OSSのパケット解析ライブラリを使用し，プログラミング言語からパケット解析を行う手法の提案

城倉 弘樹 *,a)　金井 敦 †,b)

概要：これまでのパケット解析は Wireshark などのパケット解析アプリケーションを利用した解析が定石である. しかし, このようなパケット解析では柔軟な処理などに対して限界があり, より高度な通信解析を行う場合に不向きである. また, 近年 IoT 社会の実現化などより新たなプロトコルの出現の頻度が増えてきていて, パケット解析環境の高拡張性も必須となりつつある. 本稿ではプログラミング言語からパケット解析を行うことを補助するライブラリを設計実装することにより, これからの環境での新たなパケット解析手法を提案する.

1. はじめに

近年 IoT の実現化が進んでおり, 様々なデバイスが身の回りのいたるところに偏在し, ネットワークに繋がれて通信を行っている. このようなデバイスの開発者はこれらの通信を解析, デバッグや管理をしなければならない. これまでのパケット解析は wireshark や network minor のようなアプリケーションを使用して, 各プロトコルの解析を行うことが一般的であるが,IoT デバイスなどの通信は既存の通信プロトコルを採用していない場合があり, そのような場合に迅速にその通信を解析することが困難である. それだけでなく, 通信をデバッグする場合, ただパケットをキャプチャするだけでなく, プロトコル通りの通信が行われているかを調べたり, ある条件の通信のみログをとったりすることができることが望ましい. つまりパケット解析をよりプログラマブルに行うことができると作業の効率をよくすることができる. このように本稿では以下の 2 つをパケット解析の課題とする.

- 新たなプロトコルに対してすぐに対応できる
- パケット解析をよりプログラマブルに行う

本稿では新たに設計, 実装した拡張可能なパケット解析ライブラリである LibPGEN を使用, 拡張することで, これらの課題を解決し, 効率のよいパケット解析をする手法を提案する.

また, 効率のより開発のために, 本稿で実装する LibPGEN は C++11 を使いオブジェクト思考で設計を行った. パケット解析などをおこなうツールは多数のプロトコルをサポートすることが好ましく, そのような実装を少人数で行うのは困難である. なので LibPGEN がオブジェクと思考を利用して, より簡単に開発に参加しやすいように設計を行った.

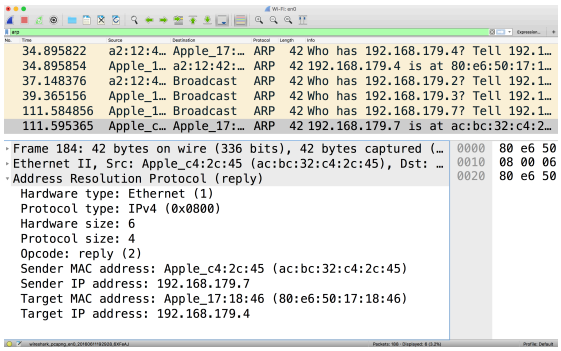

図 1　Wireshark による ARP の通信解析

2. 既存の手法

既存のパケット解析の手法はパケットキャプチャでプロトコルアナライザである Wireshark を例にして進めていく. Wireshark は世界中の開発者が協力して開発を行っている OSS であり, 多数のプロトコルをサポートし, フィルタ機能や統計機能, 可能な範囲で異常検知などを行うことができる強力なパケット解析環境である. Wireshark で arp の通信を解析している図 1 を以下に示す.

また, Wireshark は対応していないプロトコルに関する

*法政大学
†法政大学
a) slank.dev@gmail.com
b) yoikana@hosei.ac.jp

```
pgen::net_stream net("eth0",
        pgen::open_mode::netif);
uint8_t buf[10000];
size_t recvlen = net.recv(buf, sizeof buf);
pgen::udp pack(buf, recvlen);
if (pack.UDP.src==8888 || pack.UDP.dst==8888) {
    printf("%s -> %s \n",
            pack.IP.src.str().c_str(),
            pack.IP.dst.str().c_str());
}
```

図 2　パケット解析のプログラミング例

拡張 (dissector の実装) を簡単に行えるように設計されている.dissector は C/C++や lua で開発をすることができ, プラグインとして Wireshrak の起動時に指定するだけですぐに使用できる.

3. 提案方法に求められる要件

3.1 プログラミング言語から解析を行う

前節より Wireshark は強力なパケット解析アプリケーションであることがわかる. だが,Wireshark では GUI アプリケーションでなので自動化やより柔軟な作業に対して弱い. 例えば,ARP スプーフィングかもしれない通信をキャプチャし, ハイライトするところまでは Wireshark の標準の機能で行うことができるが, それに関与した通信端末のログをとって異常検知のデータとして詳しく調べたり, 一定の処理を自動化させるといったことをする場合, ある程度プログラマブルな環境が必要になり,Wireshark はこのような作業はプラグインを実装して Wireshark にロードさせなくてはいけなく, すぐに処理を行えるわけではないので目的達成のコストが高い. この環境だと, パケット解析だけを行いたいユーザにとっては敷居が高いと言える. 本稿ではパケット解析を一般的なプログラミング言語から行うことによってこのような問題を解決する手法の一つを提案する.

例えば TCP のポートが 8888 番のアプリケーションの通信を行っている端末の IP アドレスを列挙したい時などは Wireshark のディスプレイフィルタのみでは, 実現することができない. この場合,Wireshark は 8888 番の通信をフィルタすることはできるが, その通信を行っている端末の IP アドレスを直接列挙するには別途でプラグインの実装が必要である.

もしプログラミング言語からパケット解析ができるとすると, ユーザは TCP のポート 8888 のパケットをフィルタし, それらのパケットの IP アドレスを printf 関数などで列挙することができる. このような場合でのプログラミング例を図 2 を示す.

このようにアプリケーションを使用して, パケット解析を行うのでなく, プログラミング言語からパケット解析を行うことで, 簡単により柔軟な処理を実現させることができる.

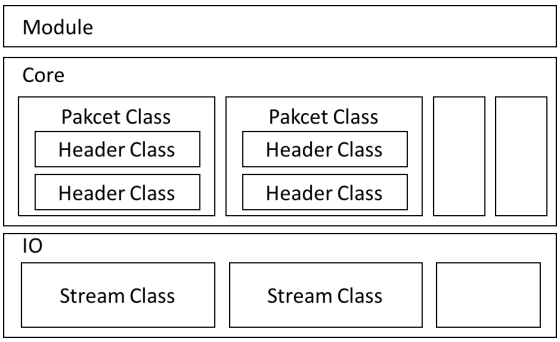

図 3　LibPGEN のアーキテクチャ

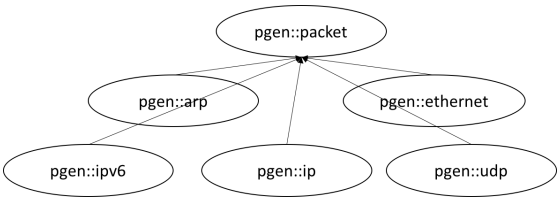

図 4　パケットクラスの継承関係

3.2 新たなプロトコルを迅速に解析する

Wireshark では dissector の実装による拡張で, 新たなプロトコルを解析できるように設計されている. 本稿でもこのような環境に簡単に対応できる環境をパケット解析のライブラリの設計により提案し, 新たなプロトコルに迅速に対応できることを示す.

4. パケット解析ライブラリの設計

Wireshark などを用いた, 既存の手法では柔軟な処理を行うことが難しいと思われる. 本稿ではそのような状況下でパケット解析の機能をプログラミング言語から呼び出してプログラマブルなパケット解析を行うために LibPGEN というパケット解析のライブラリを設計し, 実装した.

LibPGEN は以下の図 3 のように Core, IO, Module の 3 つのコンポーネントに分けて設計を行った. Core はメインの処理であるパケット解析を行うコンポーネントが含まれている.IO はパケットの入出力を行うコンポーネントが含まれていて, 現在はネットワークインターフェース,pcap ファイル,pcapng ファイルにパケットの送受信を行うことがでる.Module は Core,IO を使ったモジュール群で構成されている. 本稿では Core コンポーネントの設計を中心に議論を進めていく.

パケット解析を行う機能はパケットクラスとして設計,

```
#include <pgen2.h>
int main()
{
 pgen::arp pack;
 pack.ETH.src  = "11:22:33:44:55:66";
 pack.ETH.dst  = "ff:ff:ff:ff:ff:ff";
 pack.ETH.type = pgen::ethernet::type::arp;
 pack.ARP.operation =
              pgen::arp::operation::request;
 pack.ARP.hwsrc = pack.ETH.src;
 pack.ARP.qsrc  = "192.168.0.10";
 pack.ARP.hwdst = pack.ETH.dst;
 pack.ARP.qdst  = "192.168.0.1";
 pack.compile();
 pack.hex();

 pgen::net_stream net("eth0",
                  pgen::open_mode::netif);
 net << pack;
}
```

図 5 ARP パケットを解析するコード

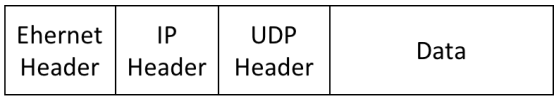

図 6 UDP パケット

実装している. 各プロトコルごとに一つパケットクラスは実装されていている.ARP パケットを解析するパケットクラスは pgen::arp クラスとして実装され,UDP パケットを解析するクラスは pgen::udp クラスとして実装されている. すべてのパケットクラスは抽象クラスである pgen::packet クラスを基底クラスとして継承している. パケットクラスの継承関係は図 4 のようになっている.

4.1 パケット解析をプログラミングする例

パケットクラスは新たにパケットのバイナリを生成, 既存のパケットのバイナリを解析し, 内容を調べたり, 一部, または全てを改変することができる. ARP パケットをネットワークインターフェースに送信するサンプルコードは図 5 のようになる.

図 5 で示したコードは典型的な arp リクエストパケットのバイナリを生成して, そのバイナリの hexdump を表示し, eth0 からパケットを送信するサンプルコードである. このようにパケットクラスは各プロトコルのヘッダフィールドをメンバ変数で保持しており, それを任意に編集することで, 自由にパケットを生成することができる. 各要素は [instance.PROTOCOL.element] で直接指定することができる.

```
class header {
    public:
        virtual void clear() = 0;
        virtual void summary(
                bool moreinfo=false) const = 0;
        virtual void write(void* buffer,
                size_t bufferlen) const = 0;
        virtual void read(const void* buffer,
                size_t bufferlen) = 0;
        virtual size_t length() const = 0;
};
```

図 7 pgen::header

```
class packet {
    protected:
        virtual void init_headers() = 0;
    public:
        virtual void clear() = 0;
};
```

図 8 pgen::packet

表 1 pgen::header の純粋仮想関数の概要

関数名	役割
clear()	ヘッダ値をデフォルト値に戻す
summary	ヘッダの情報を出力
write	メモリにヘッダのバイナリを書き込む
read	メモリからヘッダのバイナリを読み込む
length	ヘッダの長さを返す

4.2 パケットクラスの実装

パケットクラスはメンバ変数として, ヘッダクラスを保持している (packet-class has-a header-class の関係) 一般的な UDP パケットは図 6 ような構造をしている

一般的な UDP パケットは ethernet, ip, udp のヘッダ, 通信内容である UDP のデータで構成されている. ヘッダクラスはこの ethernet ヘッダや udp ヘッダを表現している. ヘッダクラスも各プロトコルごとに実装されていおり, tcp ヘッダを表現するクラスは pgen::tcp クラスとなる. 今回の pgen::udp パケットクラスは pgen::ethernet_header, pgen::ipv4_header, pgen::udp_header クラスを保持している. パケットクラスと同様にすべてのヘッダクラスは pgen::header クラスを基底に継承をして実装をしている.

4.3 効率的な設計による作業の効率化

LibPGEN はオブジェクト思考プログラミング (OOP) でして設計開発を行った. OOP では抽象クラスを用いることによって, 開発者をある程度束縛することによって, 開発作業の簡素化などを行うことができる. 本稿で実装した LibPGEN も OOP で開発をおこない, 拡張を容易に行える

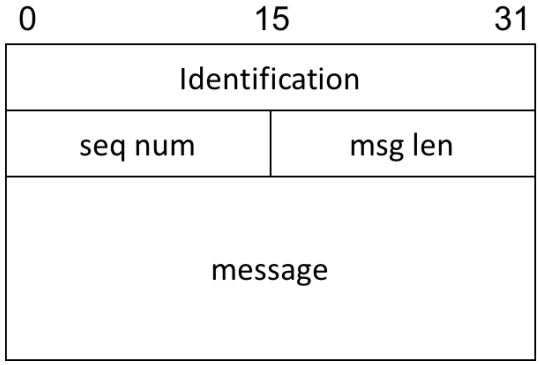

図 9 TMP Header Format

ように設計をした.

例えば, パケットクラスやヘッダクラスでは抽象クラスとして,pgen::packet クラスや, pgen::header クラスを設計してある. これらのクラスには幾つかの処理に対応した純粋仮想関数が定義されており, 拡張するユーザはこの純粋仮想関数を実装することに開発のタスクを分けることができるので, より簡単にすることができる. 図 7,8 は pgen::header と pgen::packet クラスの純粋仮想関数のプロトタイプの一部を示している.

新たにヘッダクラスを実装したい場合, 開発者は pgen::header クラスを継承したクラスで clear 関数,summary 関数,write 関数,read 関数,length 関数を実装すればいい. 各関数の役割を以下に表 1 で示す.

もしユーザが IPv6 の UDP パケットを解析したい場合, ethernet, ipv6, udp のヘッダクラスを保持するパケットクラスを新たに定義してそのクラスのメンバとして ethernet, ipv6, udp のパケットを保持させることで柔軟なパケット解析を既存の資源を再利用することで再現することができる。

新たに DHCP パケットの解析を行うパケットクラスを実装したい場合,pgen::header クラスを継承した dhcp_header クラスを実装し,pgen::packet クラスを継承した dhcp クラスを実装すればよい. 実装するパケットクラスでは DHCP パケットとして必要な ethernet,ip,udp のヘッダを実装した pgen::ethernet_header,pgen::ip_header,pgen::udp_header クラス, そして, 新たに実装した hdcp_header クラスを呼び出すことによって dhcp パケットを扱うクラスを実装することができる.

4.4 新たなプロトコルを解析する例

前節で課題をあげた通り,LibPGEN は新たなプロトコルに関する拡張が容易に行えるように設計してある. 例として本稿では実在しない簡易メッセージ通信用のプロトコルである TMP(TestMessageProtocol) を用いて, その通信を LibPGEN で解析できるように拡張することによって, かんたんに新たなプロトコルの解析が行えることを示す.TMP のヘッダフォーマットを以下に示す. TMP プロトコルは udp での通信で簡易的なメッセージ通信を行うプロトコルで udp8888 で動作する簡易メッセージ用プロトコルである.

図 9 の sequence number はパケットのシーケンス番号を示し,id は通信端末の id を表しているものとする. msg-len はその後に続くメッセージの長さをバイト単位で示しており,message は通信したいメッセージが格納されている.

LibPGEN を使用した新たなプロトコルに対するパケット解析は以下の手順で行うことができる. 抽象クラスを継承する場合はその仮想関数 (純粋仮想関数を含む) を実装する必要がある.

- 対応させるプロトコルのヘッダフィールドを表現するヘッダクラスを実装する
  (pgen::tmp_header ヘッダクラスを実装)
- 対応させるプロトコルのパケットを表現するパケットクラスを実装する
  (pgen::tmp パケットクラスの実装)

これらの作業で新たなプロトコルに対するパケット解析をすることが可能になる. 実際に TMP のパケットを解析できるようにパケットクラスを実装する手順を詳細に以下に示す.

- pgen::header クラスを継承して pgen::tmp_header クラスを実装する. クラスの継承は以下の純粋仮想関数を実装することに帰着する
  - virtual void clear();
  - virtual void summary(bool moreinfo=false) const;
  - virtual void write(void* buffer, size_t bufferlen) const;
  - virtual void read(const void* buffer, size_t bufferlen);
  - virtual size_t length() const;
- pgen::packet クラスを継承して pgen::tmp クラスを実装する. クラスの継承は以下の純粋仮想関数を実装することに帰着する
  - virtual void init_headers();
  - virtual void clear();

5. 考察

従来までの Wireshark などのアプリケーションに頼った解析では以下のように, 簡単には新たなプロトコルの解析を行うことは難しかった Wireshark では新たなプロトコルに関する迅速な対応は難しく, プログラマブルな処理に適していない. 図 10 より,UDP の通信というのは解析できているが, それより上のレイヤでの通信内容を解析できていなく, 上位プロトコルが不明な UDPData として表示されてしまう.

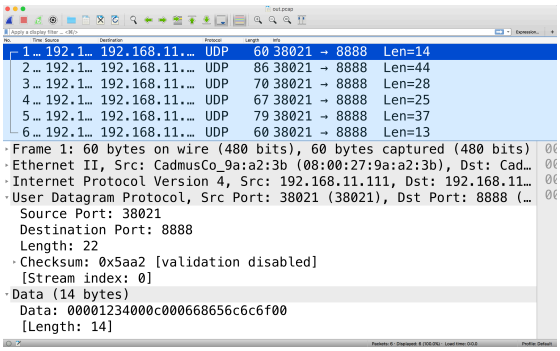

図 10　TMP を Wireshark でキャプチャ

```
pgen::pcap_stream packet_stream("eth0",
        pgen::open_mode::netif);
while (1) {
    uint8_t buf[10000];
    size_t recvlen =
        packet_stream.recv(buf, sizeof buf);

    if (is_tmp_packet(buf, recvlen)) {
        tmp pack(buf, recvlen);
        printf("[%s] 0x%04x: %s\n", getnow(),
          pack.TMP.id, pack.TMP.msg.c_str());
    }
}
```

図 11　TMP を C++で解析するプログラム

```
[13:11:32] 0x1234: Hello World
[13:11:41] 0x1234: Hi, Analyzing with PGM is awesome
[13:12:11] 0x1234: I think so too.
```

図 12　TMP の解析結果

LibPGEN を使用した場合, 先ほど示した少ないコードで TMP のパケットを簡単に解析できるようにし, 通信内容に対する柔軟なプログラムを組むことが可能である.TMP の通信を解析するプログラムと実行例を図 11,12 に示す. この例では, あるユーザが送信している相手をデータとしてまとめてログに書き込むスクリプトの実装例である.

このように今回 LibPGEN を実装して, 新たなプロトコルに対するパケット解析を簡単にし, それをプログラマブルに行うことでパケット解析を柔軟に行えるようにすることを示した.

また, 現在の LibPGEN の実装ではまだ実用段階とは言えず, 幾つかの改善点が存在する.

5.1　動作速度の高速化

現在の LibPGEN はパケット解析を簡単に行うために動作速度を犠牲にしている. パケットクラスでは, ヘッダとデータを分けて考えており, ヘッダ要素の値を一つでも変更すると, ヘッダ部分の全てをコピーしなくてはならない実装になっている. これでは, ヘッダの要素を更新するたびにデータのコピーが行われてしまい, 高速性を失ってしまう. また, パケット送信部分に関しても linux での実装は pf_packet を使用しているので高速通信を行うのは難しい (GBE は難しい) 高速化を実現させた場合, アプリケーションとしての使用用途もさらに増えるため, 今後は高速化の仕組みを組み込んでいく予定である.

5.2　より自動的なプロトコルに対する対応

本稿ではパケットクラスを実装することで, 新たなプロトコルに対して素早く対応できることを示した. だが対応する必要のあるプロトコルの数が多くなってきた場合, 人の実装ではコストがかかりすぎてしまう. このような場合に対応して, プロトコルを一定のフォーマットで記述したファイルを渡すと, 自動でそのプロトコルに対応するクラスを実装する連携アプリケーションを検討中である. これが可能になれば, さらにコストは軽減することが期待できる.

5.3　様々なプラットフォームでの使用

今回は libpgen を通信の解析, デバッグ用途として紹介をした. だが,6.1 で述べた高速化の問題点が改善された場合, アプリケーションとして利用することが可能である. 特に openflowsw などはワイドレイヤのパケットを解析するため, 強力なパケット解析能力を持つ LibPGEN には相性のいい分野であると言える. オーバレイネットワークのパケット生成など, 様々な用途での使用の可能性も考えることができる. そのような場合に向けて,LibPGEN をさまざまなプラットフォームで動作するように開発をしなくてはならない. 現在 LibPGEN は linux と bsd 系の os で動作をするが,libc が標準のものでなくなったり, 組み込み端末での動作に対しては想定されていない. また fpga などでのパケット転送高速化との研究との連携もすることができるので, なるべく, 既存の api に頼らず,c++のみの実装で使用できるように設計を行っていく予定である.

6.　結論

本稿では既存のパケット解析の課題を示し, それをパケット解析のライブラリを実装することによって解決する手法を提案し, 課題を解決できることを示した. 始めに既存のパケット解析の課題を示し, その課題を克服することによって, 柔軟なパケット解析を行えることを示した. プログラミング言語からパケット解析を行うことができることで, 従来までのツールに頼り場合より, 自動的, 多くの開発者の慣れ親しんだ手順で簡単にパケットを解析できることを示した. また, 新たなプロトコルにたいして, 簡単にライブラリ

を拡張できるようにすることで, 今後の IoT 社会などで行われる既存でない通信に迅速に対応できることを示した.

参考文献

[1] Chris Sanders・高橋基信・宮本久仁男・岡真由美 (2012) 実践パケット解析 第 2 版–Wireshark を使ったトラブルシューティング, O'REILLY

[2] 村山公保 (2004) 基礎からわかる TCP/IP ネットワーク実験プログラミング–Linux/FreeBSD 対応, Ohmsha

[3] 小俣光之 (2011) ルータ自作でわかるパケットの流れ, 技術評論社

[4] Scott Meyers・千住治郎 (2015) Effective Modern C++ – C++11/14 プログラムを進化させる 42 の項目, O'REILLY

[5] Martin Reddy・ホジソンますみ・三宅陽一郎 (2013) C++ のための API デザイン, SoftBank Creative