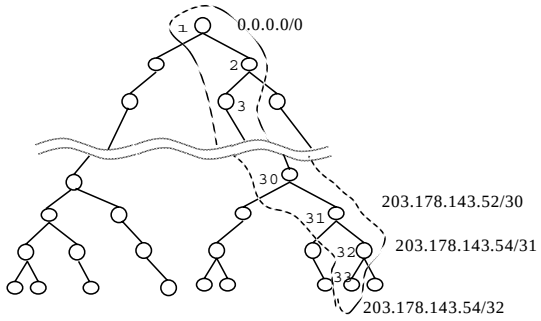

図 2.1. データ収集によって構築された IP アドレスツリー

データを収集しただけではなく、以下に示す IP アドレス空間を送信元とする 80 バイトのデータを収集したことにもなる。

```
 1:          0.0.0.0/0
 2:        128.0.0.0/1
 :               :
32:   203.178.143.54/31
33:   203.178.143.54/32
```

このように、1 パケット毎にツリー構造にデータを保存する。これを全パケットに対して行う事によって、IP アドレスから IP アドレス空間を算出することが出来る。

2.3.2 データ蓄積部

AGURI では収集したデータを、送信元 IP アドレス、受信先 IP アドレス、送信ポート番号、受信ポート番号の 4 種類のツリーに蓄積し、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。AGURI では、このサマリを更に時間をもと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 2.2 を用いて、サマリが更新される時間を「5 分」

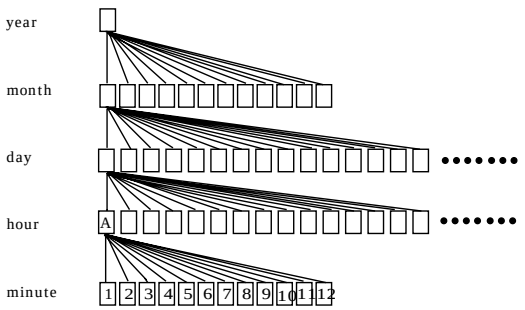

図 2.2. 異なる時間粒度のサマリ作成

と仮定した場合の時間粒度の議論を行う。図 2.2 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 5 分間のネットワーク状況を表したサマリ A は、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。

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、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。

2.4 実装

この章では AGURI の実装について説明する。AGURI は大きく分けて、データ収集部、データ集約部、データ出力部からなる 3 つの部分で構成されている。これらの構成要素について述べた上で、更に AGURI の出力を利用したツール群について説明する。AGURI は C 言語、ツール群は Perl 言語を用いて実装した。開発環境は FreeBSD4.2 である。

2.4.1 データ収集部

データ収集部では、ネットワーク上のパケットを収集するために LBNL (Lawrence Berkeley National Laboratory) のパケットキャプチャーライブラリである libpcap[105] を用いた。このライブラリを用いることによって、多くの UNIX プラットフォームで IP 層でのパケット収集を行うことができる。

AGURI では収集したパケットから以下に示す 7 つの要素を抽出する。

- (1) データ量 (2) 送信元 IP アドレス
  (3) 受信先 IP アドレス (4) IP バージョン番号
  (5) プロトコル (6) 送信ポート番号
  (7) 宛先ポート番号

これらの要素を更に以下の 4 つに分類し、それぞれの要素の組に対して「データ量」をノードの保持する値としてツリー構造に蓄積する。

1. IP バージョン番号 + 送信元 IP アドレス
2. IP バージョン番号 + 受信先 IP アドレス
3. IP バージョン番号 + プロトコル + 送信ポート番号
4. IP バージョン番号 + プロトコル + 受信ポート番号

AGURI では収集したパケットの情報を 4 つの属性に分類し、複数の属性からフローを検知することを可能にしている。

2.4.2 データ集約部

収集したパケットを蓄積し続けた場合、以下の 2 つの欠点が生じる。

1. フローの動的検知が困難になる
   木構造中に存在するノード数が増加するにつれて、計算の処理に時間がかかり、トラフィックの多いフローを抽出するまで時間がかかる。そのため、短期的なトラフィックモニタリングの効果を減少させる。
2. 保存するデータサイズが大きくなる
   木構造中に存在するノード数が増加するにつれて、保存するデータサイズ増大する。そのため、長期間に渡ってデータを蓄積することが難しくなる。

この問題を解決するために、AGURI では 1) パトリシア木アルゴリズムと、2)LRU (Least Recent Used) を用いた。パケットを収集しツリー構造にデータを蓄積する時に、単純な 2 分木ではなくパトリシアツリーを用いることによってツリー構造全体のノード数を少なくした。更に、LRU を用いてツリー構造中のノード数を一定に保った。この工夫によって、AGURI を短期から長期に渡って利用できる。しかし、一方 LRU を用いることで情報の精度が減少している。この点に関して評価を行う必要がある。

2.4.3 データ出力部

データを収集、集約し続けている AGURI に対して “HUP シグナル” を送ることによって、その時点まで収集したネットワークサマリを出力する。この出力するネットワークサマリはテキスト形式であり、AGURI の出力結果を用いたスクリプト等との親和性が高い。図 2.3 に AGURI の出力例を示す。

2.4.4 AGURI を用いたツール群

AGURI を用いたスクリプト群は、現在以下の 3

```
%%!AGURI-1.0
%%StartTime: Thu Mar 01 00:00:00 2001 (2001/03/01 00:00:00)
%%EndTime:   Sun Apr 01 00:00:00 2001 (2001/04/01 00:00:00)
%AvgRate: 14.91Mbps

[src address] 4992392109177 (100.00%)
0.0.0.0/0       87902964189 (1.76%/100.00%)
 0.0.0.0/1      206637364377 (4.14%/14.78%)
 0.0.0.0/2      205796877844 (4.12%/7.12%)
   60.0.0.0/6   97928228974 (1.96%/3.00%)
        62.52.0.0/16    51875058871 (1.04%/1.04%)
    64.0.0.0/8  100831910967 (2.02%/3.51%)
     64.0.0.0/9 74610984109 (1.49%/1.49%)
 128.0.0.0/2    142349668983 (2.85%/13.33%)
  128.0.0.0/3   197067746696 (3.95%/10.48%)
   128.0.0.0/5  202911635757 (4.06%/5.45%)
    133.0.0.0/8 69142535628 (1.38%/1.38%)
                150.65.136.91   54123094932 (1.08%)
  192.0.0.0/4   212653628837 (4.26%/38.41%)
   192.0.0.0/6  88855538654 (1.78%/1.78%)
    202.0.0.0/7 235853368912 (4.72%/14.70%)
    202.0.0.0/8 65873946376 (1.32%/8.09%)
     202.0.0.0/9        117196493427 (2.35%/6.77%)
                202.12.27.33    160473669718 (3.21%)
             202.30.143.128/25  60239291958 (1.21%/1.21%)
                203.178.143.127 94031811680 (1.88%)
   204.0.0.0/6  228960094456 (4.59%/17.68%)
    204.0.0.0/8 125458765333 (2.51%/7.58%)
                204.123.7.2     87103414877 (1.74%)
                204.152.184.75  165733431144 (3.32%)
    206.0.0.0/7 164036959478 (3.29%/5.51%)
     206.128.0.0/9      53526598302 (1.07%/1.07%)
    207.0.0.0/8 57628266965 (1.15%/1.15%)
  208.0.0.0/4   282590640975 (5.66%/31.72%)
   208.0.0.0/6  116047154301 (2.32%/22.20%)
    208.0.0.0/7 249800961034 (5.00%/16.79%)
    209.0.0.0/8 140888988219 (2.82%/11.78%)
                209.1.225.217   238192306019 (4.77%)
                209.1.225.218   209160635530 (4.19%)
    210.0.0.0/7 154008321340 (3.08%/3.08%)
     216.0.0.0/9        192899750315 (3.86%/3.86%)
%LRU hits: 86.82% (1021/1176)
```

図 2.3. AGURI の出力例

つがある。

- agurify.pl
  2 分間隔で現在起動している AGURI プログラムに “HUP シグナル” を送ることによって、その時のネットワークサマリをファイルに保存する。そのファイルの時間を読み込んで、時間に適したディレクトリにファイルを保存する。例えば、2001 年 4 月 1 日 13 時 30 分から 2001 年 4 月 1 日 13 時 32 分のデータの場合、“aguri.log/2001/04/01/13” というディレクトリの中に “20010401.1330.agr” というファイル名で保存される。更に、1 時間毎、1 月毎、1 年毎、のネットワークサマリを自動作成し時間粒度に適したディレクトリにサマリファイルを保存する。
- makeplot.pl
  makeplot.pl は gnuplot を内部で用いたスクリ

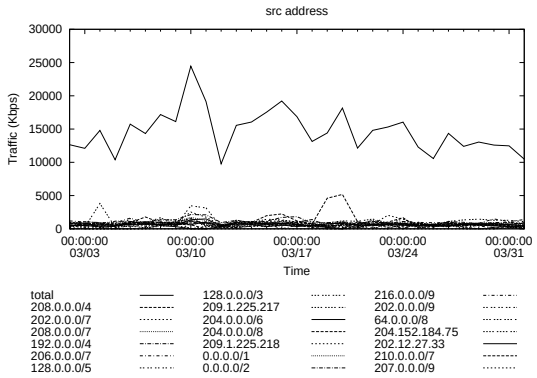

図 2.4.　送信元 IP アドレス

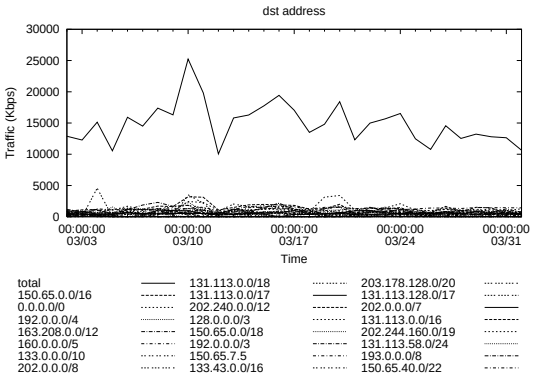

図 2.5.　受信先 IP アドレス

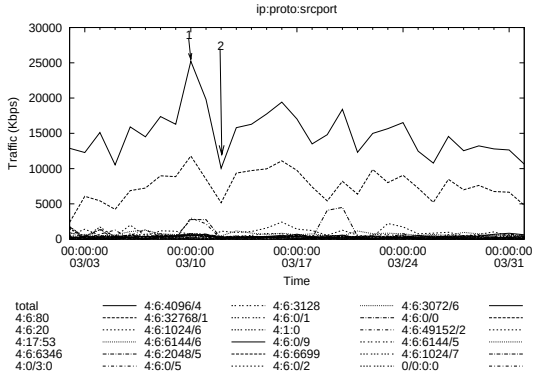

図 2.6.　送信元ポート番号

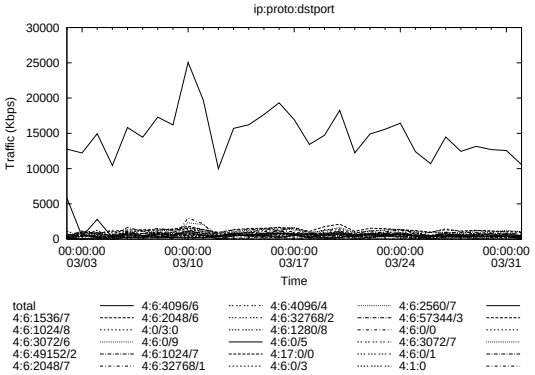

図 2.7.　受信先ポート番号

プトである。ある時間の中でトラフィック量の多いフローを折れ線グラフにして視覚化する。

- density.pl

  density.pl は IPv4 のアドレス空間のトラフィック密度を視覚化するスクリプトである。トラフィックパターンには特定の傾向がある場合が多いため、定常的なネットワーク状態では出力図はあまり変化しない。しかし、急激にトラフィックパターンが変わった場合、出力図の変化が顕著に現れる。

2.5 AGURI による WIDE 国際線のデータ収集

WIDE プロジェクトでは、AGURI を用いて、実際に国際線のトラフィックの傾向を収集し続けている。

2.5.1 トラフィック傾向

本章では、WIDE インターネット国際線のトラフィック傾向を示す。図 2.4、図 2.5、図 2.6、図 2.7 は、AGURI で 2001 年 3 月の国際線トラフィックを収集し、スクリプト群を用いて視覚化したものである。図 2.6、図 2.7 中に出て来る “4:6:80” とは IP バージョンが 4、プロトコル番号が 6 (つまり TCP)、送信元ポート番号が 80 (つまり HTTP) ということを示している。

図 2.4、図 2.5、図 2.6、図 2.7 に示した出力図は 2 つの情報を持っている。

- 折れ線グラフ

  回線を占めているトラフィックの属性を視覚的に見ることができる。今回取り上げた WIDE インターネットの例では、図 2.6 に示した HTTP データの割合を把握できる。また同様に、図 2.6 を別の視点から捉えた場合、全トラフィックの中で一つの IP アドレス空間や HTTP を除く他の特定のポート番号を使用したアプリケーションによってトラフィックが圧迫されていないということが分かる。

- 項目

  折れ線グラフの下にリストアップされる項目数

表 2.1. 識別された IP アドレスとポート番号

graph	address	host name / port number
	209.1.225.217	vip3.intl.geocities.yahoo.com
src	209.1.225.218	vip4.intl.geocities.yahoo.com
addr	204.152.184.75	ftp.netbsd.org
	202.12.27.33	m.root-servers.net
	150.65.0.0/16	jaist.ac.jp
dst	150.65.7.5	ftp.jaist.ac.jp
addr	131.113.0.0/16	keio.ac.jp
	4:6:80	HTTP
src	4:6:20	FTP
port	4:17:53	DNS
	4:6:6346	GNUTELLA

は、AGURI によって設定することができる。この項目は全トラフィック中の占有率順にリストアップされるため、回線を使用している組織や使われているアプリケーションを検知することができる。表 2.1 に、図 2.4、2.5、2.6 中にリストアップされている IP アドレス空間やポート番号の対応を示す。このように IP アドレスからは、特定のホストや特定の組織の識別が可能であり、また、ポート番号からは特定のポートを使用するアプリケーションを検出ができる。なお、図 2.7 には特定のポート番号がリストアップされなかったことから、特定の受信先ポートを使用したアプリケーションのデータの全トラフィック中での占有率が低いことがわかる。

以上のように、折れ線グラフで表された情報とリストアップされた項目から、全トラフィックを構成している特徴的な要素を抽出することができる。

2.5.2 詳細な傾向

AGURI は異なる時間粒度でのトラフィックモニタリングを実現している。そのため、粗い時間粒度でトラフィック傾向を視覚化し、その中で特徴的な時間に対して更に細かい粒度でトラフィックの視覚化を行える。

図 2.6 中に矢印で示した “1” から “2” への総トラフィック量の急激な減少に着目した。この時間帯の出力を図 2.8 に示す。この図 2.8 は図 2.6 中に示した “1” から “2” の時間帯を詳細に示した出力図である。図 2.8 中に矢印で示した “3” から “4” の期間中、

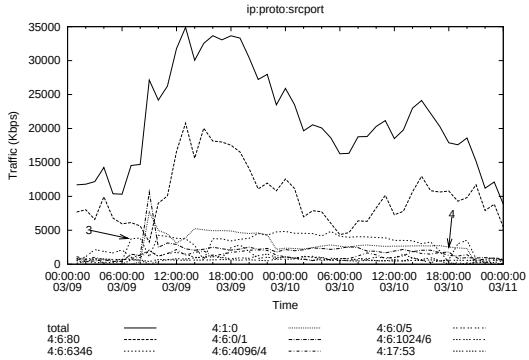

図 2.8. 送信ポート番号の詳細な傾向

“4:6:6346” (GNUTELLA) と “4:1:0” (ICMP) がそれぞれ 5Mbps 弱のデータ量を占めている。このため、HTTP のデータ量は普段の日とさほど変化が存在していないのに総トラフィック量が急激に増えている。つまり、図 2.6 中の “1” から “2” への総トラフィック量の急激な減少は、“2” の時間にトラフィック量が減少したのではなく、“1” の時間に急激にトラフィック量が増加していたために起こった。

このように、細かい粒度で視覚化することによって、より詳細なトラフィック傾向を把握することを可能にした。

2.6 結論

本報告書では、トラフィックモニタリングツール AGURI の設計、実装、並びに AGURI を用いた WIDE インターネット国際線のトラフィック傾向を述べた。WIDE インターネットのような広域なネットワークを運用し続けていくためには、トラフィックモニタリングを多地点、かつ長期間行い、

ネットワークの現状に適した通信機器の設置、設定を行う必要がある。しかし、現存するネットワークモニタリングツールは長期に渡ってトラフィックの傾向を収集し続けることが難しい。そこで、本研究では収集したトラフィックを効果的に集約することによって、ネットワークの特徴を抽出することのできるトラフィックモニタリングツール AGURI の設計、実装を行った。作成した AGURI は、収集したデータを効率的に集約することによって、短期的なトラフィックモニタリングから長期に渡るトラフィックモニタリングまでを可能にした。

実際に AGURI を用いて WIDE インターネット国際線でデータを収集し、対象とした国際線のトラフィックの傾向を明らかにした。WIDE プロジェクトでは、AGURI の開発をすすめると共に、WIDE インターネットのバックボーンにおいて AGURI を運用し続けている。今年度以降の WIDE 報告書における WIDE バックボーントラフィックの収集、解析の報告は AGURI を用いた結果を使用する予定である。

第 3 章　カーネルモニタリングシステムの設計と実装

3.1 カーネルモニタリングシステムの実装

サーバシステムの計測手法はこれまでに様々な手法が提案、実装されている。これまで述べてきたパケットモニタリングによる手法は、サーバシステムに負荷を与えずに計測できる手法として有効である。しかし、サーバ管理者がサーバシステムのボトルネックを調査する場合、パケットモニタリングだけでは不十分である。サーバシステムのボトルネックを調査する場合、カーネルモニタリング手法を用いて調査する。これまで、カーネルモニタリング手法は、サーバシステムに負荷を与えるため有効でないと考えられてきた。そこで本章ではサーバシステムの性能に影響の少ないカーネルモニタリングシステムの提案、実装、評価について述べる。

3.2 あらまし

情報化社会の急速な発展により、インターネットの利用者が急速に増加している。中でも World Wide Web (WWW) は携帯電話や PDA などからも利用できるほど、社会に広まっている。このようなインターネットユーザの急速な増加により、WWW サーバの数も同様に増加している。ユーザの増加により、より良いサービス品質が望まれる。サーバ数の増加、サービス品質の向上によりサーバ管理者の負担は増大している。

このような要求があるにもかかわらず、サーバ管理者は自身の経験とそれに基づいた勘を頼りにサーバを管理している。その理由は、サーバの性能を運用中に容易に知る事が出来ないからである。そこで我々は ENMA を開発し、運用中にサーバの性能を知るためのシステムを構築した。しかし、次に問題となった項目は、サーバシステムのボトルネックの調査である。ENMA はパケットモニタリングを基に計測するため、サーバシステム内部の出来事を観測できない。これまで、サーバシステムの内部情報を抽出するにはカーネルモニタリングを行ってきた。しかしカーネルモニタリングは負荷が高く実用的でないと考えられてきた。そこで我々は、サーバシステムの性能に影響をほとんど与えないカーネルモニタリングシステム (rep2) を設計、実装した。そしてその有効性を示すために実際に運用されているサーバシステムに適用した。

本章では ENMA と rep2 を組み合わせ、サーバ外部および内部の観測結果から、サーバ管理者がとるべき行動について議論した。

3.3 従来システム

サーバシステムの内部情報としては、計算機依存部分 (CPU や memory に関する使用量など) および、ネットワークに関する指標 (ファイルディスクリプタ、トラフィック、パケット数など) などがある。

vmstat
仮想メモリの統計情報を表示するプログラム。リアルタイムにディスプレイに表示する。CPU 利用率 (アイドル率、ユーザ使用率、カーネル使用率)、メモリ利用率、ページの状況などがわかる。欠点としては、タイムスタンプが残らない、計測と表示をサーバシステム内部で行うためログを残すことが困難であるなどが挙げられる。

netstat
ネットワークシステムの統計情報を表示するプロ

グラム。オプションの指定により、様々な情報を得ることが出来る。ルーティング情報、ネットワークインターフェース情報、プロトコル別統計情報、ネットワークメモリバッファの使用量、入力パケット数および出力パケット数、エラーパケット数などを得ることが出来る。netstat も vmstat 同様、タイムスタンプが残らず、ログを残すことが困難である。

ps

プロセスに関する情報を表示するプログラム。オプションの指定により、プロセス ID、親プロセス ID、CPU 利用率、実行開始時間など、様々な情報を得ることが出来る。ps は、システムで動作しているプロセスの状態のスナップショットを得るプログラムである。したがって、時間遷移のログを残すことは出来ない。

3.4 カーネルモニタリングシステムの設計

本節ではカーネルモニタリングシステムの設計について述べる。本システムを rep2 とした。カーネルモニタリングシステムは OS 依存の実装である。可能な限り OS 依存部分を減らすために、我々は kvm インターフェースを用いて実装した。rep2 は二つの要素から構成されている。カーネルモニタデーモン (rep2) とデータ収集プログラム (collector) である。(図 3.1)

従来システムの問題点は、ログ出力とモニタリングを同一ホストで行っていたことが挙げられる。これにより、ログ出力のオーバヘッドが大きくなり計測自体が高負荷なものとなっていた。そこで我々はログを別ホストで出力することによりこの問題点の解決を図る。

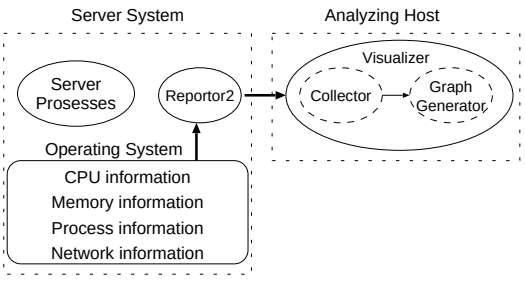

図 3.1. カーネルモニタリングシステム

3.4.1 カーネルモニタデーモン (rep2)

rep2 は collector からのリクエストに応じて、カーネル情報をネットワークに出力する。rep2 では以下の情報について観測する。

フリーメモリ
サーバシステム内部の未使用メモリ量
使用メモリ
サーバ内部で使用されているメモリ量。カーネルが使用しているメモリ量は含まれない。
プロセス数
サーバシステム内部で起動しているプロセスの総数
コネクション到着数
サーバに到着したコネクション数。
タイムスタンプ
計測した時刻。秒単位で計測する。

マルチプロセッサシステムの場合は CPU 毎の以下の項目についてモニタリングする。

アイドル時間率
CPU が休んでいる割合。この時間が多ければ、CPU にはまだ余力があると判断できる。
カーネル時間率
カーネル内部が処理している割合。この時間が多くなると CPU の余力は少ないと判断できる。
ユーザ時間率
CPU がユーザプロセスを処理している割合。
コンテキストスイッチ数
CPU で 1 秒間にコンテキストスイッチが発生している回数。コンテキストスイッチとはプロセス切替のことであるので、この回数が多いとサーバの負荷が高いと言える。
システムコール数
CPU で 1 秒間にシステムコールが発行されている回数。システムコールが発行されるとユーザ空間からカーネル空間への切り替わりが発生する。したがって、この回数が多いとサーバの負荷が高いと言える。
割り込み数
CPU に対して 1 秒間に発生している割り込みの回数。割り込みが発生するとシステム内部の割り込み処理ルーチンに処理が移る。したがって、こ

の回数が多いとサーバの負荷が高いと言える。

ページフォルト数

CPU に対して 1 秒間に発生しているページフォルトの回数。ページフォルトが発生するとシステムはメモリ確保や解放のためのルーチンに処理が移る。したがって、この回数が多いとサーバの負荷が高いと言える。

3.4.2 データ収集プログラム (collector)

データ収集プログラム (collector) は 10 秒間隔で rep2 に対してデータを要求する。得られたデータはテキストファイルとして出力され、視覚化モジュールによってグラフを生成する。サーバシステムの負荷を軽減しリアルタイムに解析するためにはサーバ外部での解析が不可欠である。rep2 は collector を用いることでデータを外部ホストへ出力し、外部ホストによる解析を可能とした。

3.5 カーネルモニタリングシステムの実装

我々は UNIX プラットホームで動作するカーネルモニタリングシステムを目標としてシステムを実装した。rep2 および collector は C 言語で実装した。

カーネル内部の情報は OS 毎に異なる。可能な限り OS 依存でない実装にするために、rep2 ではカーネル情報を抽出するために kvm インターフェースを用いた。kvm インターフェースを用いることで、Solaris 2.6 および FreeBSD 4.0 において統一した情報を得ることが可能となった。

しかし、kvm インターフェースだけでは十分にカーネル情報を引き出すことが出来ない。そこで、OS の実装によって以下に示す異なるインターフェースを用いて、カーネル情報を抽出した。

- Solaris の場合
  kstat インターフェースを用いたデータ出力。kstat インターフェースを用いて、ネットワーク関係の情報を得ることが可能。
- FreeBSD の場合
  sysctl インターフェースを用いたデータ出力。上と同様、sysctl インターフェースを用いて、ネットワーク関係の情報を得ることが可能。

3.6 実証実験

本節では、rep2 を起動することによるサーバへの負荷を調査する。そして、実際に運用されているシステムに適用した例について述べる。

3.6.1 rep2 による負荷の影響

本節では、我々の実装した rep2 がサーバの性能にどの程度の影響を与えるのか確認する。rep2 を使用した場合と使用しない場合におけるサーバシステムの応答時間を計測する。応答時間がどの程度劣化するかで rep2 のサーバに与える影響を調査する。

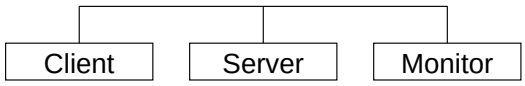

図 3.2.　ネットワーク環境

図 3.2 に、実験で用いたネットワーク環境を示す。表 3.1 に使用したソフトウェアおよびハードウェアを示す。表 3.1 に示すように、クライアントは httperf 0.8 を用い、リクエストレートは 300、総コネクション数はそれぞれ 300000 コネクションとした。サーバは apache 1.3.12、応答時間の計測のために ENMA を用いた。OS は全て FreeBSD 4.1.1 である。

図 3.3 に測定結果を示す。図 3.3 は rep2 を用いた場合と用いない場合の両方の場合における応答時間の累積分布である。横軸に応答時間、縦軸に累積分布をとっている。図 3.3 からわかるように rep2 を用いる場合と用いない場合で、ほとんど差は見られない。これから、我々の提案した rep2 はサーバの性能にほとんど影響を与えないと言える。

表 3.1.　実験構成

ホスト	OS	CPU	メモリ	アプリケーション
Client	FreeBSD 4.1.1	Pentium III 700MHz	256M Bytes	httperf 0.8
Server	FreeBSD 4.1.1	Pentium III 700MHz	256M Bytes	apache 1.2.12
Monitor	FreeBSD 4.1.1	Pentium III 700MHz	256M Bytes	enma

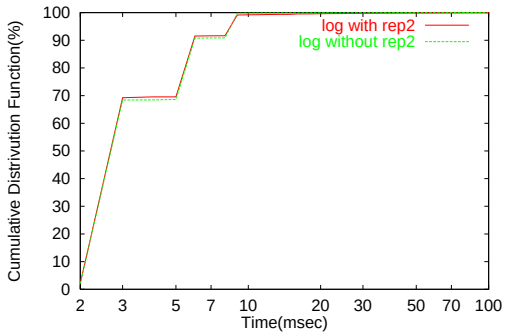

図 3.3. 応答時間の累積分布

3.6.2 実際のシステムへの適用例

rep2 の有効性を確認するために実際に運用されている WWW サーバに対して適用した。第 82 回全国高校野球選手権大会のインターネット中継に用いられたサーバを観測した。このサーバでは 1 日に最大約 4600 万アクセスが得られた。

システム構成

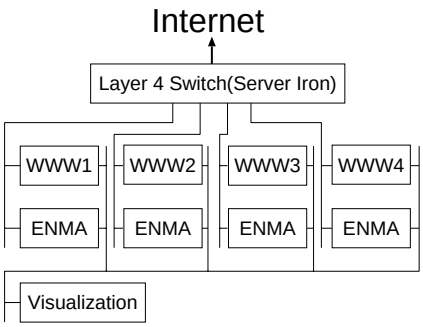

図 3.4. ネットワーク構成図

図 3.4 にネットワーク構成図を示す。図 3.4 に示すように、4 台のサーバを用意した。上流には Foundary 製 Layer 4 Switch を用い、ラウンドロビン方式で負荷を分散した。

WWW サーバや観測に用いた計算機のスペックおよびアプリケーションを表 3.2 に示す。表 3.2 に示すように、WWW1 は apache を用い、WWW2〜WWW4 では、chamomile と呼ばれる独自実装のサーバアプリケーションを用いた。

chamomile では、シングルプロセス、マルチスレッドアーキテクチャが採用されている。これは主にマルチスレッドのコンテキストスイッチによるオーバヘッドを軽減するためである。

また、図 3.4 に示すように、ENMA を用いて外部より観測した。

観測された全てのデータは制御線を用いて視覚化ホストへ転送され、リアルタイムにデータの視覚化が行われた。

3.6.3 結果および考察

本節では、大会期間中最もアクセスの集中した時間帯 (8/21 の 14:45〜15:45) の解析結果について述べる。

ENMA による観測結果

図 3.5 に示すように、4 台合計のセッション数は約 6 万本である。コネクション到着数は 4 台合計で約 1800/秒である。同様に、コネクション処理数も約 1800/秒である。そして、トラフィックは 4 台合計で約 70MBytes/秒である。

コネクション到着数とコネクション処理数の差はほとんど見られないので、サーバシステムは通常状態であると言える。

rep2 による内部観測

- プロセス数

  図 3.6 に apache と chamomile のプロセス数の遷移の結果を示す。apache はマルチプロセスによる実装のため、プロセス数が上下に揺らいでいる。しかし、chamomile はシングルプロセスな実装のため、プロセス数の揺らぎはない。

  プロセス数が増加すると、OS 内部でのプロセス管理表が増大するためシステムが利用できるメモリ数が減少すると考えられる。

表 3.2. システム構成

	WWW server 1	WWW server 2-4	ENMA host
OS	Solaris 2.7	Solaris 2.7	FreeBSD 4.0
CPU	UltraSPARCII 400MHz	UltraSPARCII 400MHz	PentiumIII 700MHz
Memory	2 GBytes	2 GBytes	256 MBytes
Application	Apache 1.3.12	chamomile	enma

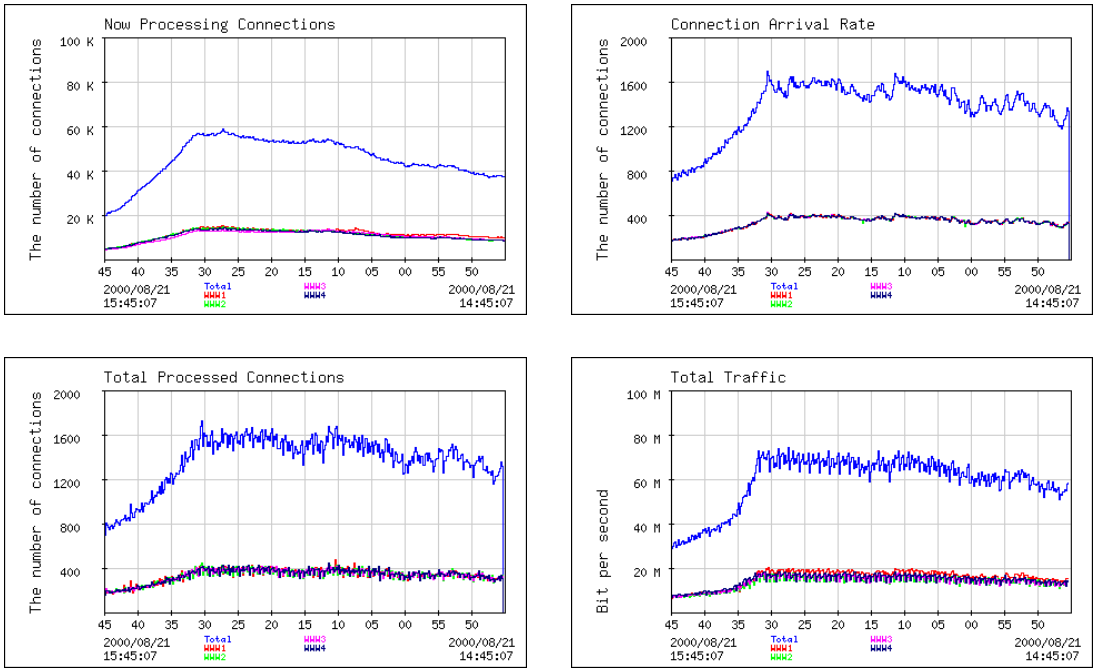

図 3.5.　4 台合計の結果

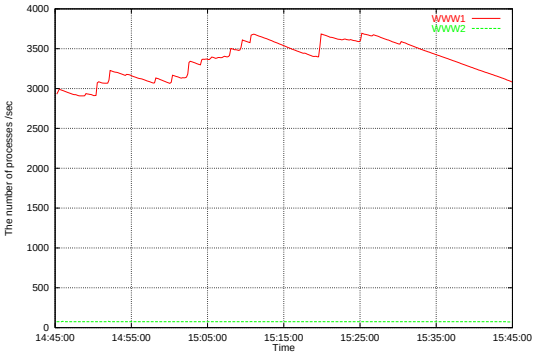

図 3.6.　プロセス数の遷移

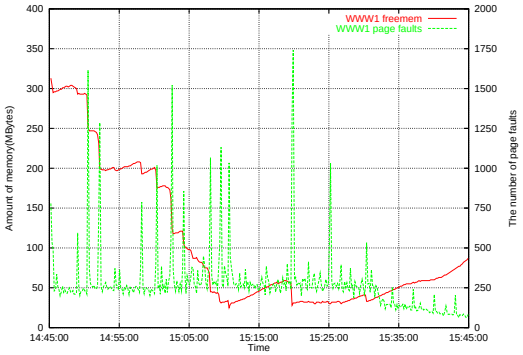

図 3.7.　メモリ量とページフォルト数の遷移

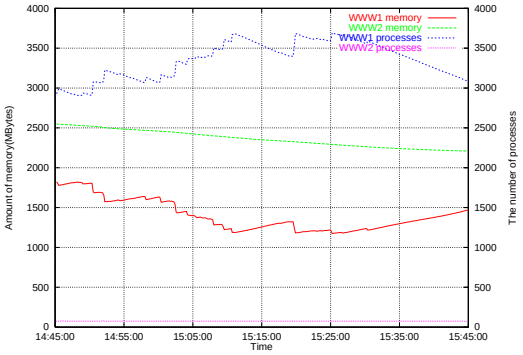

図 3.8.　メモリ量とプロセス数の遷移

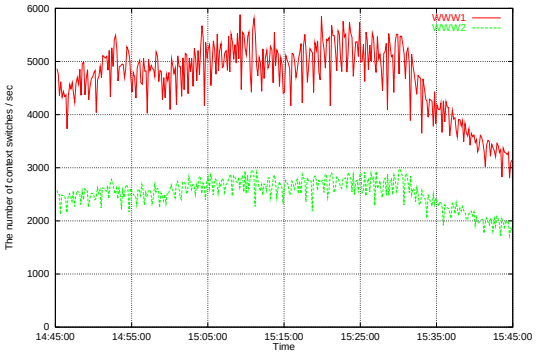

図 3.9.　コンテキストスイッチの遷移

- メモリとページフォルトの関係

  図 3.7 に apache のメモリ量とページフォルト数を示す。図 3.7 に示すように free memory が減少してくると、ある閾値から突然ページフォルトが発生している。

  このページフォルトはシステム内部の再利用されていないメモリを検索するために発生していると考えられる。そのため、ページフォルトに

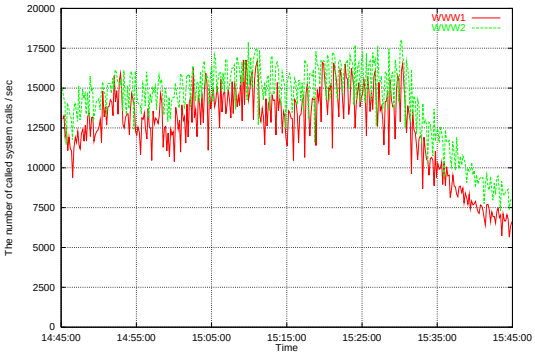

図 3.10. システムコール数の遷移

図 3.11. 割り込み数の遷移

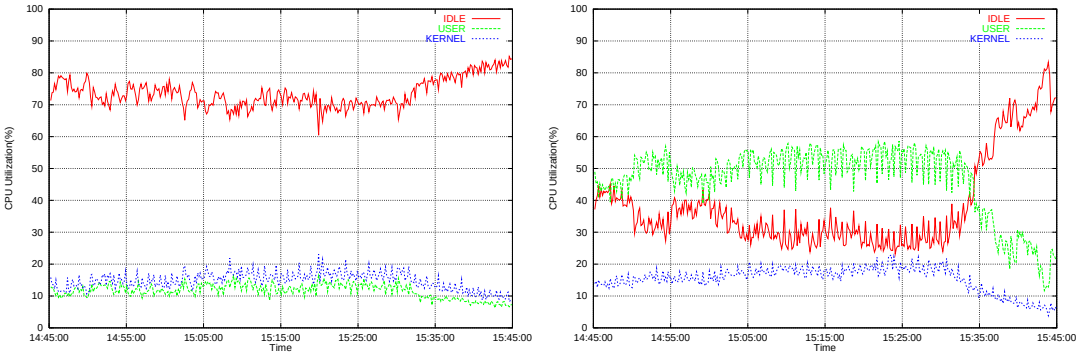

図 3.12. CPU 利用率の遷移

よるメモリの確保とリクエストの到着とのバランスが崩れる時、サーバは飽和状態になると考えられる。

- メモリ量とプロセス数の関係

  図 3.8 にメモリ量とプロセス数を示す。図 3.8 に示すように、WWW1 では、プロセス数が増加するとメモリ量が減少している。これは、プロセス数が増えると、OS 内部でのプロセス管理テーブルが大きくなり、ユーザプロセスの使用できるメモリが減少するからであると考えられる。

  一方、WWW2 ではプロセス数がほとんど変化していないので、メモリ量もあまり変化していない。

- コンテキストスイッチについて

  図 3.9 に WWW1 および WWW2 のコンテキストスイッチの遷移を示す。図 3.9 に示すように、WWW1 の方が WWW2 よりも明らかにコンテキストスイッチの数が多い。これは WWW1 の方が多くのプロセスを起動しているからであると考えられる。

- システムコールについて

  図 3.10 に WWW1 および WWW2 のシステムコール発行数の遷移を示す。図 3.10 に示すように WWW1 と WWW2 では WWW2 の方が若干システムコールの発行数が多くなっている。chamomile はマルチスレッドを用いた実装であるため、スレッド間の共有変数をロックする必要がある。このロックによって apache よりもシステムコールの発行数が多くなったと考えられる。

- 割り込みについて

  図 3.11 に WWW1 および WWW2 の割り込み発行回数の遷移を示す。図 3.11 に示すように、WWW1 と WWW2 ともに同様の値を示している。リクエスト到着数がほぼ同じなので、パケット到着による割り込みの発生も同じになったと考えられる。

- CPU 利用率について

  図 3.12 に WWW1 および WWW2 の CPU 利用率を示す。図 3.12 に示すようにカーネル利用率は WWW1 および WWW2 は同様の値で遷移している。しかし、ユーザ利用率は chamomile の方が高く、アイドル率は apache の方が高くなっている。文献 [7] によると、カーネル利用率が全体の 90%近くになったら飽和すると報告

している。これから、WWW1 および WWW2 はまだ十分余力を残していると考えられる。

考察

- ページフォルトと free memory の残りに注意を払えば、メモリ不足による飽和状態を防止できる。
- 今回の計測では、WWW1 においてもう少しリクエストレートが上昇したら、飽和状態に移ると考えられる。これは、free memory が少なくなり、OS がページフォルトを発行してメモリを確保していたからである。リクエストレートが高くなると、ページフォルトによるメモリの確保が間に合わなくなり、スラッシング状態になると考えられる。
  この状態を防ぐには、ページフォルトの抑制が不可欠である。apache システムでは fork システムコールの発行を抑制する事でページフォルトの発行が抑制されると考えられる。
  chamomile でこの現象が現れなかったのはシングルプロセスによる実装のため、システムが消費するメモリ量を抑制できたからであると考えられる。
- WWW1 ではプロセスを生成し過ぎでシステムの領域が多く確保されてしまって、free memory が少なくなった。これは非効率的であり、apache システムの問題点であると言える。

3.7 管理者の行動に関する議論

前節の結果を踏まえて、サーバ管理者の管理業務とその行動について議論する。サーバ管理者の管理業務には大きく分けて 2 つある。

3.7.1 障害発生時における障害回避

サーバ管理者はサーバシステムに障害が発生した時、障害の生じている項目について調査しその問題を回避しなければならない。ハードウェア的な故障であれば、部品を交換しなければならない。ハードウェア的性能限界まで処理が到達していない場合、パフォーマンスチューニングによって、システムの性能限界を向上できる。

カーネルモニタリングの結果を観測していると、システムのボトルネックや障害は容易に発見できる。しかし、サーバシステムがハードウェアの性能限界まで処理しているかどうかの判断は困難である。

3.7.2 リクエスト到着予測によるシステムの拡張 (キャパシティプランニング)

サーバ管理者は上述した障害の回避よりも、障害を発生させないことに注意を払う必要がある。最も多く発生する障害は、サーバシステムの性能限界を超えてリクエストが到着することである。このため、サーバ管理者は長期間の計測から将来どの程度のリクエストが到着するかの予測を立てる必要がある。現行システムの性能限界と新システム導入後の性能限界を把握しなくてはならない。そして新システム導入の際にはコスト算出する必要がある。

サーバシステムの性能は様々な要因がある。大きく分けると以下に示す 4 つに分類できる。(1) ネットワーク帯域 (2) ハードウェア能力 (3)OS(4) アプリケーション。ネットワーク帯域が不足する場合新たに帯域を購入するか、キャッシュシステムの導入によりボトルネックを解消できる。しかしユーザ数の増加率とコスト算出といったトレードオフから明確な指針は存在しない。

ハードウェア能力は処理能力に限界が到達したなら、新たなハードウェアを購入する以外に性能向上はあり得ない。しかし、OS やアプリケーションのパフォーマンスチューニングにより、サーバシステムの性能向上が見込まれることがあるため、一概に新たなハードウェアの購入が良い選択であるとは限らない。OS やアプリケーションのパフォーマンスチューニングは更に複雑である。前節でも述べたように、アプリケーションの設計方針が変更されるとシステム資源の消費量が劇的に変化する。そのため OS のパフォーマンスチューニングの指針が以前の指針と大いに異なる場合がある。これらから言えることはサーバシステムを構築する場合、始めにネットワーク帯域、ハードウェアの選定を厳密に行う必要があるということである。これは、OS やアプリケーションはネットワーク帯域やハードウェアに依存しているからである。

3.8 今後の課題

3.8.1 キャパシティプランニング

サーバ管理の支援には 2 つの要素があると考えら

れる。(1) サーバシステムの現状把握 (2) 将来のサーバシステム拡張の方針。これまで我々は (1) に重点を置いて議論してきた。しかし、これだけではサーバ管理支援という立場では不十分であると考えられる。そこで、(2) に関して議論する。現在、キャパシティプランニングには以下の要因から評価が困難である。

- リクエスト到着予測の困難さ (ユーザの増加率の算出)
  長期的観測をすればある程度は予測可能と思われる。しかし、新たなサービスを開始する時にユーザがどの程度存在するかの調査は容易ではない。
- ハードウェア的な性能限界の把握の困難さ
  様々なベンチマークシステムにより、性能限界の調査は容易にはなっている。しかし、ベンチマークによる結果と現実に発生する現象とは異なる。このため、ある条件下での性能限界を知ることは出来ても、実際の環境での性能限界の把握は困難である。
- 導入後のシステムに対する妥当性
  プランニングした後のシステムに対する評価方法が確立されていない。基本的に払われたコストに対してどの程度の性能向上したかが判断できれば良い。しかし、先に述べたように性能限界の調査が容易ではないため、新しいシステムに対する評価も困難である。

3.9 まとめ

WWW サーバ内部のボトルネックを調査する場合、カーネルモニタリングが不可欠である。これまで、カーネルモニタリング手法は、サーバシステムに負荷を与えるため有効でないと考えられてきた。従来のカーネルモニタリングシステムの問題点はログ出力と計測が同一ホストで行われている事が考えられる。これにより負荷が高く実用的でなかった。そこで我々は計測とログ出力を異なる計算機で実行できるシステムを構築した。

我々は、新たなカーネルモニタリングシステムを実際に運用されているサーバシステムに適用し、有効性を示した。また、ENMA と rep2 を組み合わせ、サーバ外部および内部の観測結果から、サーバ管理者がとるべき行動について議論した。

第 4 章　tcpdump 関連ツールの開発

4.1 はじめに

ネットワークを流れるパケットを収集 (ダンプ)・解析し、ネットワークの性能管理・障害管理・セキュリティ管理等を行う研究が進められている。パケットのダンプに特化した専用の測定器も市販されているが、通常の計算機上で利用可能な tcpdump が現在広く用いられており、またダンプデータの共通フォーマットとしての地位を確立しつつある。

ここでは tcpdump およびダンプデータに関する話題として、1) プライバシー情報の保護、2) 全二重回線の観測を取り上げる。それぞれを実現するため開発したツールを紹介すると共に、その他 tcpdump に関係の深い一連のツールを取り上げる。

4.2 プライバシー情報の保護

4.2.1 背景

tcpdump に代表されるツールを使ってネットワークを流れるパケットをダンプし、ネットワークの性能管理・障害検出・セキュリティ管理を行う研究が行われている。しかし、ダンプデータにはパケットを生成した送受信者・アプリケーション・実際に伝送されたデータ等、プライバシー情報を多く含み、その取り扱いに細心の注意が必要となる。これは不特定多数の人間がダンプデータにアクセスできる環境や、ダンプデータをコピーした場合は、さらに大きな問題となる。

そこでダンプデータに含まれるプライバシ情報の保護を実現する tcpdpriv が開発されている。このプログラムは IP アドレスやポート番号を他の番号に変換することによって、元の値を特定不能にする。しかし複数のダンプファイルを変換した場合、ダンプファイル間には変換の一貫性が無いという欠点を持つ。

そこで変換に用いた変換表の保存、読み込み機能を付加することによって、一貫性を維持したまま複数ファイルを変換可能とした tcpdpriv type R を開発した。本節ではその報告を行う。

4.2.2 tcpdpriv を用いたプライバシー情報の保護と問題点

tcpdpriv は tcpdump のダンプデータ内に含まれる IP アドレス、ポート番号を変換する機能を持つプログラムである。tcpdpriv を用いた IP アドレスの変換の一例を図 4.1 に示す。オリジナルのダンプデータ内に新しい IP アドレスが出現する度、カウントアップする通し番号が割り当てられ変換が行われる。変換された結果は再度ダンプデータとして保存することが可能である。またファイル内においては、変換に一貫性があることが保証される。

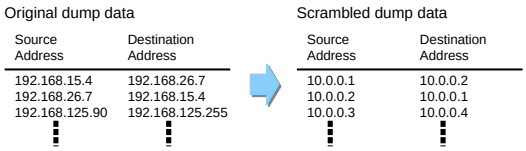

図 4.1.　tcpdpriv による変換例

tcpdpiv ではダンプデータに依存した変換が行われるため、高いプライバシー保護が実現されるが、一方で複数のダンプデータに対して同一の変換を行うことができない。次に示す例はファイル 1 、ファイル 2 に共通して含まれた IP アドレス (192.168.0.1) が異なるアドレスに変換されてしまった例を示している。

- ファイル 1 に対する変換
  192.168.0.1 → 変換 → 10.0.0.2
- ファイル 2 に対する変換
  192.168.0.1 → 変換 → 10.0.3.1

観測結果が複数のファイルに別れていた場合、ファイル間の一貫性が維持できないことが、その後の処理を困難にする恐れがある。

4.2.3 tcpdpriv type R の開発

tcpdpriv type R は tcpdpriv に変換表の保存、読み込み機能を追加したものである。tcpdpriv はアドレス変換のためプログラム内部にバイナリツリー (変換表) を構築するが、tcpdpriv type R はこの変換表を入出力することができる。

tcpdpriv type R では新たにオプション -m が追加され、ファイル名を指定することによってアドレス変換に用いた変換表が書き出される。別のダンプデータに対して同一の変換を行いたい場合、使用したい変換表を -m で指定し読み込ませると、プログラム内部にバイナリツリーが再構築され同一の変換が行われる。新規の IP アドレスが出現しバイナリツリー (変換表) が成長した場合、プログラム終了時に自動的に同じファイル名で保存が行われる。

実際の使用例を以下に示す。

1. 新規 IP アドレスが出現するたび、カウントアップする通し番号に変換。ポート番号はそのまま。
   prompt% tcpdpriv -A0 -P99 -m maptable -r in_file -w out_file
2. 新規 IP アドレスが出現するたび、IP アドレスの上位 16bit と下位 16bit を別々のカウンタでカウントアップする通し番号に変換。TCP ポート番号のみ通し番号に変換 (UDP ポート番号はそのまま)
   prompt% tcpdpriv -A1 -T0 -m maptable -r in_file -w out_file

tcpdpriv type R は tcpdpriv が構築する全てのバイナリツリーを保存し読み込むことが出来るため、tcpdpriv で提供されている全ての変換機能 (アドレス変換、ポート番号変換) に対しても、一貫性を維持した変換を実現可能にする。

4.2.4 実験および評価

ネットワークから収集したダンプデータを用いて、tcpdpriv type R の評価を行った。あるネットワークから 30 分毎に区切ったダンプデータを 1 週間分収集した。この合計 336 個のダンプデータに対して、tcpdump type R による IP アドレスの変換を行った。図 4.2 は、336 個のファイル (横軸) のそれぞれ

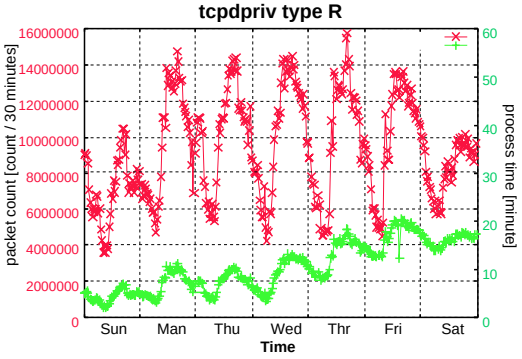

図 4.2.　tcpdpriv type R の処理速度の評価例

について、1 つのダンプデータに含まれていた総パケット数 (縦軸左)、1 つのダンプデータの変換にかかった処理時間 (縦軸右) を示したものである。全てのファイルについて 30 分以下の時間で処理が行われるため、リアルタイム処理が可能であった。また、時間が経過するにつれ、バイナリツリー (変換表) が成長し、これに伴い処理時間が増加していく様子が示されている。

4.2.5 まとめ

本ツールを用いることにより、複数のダンプデータに対して一貫性を維持した変換を行うことが可能となる。また変換表と変換済みのダンプデータを分けて保存することにより、プライバシ情報を含んだ情報とそれ以外を別々に管理できる。しかし一方で、変換表を用いれば逆変換も可能であるため、変換表を厳重に管理する必要がある。

今後、変換時のコマンドライン引数の保存機能、変換されたダンプデータと変換表を用いた逆変換機能、IPv6 への対応を予定している。

4.3 ダンプファイルの連結

4.3.1 背景

高速化のため全二重 (FullDuplex) 通信を行うネットワークが普及している。全二重回線では双方向の通信を同時に行うため (図 4.3)、タッピングキットと呼ばれる装置を使ってパケットを観測、収集する手法が存在する。しかし通信の方向毎に別々にダンプデータが収集され、ダンプデータの取り扱いが煩雑になる恐れがある。

そこで 2 つのダンプファイル含まれるパケットを、パケットの到着時刻 (タイムスタンプ) に基づきソートしながらマージし、1 つダンプファイルに保存しなおすツール tcpdmerge を開発した。本節ではその報告を行う。

4.3.2 ダンプデータの結合と問題点

tcpdump によってダンプデータを収集する場合、一定期間ごとにダンプファイルを分割し、データの

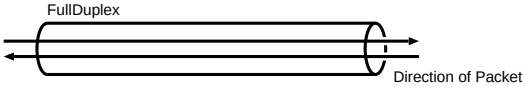

図 4.3. 全二重 (FullDuplex) 回線の概念

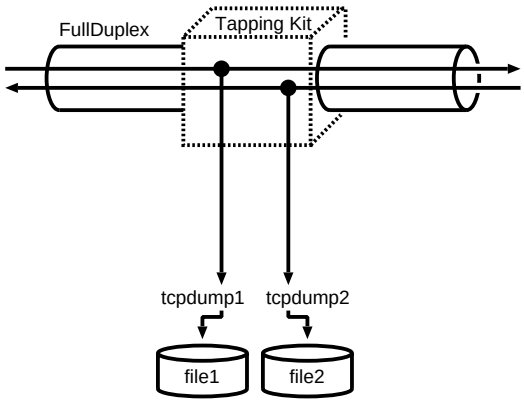

図 4.4. タッピングキットを用いた全二重回線の観測

取り扱いを容易にすることが多い。また FullDuplex の通信回線では、双方向の通信を観測するためタッピングキットによってパケットを分配し、それぞれの方向毎にダンプデータの収集する手法がある (図 4.4)。

これらの環境ではダンプデータが複数のファイルに分割されることになるため、ダンプデータの取り扱いが煩雑になる恐れがある。このため複数のダンプデータを結合するツールがいくつか開発されている。しかし tcpslice は一つ目のファイルの最終パケットの到着時刻以降に、二つ目のファイルの先頭パケットの到着時刻が観測されていなければ結合を行うことができない。また pcapmerge はメモリ資源を多く消費し動作速度も遅い。

4.3.3 tcpdmerge の開発

tcpdmerge は 2 つのダンプデータを、個々のパケットのタイムスタンプを見ながらソートし、マージした結果を 1 つのファイルに保存することができる。開発は C 言語を用い、Solaris と FreeBSD 上で動作を確認している。

以下に具体例によってその動作を示す。file1.log と file2.log に次のようなパケットが含まれていたとする。

```
prompt% tcpdump -nttr file1.log
977231921.235979 192.169.1.1.1023 > 192.168.2.2.54396: udp 92 (DF)
977231921.238222 192.169.1.1.1023 > 192.168.2.2.54396: udp 96 (DF)
977231921.240246 192.169.1.1.1023 > 192.168.2.2.54396: udp 88 (DF)

prompt% tcpdump -nttr file2.log
977231921.232584 192.168.2.2.54396 > 192.169.1.1.1023: udp 68 (DF)
977231921.236564 192.168.2.2.54396 > 192.169.1.1.1023: udp 80 (DF)
977231921.238585 192.168.2.2.54396 > 192.169.1.1.1023: udp 80 (DF)
```

tcpdmerge はパケットのタイムスタンプに基づき、2 つのダンプファイルを以下のように 1 つにする。

```
prompt% tcpdmerge -w join.log file1.log file2.log
prompt% tcpdump -nttr join.log
977231921.232584 192.168.2.2.54396 > 192.169.1.1.1023: udp 68 (DF)
977231921.235979 192.169.1.1.1023 > 192.168.2.2.54396: udp 92 (DF)
977231921.236564 192.168.2.2.54396 > 192.169.1.1.1023: udp 80 (DF)
977231921.238222 192.169.1.1.1023 > 192.168.2.2.54396: udp 96 (DF)
977231921.238585 192.168.2.2.54396 > 192.169.1.1.1023: udp 80 (DF)
977231921.240246 192.169.1.1.1023 > 192.168.2.2.54396: udp 88 (DF)
```

4.3.4 評価

tcpdmerge と pcapmerge の処理速度を比較した。使用したデータは、ある 100BASE-TX の回線をタッピングキットを用いて収集したダンプデータである。表 4.1 は 2 つのファイルのサイズを示している。

表 4.1.　データのサイズ

file name	size [byte]
dump_in.log	71,648,148
dump_out.log	12,553,036

Sun 社製 Ultra2 上でそれぞれのプログラムを実行し、time コマンドで処理速度を測定した。以下に示す結果より、本ツールを用いることにより大幅な処理速度の向上が実現できた。

```
[tcpdmerge]
prompt% time tcpdmerge -w tcpdmerge.log2 dump_in.log dump_out.log
   PID USERNAME THR PRI NICE  SIZE   RES STATE    TIME    CPU COMMAND
 20649 akiraka    1  60    0  976K  648K sleep    0:04  7.28%
3.15u 5.13s 0:34.47 24.0% (約 34 秒)

[pcapmerge]
prompt% time ./pcapmerge -w pcapmerge.log -r dump_in.log -r dump_out.log
   PID USERNAME THR PRI NICE  SIZE   RES STATE    TIME    CPU COMMAND
17295 akiraka    3  20    0  263M  262M cpu/1    5:43 48.29% pcapmerge
3785.11u 16.14s 1:09:05.29 91.7% (約 1 時間 9 分)
```

4.3.5 まとめ

パケットのタイムスタンプに基づき複数のダンプファイルを結合する tcpdmerge を開発した。現在 α バージョンであり、マルチプラットフォーム化やその検証などが十分ではないため、今後も開発を続ける予定である。

4.4 その他のツール

tcpdump によって生成されるダンプファイルは、ダンプデータの共通フォーマットとしての地域を確立しつつある。このため、tcpdump のダンプファイルの加工や情報の抽出を行う様々なツールが開発され、公開されている。表 4.2 は、公開されているツールとその機能の概要を示したものである。これらのツールを組み合わせることにより、効率的にダンプデータを取り扱うことができるようになる。

これらのツールの取り扱いについては、以下の書籍、URL で解説が行われている。

- UNIX MAGAZINE 1999 年 4 月号
  「UNIX の知恵袋」
- UNIX MAGAZINE 2000 年 5 月号
  「UNIX の知恵袋」
- TCPDUMP Tip,s 集
  (http://www.nemoto.ecei.tohoku.ac.jp/~akiraka/tcpdump/)
- Manpage of TCPDUMP (日本語)
  (http://www.linux.or.jp/JM/html/tcpdump/man1/tcpdump.1.html)
- libpcap easy tutorial (Japanese)
  (http://www.goto.info.waseda.ac.jp/~kato/pcap-eztut.html)
- libpcap Japanese Manual
  (http://www.goto.info.waseda.ac.jp/~kato/pcap.0)

最後に tcpdump に関係が深いサイトの一覧を示す。

表 4.2.　tcpdump に関連の深いツール

ツール名	機能
ethereal	パケットのデコード、ストリームの再構築
ngrep	ダンプファイルに対する文字列マッチング
editcap	ダンプファイルのデータフォーマット変換
tcpdstat	パケット数、パケットサイズなど統計情報の収集
tcptrace	パケットフロー単位の統計情報の収集
tcpslice	時間指定によるダンプファイルの切り出し
tcpdpriv	プライバシー情報の削除
tcpdpriv type R	プライバシー情報の削除＋変換表の保存
tcpdmerge	タイムスタンプに基づくダンプファイルのマージ

- LBNL's Network Research Group (http://ee.lbl.gov/)
- tcpdump.org (http://www.tcpdump.org/)
- The Internet Traffic Archive (http://ee.lbl.gov/)
- MAWI Working Group (http://www.wide.ad.jp/wg/mawi/index-j.html)

4.5 まとめ

ネットワークを流れるパケットそのものを収集・解析するツールとしてtcpdumpを取り上げ、tcpdumpに関連の深い一連のツールを概観した。これらのツールを活用した効果的な管理・研究が期待されると共に、さらに多くのツールが開発されていくと思われる。