システムソフトウェアと
オペレーティング・システム 90－8
（２００２．６．２７）

プログラマブルなパケット解析システムの提案

菅原 俊治、明石 修、廣津 登志夫、佐藤 孝治
NTT 未来ねっと研究所
{sugawara, akashi, hirotsu, koji}@t.ecl.net

概 要

収集したパケットの解析手続きをルール形式で記述し、それに基づいて実時間でパケット解析を行うシステム（Programmable Packet Analyzer Package, PPAP）を提案する。近年の急速なインターネットの拡大・普及に伴い、各種のパケットベースの表示・解析・監視・管理システム要求され、またそれに応える形で個別に構築されてきた。これらは、それぞれの役割にあった機能を持つが、基本的な部分はパケット単位の解析という共通的な仕組みである。PPAP では、これらのパケット照合や解析をルール形式で個別に記述し、それから求めるネットワークモニターを生成するシステムである。本論文では、PPAP の研究の立場と目的、機能概要、およびいくつかの例題を示す。

A Programmable Packet Analyzer Package

Toshiharu Sugawara, Osamu Akashi, Toshio Hirotsu and Koji Sato
NTT Network Innovation Laboratories

Abstract

This paper proposes a programmable packet analyzer package that generates the realtime packet analyzers based on the rule-based analysis procedures. As recent growth of the Internet enables people to use it for the wide range of activities such as business, education and research/development, it is required to maintain/monitor the networks and the nodes (hosts/servers/routers) as well as to protect over-eager accesses, attackers, intruders and various computer virus. Responding to these requirements, a number of tools such as network and host monitors, network profilers and intruder detection systems have been proposed and used. These systems usually share a packet-based analysis component whose actions are induced by captured packets. The proposed package provides the rule-like descriptions of packet-based analysis procedures and generates the desired network analyzers/monitors from the corresponding set of rules.

1 はじめに

収集したパケットの解析手続きをルール形式で記述し、それに基づいて実時間でパケット解析を行うシステム（Programmable Packet Analyzer Package — PPAP）を提案する。パケットをタップもしくは受取り、それを解析するシステムとして、パケット表示（たとえば、tcpdmp[13]）、進入やアタック検出システム（Intruder Detection System、IDS。たとえば snort[11]）, ネットワークの障害発見シス

テム（たとえば LODES[15, 16], ENCORE[3, 14], ExpertSniffer[2, 1], netsaint[10]）、サーバー類の稼動状況・パーフォーマンス測定および network profiler/grapher（netsaint, ping/arping, mrtg[9], aguri[5] など）がある。これらは、基本的には、一部あるいはすべてのパケットを収集し、必要な解析（パターンマッチングやデータ量の測定など）を行うシステム（を含んでいる）である。しかし、それぞれのシステムで、同じようなパケットの照合と解析、付随する制御が個々独立に書かれているという問題がある。

本研究では、パケット解析手続きを統一的な方法で記述し、それらを変換（translate + compile）することで必要とするパケット解析システム（装置）を生成できるような、包括的なシステムを構築するのが目的である。具体的には、パケットの収集と簡単なプロトコルチェックを行う基本部分と、解析方法を定義するためのルール（ルールセット）から構成され、必要に応じて書かれたルールと基本部分を合わせてシステムを生成することができる。本資料では、本研究の基本的な考え方、システム構成、記述例等を示し、最後に今後の研究課題を示す。

2 システムの概要

2.1 動機と目的

我々は以前、TCP/IP/Ethernet ベースのネットワークにおいて、ネットワークの各セグメント（ここでは IP ルータとブリッジを境界とする範囲をセグメントと呼ぶ）にエージェントを配置し、

(1) ネットワークに流れているパケットをタップし、それを解析し問題のありそうなパケットを検出する。

(2) 検出と同時にその（診断やマネージャに提示するための）パケットの前後数パケットの情報をファイルに出力する。

(3) (1) による問題発見と (2) のパケット情報、または、ネットワークマネジャーからの障害情報に基づいて単独もしくは他のセグメントのエージェント協調して障害の原因を究明する。

など機能を持つシステム LODES（Large internet observation and diagnostic Expert System）[15, 16] を提案した。本システムの各エージェントは、パケットを収集解析するプロセス（NOBS — network observer component）、テストパケットの送出やシュミレーションパケットを出すプロセス、診断知識を持ち障害の原因を究明するプロセス、および他のセグメントにあるエージェントと診断中に非同期に通信するためのプロセスの４つで構成されていた。このうち第一のプロセス NOBS は、NIT（network tap protocol）[1]を利用してすべてのパケットを実時間で解析し、障害や問題のありそうなパケットを発見する知識がプログラムに（当時の計算機の性能で十分な効率を得るためにかなり深く）組み込まれていた。

また、ネットワークの普及、技術革新は、ネットワークの監視・解析項目を変えてきた。10-15 年前は、多くの場合、インターネット関連のソフトウエアの実装の問題（あるいはプロトコル解釈の不備）、一般利用者からネットワーク管理者まで含めて関連知識の不足などが原因となることが多かった。そのため、ブロードキャストストームや acking ack、silly-window-syndrome など基本的な要因による障害が占めていた [4]。しかし、近年のインターネット利用者の拡大により、監視すべき項目が多様化してきた。たとえば、商用化にに伴う各種サーバの管理やスループットの把握、一部利用者によるアタック（たとえば [6]）やネットワークへの侵入に対する監視、QoS が比較的厳しい新たなアプリケーション（動画配信、VoIP など）に伴うトラフィック管理（プロファイラ）やボトルネック解析、またマルチホーム化やマルチターゲット（情報のミラー、分散された計算機資源の活用（[8] など））に伴い client 側からのルート選択（アプリケーションレベルルーティング）やスループット予測に関連するデータの収集・解析などがある。

本発表で提案する PPAP は、LODES を作成した経験と近年の計算機の高速化や上記のようなパケット解析に関する多様な要求を考慮して、(1) 解析や診断の知識・ノウハウ・ルール・プログラムを独立させ、(2) 個別にかつ簡単に変更・追加を可能とし、(3) 各種の目的に合わせたパケット照合・解析システムを生成することを目的としている。

[1]当時は libpcap はなかった。その後 libpcap を利用したバージョンもある。NIT には、自分が出したパケットは収集できないという欠点があった。また、当時は switching hub もなく、いわゆる黄色い Ethernet ケーブルとマルチポートトランシーバを使ったもので、特定のホストですべてのパケットを観測できた。

2.2 システムへの基本的な要求条件

本論文で提案するシステムを実現するために、必要な基本的機能について述べる。特にインターネットでは、パケットのドロップ・ロスト、パケットの重複などが発生し、それらを回避あるいは取り込んだシステムの動作を記述できる必要がある。

パケット照合による起動： パケットの照合は、前に述べたシステムのほか、NAT や firewall などを実現するための共通機能である。実際、パケットが原則的には唯一の入力であり、その入力をトリガーに動作を起こすシステムであるといえる。

パケット解析項目の動的な設定、追加： ここでは、特定のイベントが発生したときなどに、あるルールをアクティブにして、一定期間にモニタリングやパケットに対する操作を有効にすることを意味する。本機能の目的は、照合の効率化と、予め照合項目を設定できないという二点である。これは、可能性のある事象に関するルールを当初から動作させるのでなく、必要に応じて、ルールをアクティブにしていく必要性があると考えるからである。特に後者の目的については、たとえば、流れてしまった要注意パケットに対する他のホストの反応を見るために、発見された要注意パケットの情報に基づいて、新しいルール（監視）を有効化することや、ある外部ホストからポートスキャンが行われたときに、その発信元の情報に基づいてフィルタをかけるなどの例がある。

反応させるか否かの判定記述： 計算機などは、一定間隔で、あるいは特別なプログラムが動作したときに、あるパケットを出し、それが監視対象のパケットであることがある。仮に、この事象を単に数え上げたいのであれば問題はないが、その発生の存在を知りたいときだけの場合には同一アクションの頻繁な繰り返しは無駄であるだけでなく、管理者にも膨大なデータを提示することになる。例としては、ネットワーク監視において、あるホストが定期的にそのサイトでは認められていないパケットやイリーガルなパケットを出しているが、それらは外部には出ないか特に反応するホストが無いなどで、差し当たりは無視できる（緊急でない）事象がある [12]。なお、実現方法として、真に無視する場合（照合しない）と照合してもイベントを発生させない場合とがある。

時間記述性（時間制限付モニタ）： 上記の 2 項目とも関係するが、ネットワークのモニタには、時間記述性が重要である。たとえば、あるイベントの直後だけ短期的に有効にしたい監視や、逆に、間欠的な事象（間欠事象の例としては、マルチキャスト、ブロードキャストの同時性問題 [7] など）や定期的なアタック（定期的に ping/traceroute をかけてくることも含む）の検出のために、比較的長期に渡って観測し、間欠性に対処したり、「定期的」であることを検出することが必要である。また、実験や試験的動作確認などで一定期間のフィルターの on/off をしたいこともある（この場合、ついついもとにもどすことを忘れがちである）。

（時限付き）ステートの保持： プロトコルの把握には、多かれ少なかれ、状態遷移の記述をサポートしなければならない。これにより、パケットの重複などに惑わされなくなる。一方、インターネットにはパケットの消失もかなりある。したがって、適当なタイマーをもとに、ステートの初期化が必要なこともある（必要であれば、ステートをシミュレートしているホストにリセットなどを送る）。

本システムでは、パケットの解析の記述方法としてルールを採用しており、上記のコントロールはルールで記述できなくてはならない。

2.3 パケット解析システム作成の流れ

図 1 に本研究で想定するパケット解析システム作成の流れを示す。基本モジュールは、パケットを収集し、プロトコルタイプの同定、プロトコル境界のマーキングなどを行い、ユーザが定義したプロトコル解析用のルールから容易にアクセス可能なように準備をする。ルールは、基本的な部分は、C++ 風の言語で記述し、制御を宣言的に表す部分と解析手続きを記述する部分とに分かれる。このルールは専用トランスレータによって、C++のプログラムに変換され、同時に使われているライブラ

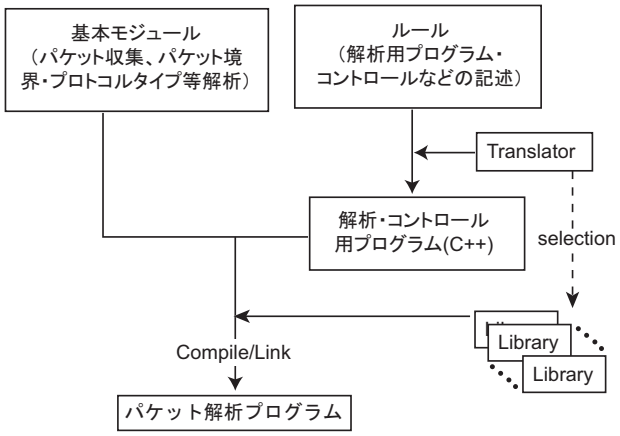

図 1: PPAP の基本構成

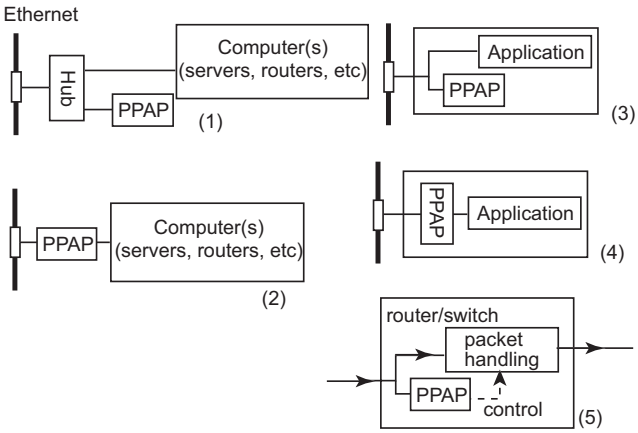

図 2: PPAP の設置例

リを選択する。最終的に、これらは基本モジュールといっしょにコンパイルされ、オブジェクトが生成される。なお、本システムの利用者（つまりルールを記述し、解析プログラムを生成、実行させるユーザ）は、ネットワーク管理者、システム管理者を想定する。

3 各モジュールの構成

3.1 基本モジュールの機能（概要）

本システムは、図 2(1)(2)(3) のような使用形態を想定している。PPAP で生成したシステムは、独立した専用のマシンかもしくはある計算機内の 1 プロセスとして動作する。どの場合にも、複数の計算機、もしくはその先のすべてのネットワークのパケットを対象に解析ができるが、(3) の場合には、特定のホストもしくはアプリケーションの連携も想定している。なお、ネットワークは、fast Ethernet（100Mbps、最大毎秒数千フレーム程度）を考える。

基本モジュールは、パケットを収集すると同時に、以下に述べるルールの実行や各種プロトコルヘッダのフィールドへのアクセスのための前処理を行うコンポーネントと、実際にルールの実行制御を行うコンポーネントがある。前処理は、(a) 各ヘッダのプロトコルの同定、(b) ヘッダ間やペイロードの境界を解析し、ルールによるアクセスの準備を行う。ルールの実行制御は、ルールのスケジューリング、以下に述べるルールの各種状態管理、終了・例外処理等が含まれる。それぞれの詳細については、以下の節で述べる。

3.2 ルールとルールセット

ルールは、基本的には、premise の部分（おもにパケットの照合部分）、action 部分（パケットが照合した場合に、そのパケットをもとに起こす手続きの記述）、control 部分（主に、実行順序やルールの時間制御と例外処理の宣言などを行う）で構成される（図 3 参照）。各ルールには引数を与えることができ、その引数を用いた照合を行うなど、ルールの実行内容を変更することができる。

```
defruleset <ruleset_name> (arg1, ... ) {
   <共有変数の定義; >   // ルールセット全体の共有変数
   control {   ルールセット全体の共通制御情報   }
   // Declare rules and exception handlers in this ruleset.
   defrule <rule_name_1> (arg1, …);
   defrule <rule_name_2> (arg1, …);
     ……
   defexhandler <handler_name_1>
     ……
}
defrule <rule_name_1> (arg1, ...) : <ruleset_name>  {
    <局所変数の宣言;>
    control { 本ルールの制御情報(全体より優先) }
    premise { 照合条件 }
    action { 実行プログラム }
}
defrule <rule_name_2> (arg1, ...) : <ruleset_name> {
    上記と同様 }
     ……
defexhandler <handler_name_1> : <ruleset_name> {
    <実行プログラム> }
    ……
```

図 3: ルールとルールセットの記述

表 1: 各種ルールの状態の説明

状態名	状態の説明
Inactive	ルールは有効でない状態
Active	ルールが有効にある状態。各パケットの入力に対しルールの照合が行われる。
Sleep	ルールが他のルールを active にしたときに、指定があれば、その終了を待つ状態。
Freeze	ルールを active にできない状態。ある一定時間後に inactive の状態になる。

ルールセットは、いくつかのルールの概念的まとまりであると同時に、各種相互制御が有効になる（したがって影響を受けるかも知れない）範囲を定義する。たとえば、(1) 同一ルールセット内で、あるルールから他のルールをアクティブにし、それらの相対的な実行順序を決定する、(2) 他のルールの終了を待つ、(3) ルールセット内の残りのルールについて現在のパケットの照合をスキップさせる（つまりパケット単位の cut operator 的動作）ことができる。なお、PPAP には、ルールセットの単位で include される。

3.2.1 ルールの状態

ルールの状態として、inactive, sleep, active, freeze の 4 つを用意した。各種ルールの状態を表 1 に示す。ここで、freeze 状態は、同じアクションを頻繁に起こさないように一定期間無効にことを想定して導入した。

3.2.2 Premise と Action パート

Premise は、パケットの各種フィールドの照合、ルールかルールセットで定義した変数の照合の記述に限らる。この条件に照合したときのみ、action 部分が実行される。Action 部分では、パケットの情報と以下に示すライブラリを利用して、必要な情報の収集・検証・解析・記録、警告や（パケットや記録）内容を提示するための処理、パケットの書き換えや転送、他のルールの有効化、PPAP で生成された他のシステム間の通信（3.6 節参照）などが記述される。両パートとも、基本的には C（実際には C++）のシンタックスに従い、実際、それぞれ C++の method としてトランスレータにより展開される。

表 2: ルール終了条件（terminate-type）の指定

Duration: 時間一定時間でルールを終了（単位：秒）
Firecount: 一定回数ルールが照合されたら終了（単位：回）この指定だけでは終了しないこともあるので、他の terminate type との併用を行うか、利用者の判断で単独使用を行う。。
Packetcount: 一定数のパケットが収集されたら終了（単位：パケット）
Inactivetime: 一定時間ルールが照合させなければ終了（単位：秒）
Inactive-packetcount: 一定数パケットが収集され、かつその間、ルールの照合が無かったら終了（単位：パケット）

3.2.3 ルールの有効化と実行順序

ルールの action 部分で、同一ルールセット内の新しいルールや他のルールセットを active にする（ルールセットが active になったときには、top ルールと呼ばれるいくつかのルールが active になる）。その際に、必要な情報を有効化するルールに引数として値を渡すことができる。ルールの実行順序は、priority（整数）で制御される。ルールの control 部分で、priority を定義できるが、特に指定しなければ、それをアクティブにしたルールより一つ高い priority となり、実行が優先される。

3.2.4 明示的な終了と割込み的終了

ルールの終了は、(1) 自ら action の一部として終了するか、それを呼び出したルールから終了させる**明示的な終了**と、(2) control 部分の宣言にしたがって、特定の条件になったときに終了させる**暗黙的な終了**とがある。後者の条件を表 2 に示す。暗黙的な終了は、プロトコルで定められたタイムアウト処理、2.2 節で述べた時間記述性やパケットのロストによる停止状態の回避を行うため、action 部分だけでは対処できない制御を記述するものである。各終了時には必要であれば exception handler を呼

び出すことができる（たとえば、終了したときに、マークをつけたパケットを管理者にメールで送るなど）。Control 部分の記述方法を図 4 に示す。

```
control {
   <terminate type>= <integer> exhandler <exhander name>;
   toprules=<top rulename1>(arg1,…), <top rulename2>,…;
   priority = <integer> or +=<integer>, or -=<integer> ;
    // toprulesはルールセット内でのみ有効
}
```

図 4: **Control** 部分の記述

3.3 プロトコルヘッダ形式の宣言とアクセス方法および追加

現在の基本モジュールでは、ethernet（IEEE 802.3 関連）、PPP、UDP and TCP/IPv4 関連のプロトコル形式とその解釈を実装している。これらより （１つ）上位のプロトコルは、それらのプロトコルヘッダを図 5 のように宣言することでルールからアクセスできる（２つ以上上位はいまのところ実装していない）。

```
protoprcol_header_structure <name> {
 <length in bit>: <accessor1>,…<accessorN>;
      ……}
// <name>はaccess methodとして、各ヘッダフィールドへの
// アクセスに使われる.

protocol_discrimination <name> { 判別式関数 }

//example (DNS protocol header)
protocol_header_structure DNS {
  16: identifiction, id, ident;
  16: parameter, param;
  16: Number-of-Query, NoQ;
      …… }
protocol_descrimation DNS {TCP(port) == 53}
protocol_descrimation DNS {UDP(port) == 53}

// access method examples
DNS(identification) and DNS(id) refer to the identification field.
DNS(NoQ) refers to the number-of-queries field.
```

図 5: プロトコルヘッダ形式の宣言と access method (accessor)

3.4 トランスレータ

トランスレータは、記述されたルール（セット）と追加したプロトコルヘッダの形式を読み込み、必要な C++のコードを生成する。また、参照されているライブラリを解析し、それらと基本モジュールを合わせて C++コンパイラを起動させる。

3.5 ライブラリとトランスレータ用マクロ

PPAP で用意されているライブラリ関数（methods）は、主に premise の部分で使う各種プロトコルヘッダの各フィールドへの access methods（以下 accessors と呼ぶ）と照合のための比較判定関数、主に action 部分から呼び出す各種 C（C++）関数群である。主なものを図 6 に示す。Premise でも action 部でも、利用者が定義した関数を利用することはできる。ただし、その関数の中でルール内の変数は参照できない。トランスレータを通せば照合したパケットの各フィールドへのアクセスはできる。[2]

Accessors や比較判定関数のために、各フィールドのサイズに合わせて、データタイプ u_int32（32b フィールド）、 u_int16（16b フィールド）、u_int8（8b フィールド）、が用意されている。これらは、プロトコルヘッダ形式の宣言から、トランスレータにより適当にシフトとマスクがかけられた値として取り出される。また、利用者の可読性から、いくつかのトランスレータ用マクロが用意されている。たとえば、accessor IP(src) は、パケットの IP ヘッダにある source address field の値にアクセスするためのもので、型は u_int32 である。しかし、たとえば、

```
IP(src) == INET_ADDR("A.B.C.D")
IP(src) == INET_ADDR("foo.XXX.co.jp")
```

というようなマクロを許すことにより、定数であれば可能な限りトランスレートするときに 32 ビット形式の値に展開する。[3]この他に、udp/tcp のポート番号、ether ヘッダのタイプフィールド、IP ヘッダのプロトコルフィールドなどにも対応したマクロが用意されている。

3.6 パケット照合に関する留意点

PPAP は、パケットベースの解析システムの作成を目指している。パケットの入力をトリガにルールが照合・処理されるが、これに関連して留意事項がある。第一はパケット分割の問題である。IP

[2]が、読みやすさの点からお勧めではない。いつの時点でのパケットになるか注意が必要。引数として渡すべきである。

[3]もし、マクロの引数が変数であったり、実行時に展開するのであれば、単に

```
IP(src) == inet_addr("A.B.C.D")
```

とすればよい。

ETHER(target), **IP**(source), **IP**(ttl), **TCP**(ack), **UDP**(sport)など：各プロトコルヘッダのフィールドへのaccessors。

bool **eq8**(u_int8, u_int8), **eq8**(u_int8*, u_int8)等：ヘッダの8bitsフィールドの比較。他に、**eq4**,**eq16**,**eq32**,などがある。主にPremise部分で使用する。

int **mark_packet**(); 現在解析しているパケットにマークをつける（後で,データの変更や印刷用などに保持している）。

RuleSet ***ACTIVATE_RULE**(<rule_name>(arg1, …)): ルールをactiveにして、そのidを返す。

wait_rule(RuleSet *), **wait_rules**(): Activeにした特定のルール、またはすべてのルールがinactiveになるまでsleep状態に入る。

set_pfield(accessor(field,markid), u_int8*): マークをつけたパケットの指定したフィールドの値を変更する。Accessorは上記のETHER,IPなど。

図 6: ライブラリ関数の一例

fragmented パケットは、最初の部分に上位の UDP, TCP のヘッダが格納されるので、ヘッダの解析には 2 番目以降のパケットはスキップされる。

第二に、TCP のようなストリームベースのプロトコルでは、その上位プロトコルのヘッダが分断されたり、逆に一つのパケット内で、上位プロトコルのペイロードの後に次のヘッダが来ることもある。これらは事実上あまりないこと（特に後者）と無視できる場合もあるが、将来 TCP のデータを蓄え、再構築するライブラリを用意することも考えている。

第三に、パケットごとに処理を実行するため、特に大量のパケットを短時間に処理しなくてはならないときには、その実行時間に制約がある。時間のかかる処理を含む場合には、別 thread として動作させ、それに関連するデータ（のポインタ）を渡すという方法を採用する。たとえば、PPAP で生成された別のシステムとの通信、ルータや他の管理システムとの通信、上記の TCP データを再構築するプロセスなどがこれにあたる。PPAP 側の action 部分が長時間停止したり、プロセスを拘束しないように留意する必要がある。

4 例題

DNS ホストの管理

図 7 は、DNS のアクティビティを調べるプログラムで、ping ではなく、query に対する応答の有無を監視している（ping に応答しても DNS が応答しているとは限らない）。なお、DNS の query が、どこかで重複されて送られてきても、ルールの動作には影響しない。

```
defruleset dns_watcher () {
  int dns_state = 0; //共有変数
  control {  // 共通制御情報の設定
  toprules= dns_watcher();
  //このルールセットはudp/ip/*のパケットに有効
  protocol_sequence= (* ip udp ); }
  // ルールなどの宣言
  defrule dns_watcher ();
  defrule dns_resp(u_int32 ipdst,u_int16 dnsid,u_int16 dport);
  defexhander when_timeout;
}
defrule dns_watcher () : dns_watcher {
  // dns queryがあった。
  premise {  ETHER(type)==IP && IP(dest)=="A.B.C.D"
      && UDP(dport) == DNS && DNS(NoQ) != 0  }
  action {// dns_respをactiveに。自己はsleep。
    ACTIVATE_RULE(dns_resp(IP(src),DNS(id),UDP(sport)));
    wait_rules();  }
}
#define RESPOND 0
defrule dns_resp(u_int32 ipdst,u_int16 dnsid,u_int16 dport)
      : dns_watcher {
  control {
    priority += 1;  // priorityを上げる（必要ない）。
    duration = 5 exhandler when_timeout; } // 5秒間有効。
  premise { IP(src)=="A.B.C.D" && IP(dest)==ipdst
     && UDP(sport)==domain && UDP(dport)==dport
     && DNS(id)==dnsid   }
  action {
    dns_state = RESPOND;  // とにかく応答があった。
    INACTIVATE(this_rule); }  //inactive this rule
}
defexhander when_timeout : dns_watcher {
  //応答が5秒間ないとき
  if (++dns_state > 2) {
      ..... // ここに警告を出す関数を挿入
      dns_state = -100;
      //この状態が続いてもしばらく警告を出さない
    } }
```

図 7: 適用例（DNS サーバの動作監視）

Intruder Detection

IDS は基本的には特定のパケットの検出である。これは、アクセスリストをルールとして記述するればよい。また、ポートスキャンなどは、ある特定のホストから頻繁なアクセスがあったときに、新規にそのホストからのパケットのみ取り出すルールを記述し、その action 部分に頻度や到着パケット履歴、パケットの内容の解析をするルールを active にすることで実現できる。

各コネクションのスループット測定

TCP のスループットをコネクションごとに測定するには、まず SYN のついた TCP パケットを見つけ出し、それから新たに source/destination の IP アドレスとポート番号の４つ組を照合するルールを active にする（照合するルールを変えることで状態遷移に追随する）。その action 部分でスループットを測定する。一方、パケットのロストなどが発生した場合は、ルール終了条件 inactivetime を設定すれば、timeout により初期化される。

NAT/NAPT

TCP については上記と同様に、転送先の IP アドレスとポート番号を管理する。さらに UDP や ICMP の場合には時限付きの管理をすることで、その扱いが可能である。（ここで生成したシステムは、対応する通信のテーブルを管理しているだけで、実際にパケットを書き換え転送するプログラムとこのテーブルを共有していると想定する）。たとえば、DNS の query を外部に出したときに、その発信元の対応ルール active にして、適当な duration を設定すればよい。同様に、ICMP echo や traceroute で発生する ICMP time exceeded なども仲介できる。

これら他に、ARP request/reply から MAC 番号と IP アドレスの対応を作る arpwatch[13] やホストのネットワークカードをチェックする arping、さらには、動的にかつ時限付きで firewall のフィルタを設定あるいは解除することなどが単純な例題として挙げられる。

5 まとめ

パケットを収集し、ルール形式で記述された解析手続きに基づいて実時間でパケット解析を行うシステム PPAP について述べた。今後は、本システムの評価と機能の充実化を図りたい。また、軽量化してカーネル内のネットワーク制御との連携や、ルータやスイッチ内で動作、連携できるようなプロセッサと合わせる（図 2 （4）,（5））ことも検討予定である。ここで提案した記述方式は、一般的なものであり、たとえば、すでにある firewall や IDS などのアクセスリストの記述を PPAP 変換するツールや、ルーターと通信しながら、PPAP による監視とルータとを連動させることも検討したい。

参考文献

[1] http://download.nai.com/products/media/sniffer/support/snd/newfea55.pdf.

[2] http://www.sniffer.com/.

[3] O. Akashi, T. Sugawara, K. Murakami, M. Maruyama, and N. Takahashi. Inter-autonomous-system diagnosis using cooperative reflector agents. In *Proceedings of IEEE International Conference on Intelligent Processing Systems (ICIPS98)*, 1998.

[4] L. Bosack and C. Hedrick. Problems in large LANs. *IEEE Network magazine*, 2(1):49–56, 1988.

[5] K. Cho, R. Kaizaki, and A. Kato. Aguri: An aggregation-based traffic profiler. In *Proc. of QofIS2001*, Sept. 2001.

[6] Computer Emergency Response Team. *IP Denial-of-Service Attacks (CERT Advisory CA-1997-28)*, Dec 1997.

[7] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Transaction on Networking*, 2(2), 1994.

[8] I. Foster and C. Kesselman eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[9] http://www.mrtg.org/.

[10] http://www.netsaint.org/.

[11] http://www.snort.org/.

[12] T. Sugawara and K. Murakami. A multiagent diagnostic system for internetwork problems. In *Proc. of INET'92*, Sep 1992.

[13] http://www-nrg.ee.lbl.gov/nrg.html.

[14] 明石, 菅原, 村上, 丸山, 高橋. マルチエージェントを用いた自律組織間診断システム：ENCORE. 情報処理学会論文誌, 40(6):2659 – 2668, 1999.

[15] 菅原. LAN の診断エキスパートシステムについて. プログラミングシンポジウム（コンピュータネットワークのヒューマンウエアシンポジウム）報告集, pp. 5–11, 1989.

[16] 菅原. 大規模インターネットワーク診断／監視エキスパートシステムについて. 信学論 D-I, J73-D-1(12):990–996, 1990.