作って学ぶ
低レイヤーネットワーク
(前半)

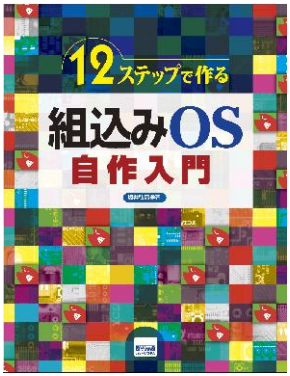
坂井弘亮

自己紹介

坂井弘亮(さかい・ひろあき)
http://kozos.jp/
個人でいろんな活動をしています

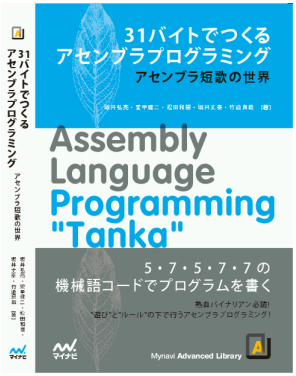
- 組込みOS自作(KOZOSプロジェクト)
- イベントへの出展・セミナーなど
  (オープンソースカンファレンス(OSC)など)
- SECCONへのコミット
- 雑誌記事や書籍執筆など
- アセンブラ短歌・六歌仙のひとり(白樺派)
- 技術士(情報工学部門)

書籍の執筆


12ステップで作る
組込みOS
自作入門


リンカ・ローダ
実践開発テクニック


全国津々浦々！
勉強会＆イベント
探訪記　完全版
坂井弘亮
北は北海道から南は
九州まで、各地の
IT系イベントを写真
と記録で振り返る。


31バイトでつくる
アセンブラプログラミング
アセンブラ短歌の世界
Assembly
Language
Programming
"Tanka"
5・7・5・7・7の
機械語コードでプログラムを書く


0と1のコンピュータ世界
バイナリで遊ぼう!


熱血!
アセンブラ入門
やらなければ、
一生
わからん!!


ハロー
"Hello, World"
OSと標準ライブラリのシゴトとしくみ
あなたが最初に書いた
プログラムにはまだ
「知らないこと」がたくさんある。

低レイヤーまわりを中心にいろいろやってます

セキュリティ・キャンプへの参加の経緯

- 組込み関連
  - 学習向け組込みシステム(KOZOS)の開発
  - 書籍の執筆
  - OSCなどのイベントへの出展・セミナー登壇

- セキュリティ関連
  - セキュリティ・キャンプへの参加
  - SECCONへのコミット

(経緯) 組込みOS開発者としてセキュリティ・キャンプに参加
→ セキュリティ界隈では低レイヤー技術が求められている
(組込み技術者とセキュリティ技術者のスキルセットは似ている)

この講義について

この講義の概要

(ホームページ上の説明)

前半でPC上でパケットを生成やキャプチャーするプログラムを自作することにより、パケットの構成を理解します。後半でArduinoのデジタル出力のみを使って10BASE-Tのフレームシーケンスを実際に送出する演習を行います。PCのOS上で動作するプログラムでは、Ethernetコントローラーが扱えるパケットしか送受信できませんが、マイコン等のハードウエアを自由に制御できる環境ではPC上では認知できないパケットもLAN上に送出可能です。それらの可能性を意識したセキュリティ対策が行える人材を育成します。

前半について

前半では，EthernetによるL2ネットワークを扱います．パケットをバイナリエディタでパケットを直接読み書きします．さらにパケットを直接送受信するツールを自作して，自作したパケットをネットワーク上に流してみます．
またツールの自作を通して，EthernetやIPルーティングの仕組みを学びます．LANコントローラは何をやっているのか，なぜIPアドレスは32ビットなのか，といったことを知ることができます．
最後に「パケット大運動会」という競技をやってみます．指定されたパケットをいち早く作成してサーバに送りつけたら得点，というものです．
この講義を受講すると，ARPくらいならば何も見ずにバイナリエディタで読み書きできるくらいには，パケットに詳しくなれると思います．

この講義の方針

高レイヤーの魅力は「いかに応用してすごいものを作るか」ですが，低レイヤーの魅力は「いかに深掘りして原理を知るか」にあると思っています．
今回の講義でも，EthernetやIPネットワークについて，そうした深掘りをしていくつもりです．
たとえば，MACアドレスとIPアドレスの違いとは何でしょうか？ なぜ2つのアドレス体系があるのでしょうか？ 片方だけじゃダメなのでしょうか？ また，MACアドレスは6バイトもあるのに，なぜIPアドレスは4バイトなのでしょうか？
これらは「あたりまえのこと」ではありません．それらがなぜなのか，深く考えてみたことはあるでしょうか．深掘りして考えることで，そういう「なぜ？」を知ることができます．

どんな内容？

CQ出版社のInterface誌で「パケットづくりではじめるネットワーク入門」という連載をしています.
記事の中ではパケットを直接送受信していろいろいじるようなライブラリを作って, そのライブラリを使っていろんなツールを作っているのですが, そんな感じでパケットを直接いじるようなツールを自作します. そしてパケット自体もバイナリエディタでいじります.
そうしたツール作りやパケット作りを通して, パケットやルーティングの仕組みを学んでしまおう, というのが目的です. 仕様を見るだけでなく, ものづくりを通して身を持って学ぶという感じです. 演習がメインになります.
いまさらツールを自作しても…と思ったりもするかもしれませんが, 自分で作ったツールは好きに改造して使えるという魅力(魔力?)があります. 「ツールを探す」だけでなく「ツールを作る」ことができるようになりましょう!

この講義の目的

- 既存のパケットデータやツールを使うだけでなく，自分でもパケットデータやツール を作ってみることで，EthernetとIPネットワークの仕組み・仕様だけでなく その目的・理由を実践的に理解する．(なぜIPアドレスは32ビットなのか？など)
- 既にあるものを使うだけでなく自作してみることで，使うだけでは知ることが できなかったような深い知識を学ぶことができる，ということを実感してもらう．車輪の再発明は悪いことではないことを知ってもらう．
- 「ほしいものを探す」だけでなく「ほしいものを作る」ということを知る きっかけにする．
- ネットワークのハードウェアとソフトウェアの間を埋めて繋げる．どちらかだけ，もしくは両方をバラバラに知っている場合は多いが，両者が繋がった 形で理解している人は少ない．点の理解を線にして繋げる．

前提知識について

以下の知識が必要ですが, 知識が無くても当日までに予習してくればOKです.

- ネットワーク基礎
  (IPアドレス, MACアドレス, TCP/IP, パケット, ルーティング)

- UNIXのシェル環境でのCUI操作 (説明はCUIベースで行います)
  (ls, cd, cp, cat など各種コマンド. ファイル参照, ファイル編集など)

- できれば, 加えてC言語の基礎知識があると, 講義がより理解しやすいです.

PCについて

PCについて

キャンプ当日は，参加者ひとりに1台，PCが貸し出されます．貸し出しPCには以下の環境はセットアップしてあるので，貸し出しPCを使う人は気にしなくていいです(準備不要です)．自分のPCを使うというひとは準備してきてください．

- 演習では，何らかのLinuxディストリビューション環境が必要です．何らかのLinuxディストリビューション環境を用意しておいてください．ネイティブにインストールしてもいいですし，VM上でも構いません．
- VM上で使う場合は，ネットワーク設定を「ブリッジ」にしておいてください．
- C言語の開発環境を入れておいてください．
- 以下のツールをビルドしておいてください．
  http://kozos.jp/software/pkttools.html
  → pkttools-1.14.zip
- ビルドしたツールが動作することを確認しておいてください．

PCについて

- その他，各自で使いたいツールがあれば入れておいてください．(GUI環境とか)
- 以下のツールを入れて使えるようにしておいてください．
  - テキストエディタ・バイナリエディタ (種類はなんでもいいですが，自分で使えるようにしておいてください)
  - Wireshark
- 講義では有線LANを使用しますので，有線のLANポートがついてるPCがのぞましいです．いちおう，ひとり1本，USBのLANアダプタを貸し出しします．型番は以下のものです．
  Buffalo LUA3-U2-ATX
- ホスト側とファイルをやりとりする手段を確立しておく(方法はなんでもいいです)
- 演習用に用意するPCには，以下のイメージがインストールされています．
  http://kozos.jp/vmimage/progtool.html「上記イメージの更新版」

現在，PCはキャンプ会場内ネットワークに
接続されています

演習が始まったら，部屋内の
隔離ネットワークに切替えます
(インターネット接続は失われます)

ダウンロードしたいものなどあれば，
今のうちにダウンロードしておいてください

(参考)

Windows上のWiresharkでUSB LANアダプタを認識しない場合 (キャプチャ対象の選択肢に出てこない)には，以下をやってみてください

1. Wiresharkを閉じる

2. 管理者コマンドプロンプトを開く
   Windows 8 の場合は，スタートボタンを右クリック

3. 以下のコマンドを実行してWinPcapのサービスを再起動する

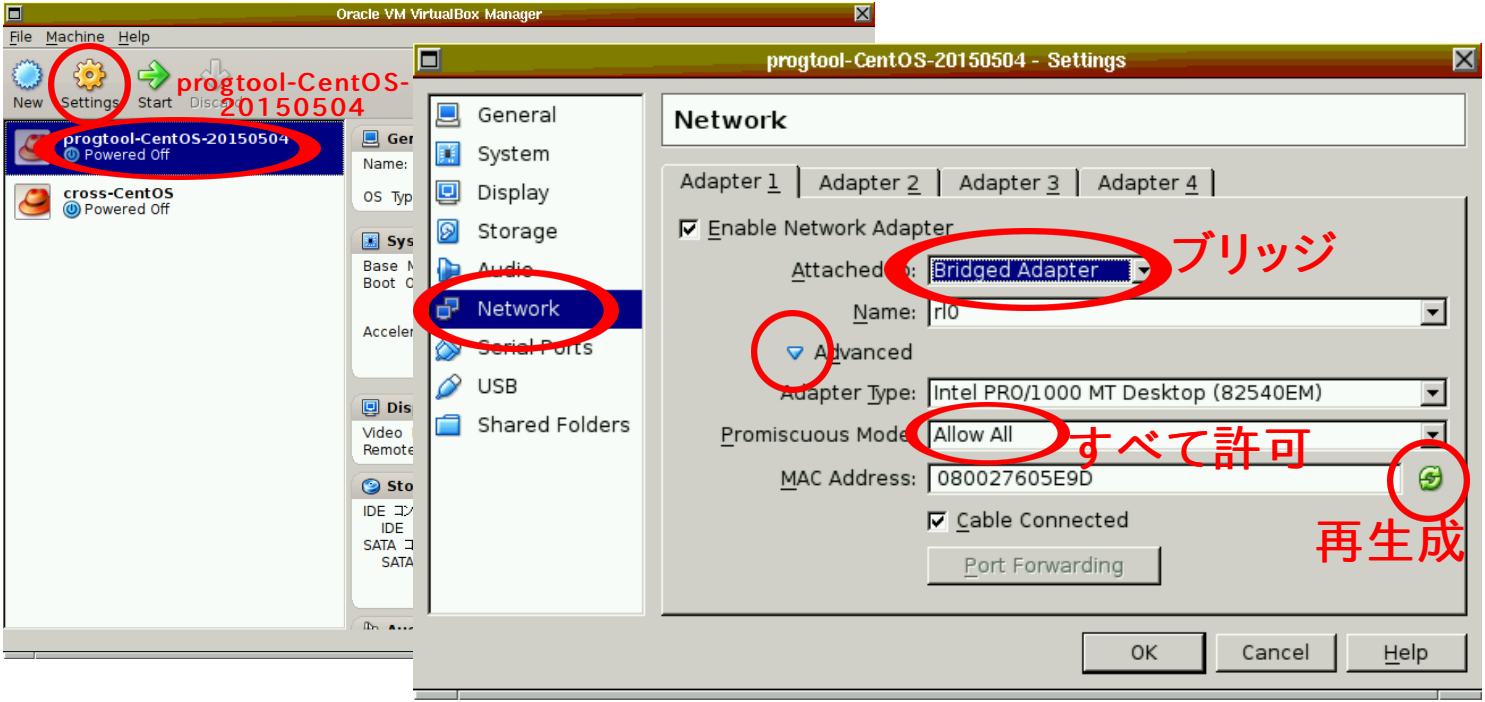
```
net stop npf
net start npf
```

4. Wiresharkを再起動する

演習用PCの
VM環境について

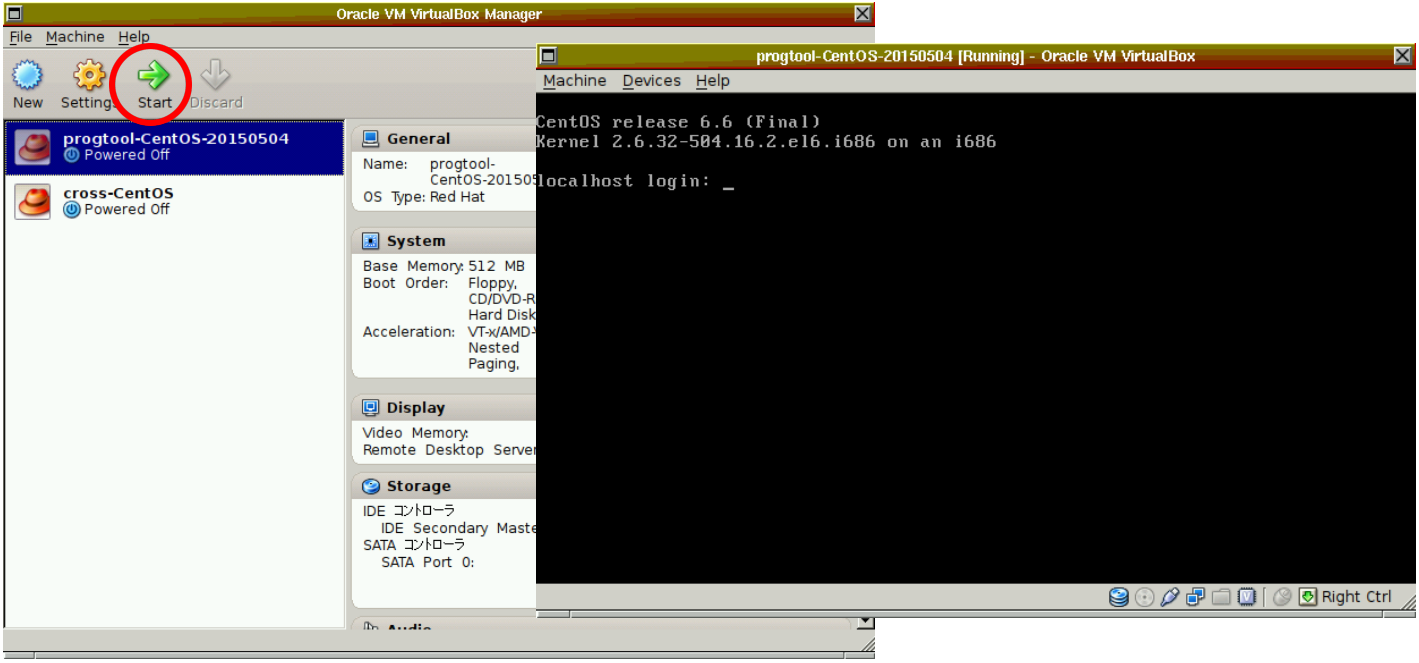
VM環境の準備

ブリッジ・プロミスキャス許可・MACアドレス再生成



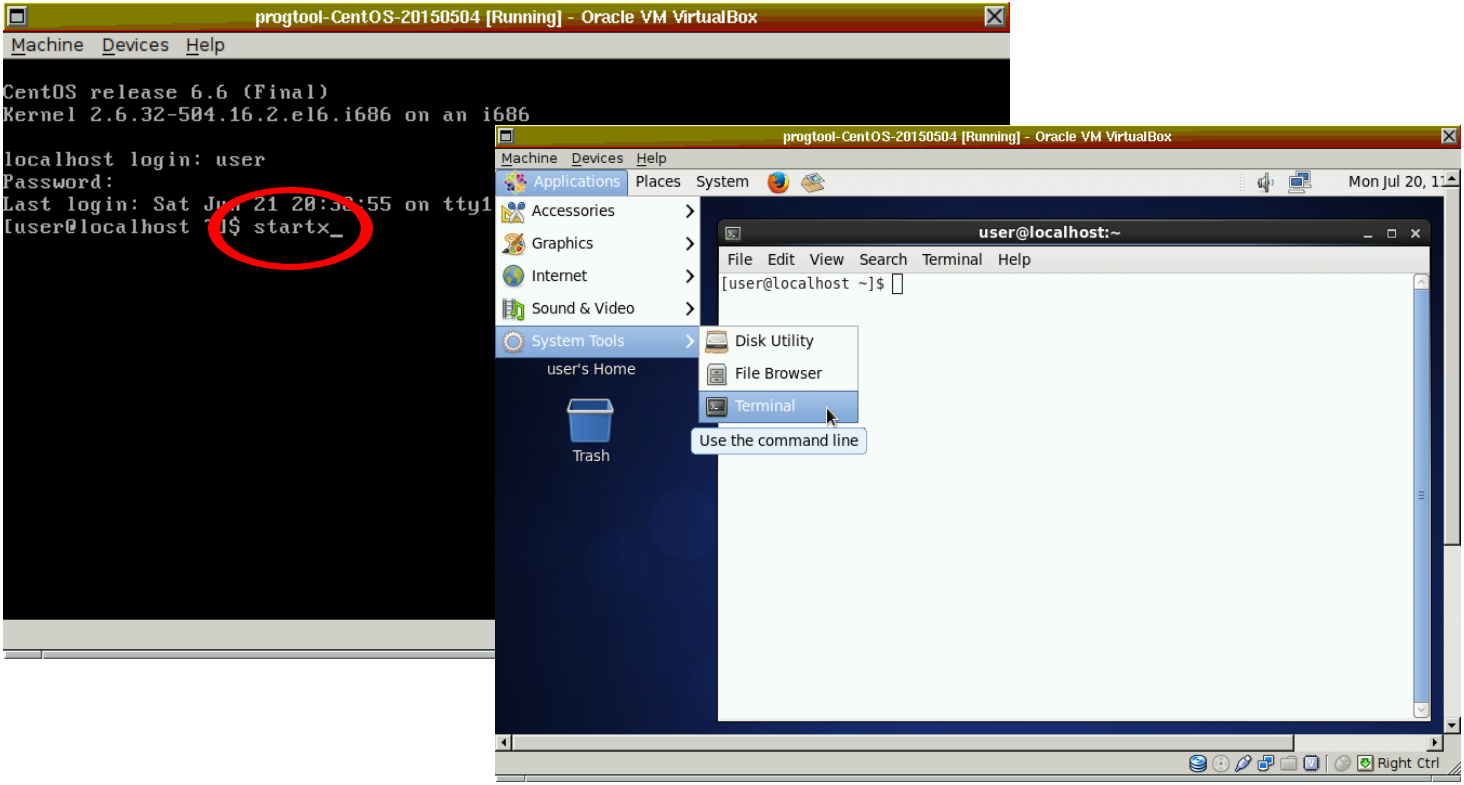
VM環境の起動

user / progtooluser でログイン



(スーパーユーザは root / progtoolroot)

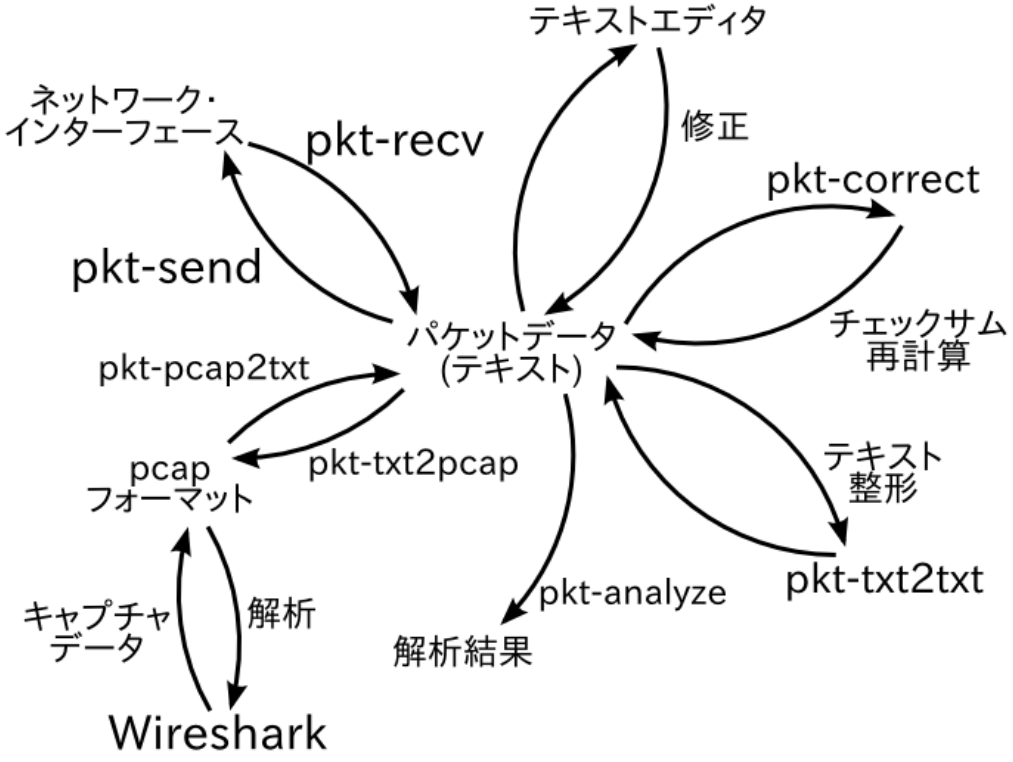
startx でGUI起動



pkttoolsでパケットをいじろう

pkttools (http://kozos.jp/software/)

pkttoolsはパケットを操作するツール集です



以下のファイルをダウンロードしてください

pkttools-1.14.zip
pkttools-1.8.zip
pkttools-1.14-win.zip

ダウンロード後に，PCを隔離ネットワークに
切替えます

では，隔離ネットワークに切替えます

pkttoolsを使いこなそう

(ツール一覧)

pkt-recv	パケットキャプチャして，受信データをテキスト出力
pkt-send	テキスト入力されたパケットを送信
pkt-txt2txt	テキスト入力されたパケットをテキストで再出力(テキストの整形)
pkt-txt2pcap	テキスト入力されたパケットをpcapフォーマットに変換
pkt-pcap2txt	pcapフォーマットを解読してテキスト出力する
pkt-txt2pcapng	テキスト入力されたパケットをPcapNgフォーマットに変換
pkt-pcapng2txt	PcapNgフォーマットを解読してテキスト出力する
pkt-txt2bin	テキスト入力されたパケットをバイナリデータに変換
pkt-bin2txt	バイナリデータを変換してテキスト出力する
pkt-fragment	IPv4/IPv6パケットをフラグメント分割する
pkt-defragment	フラグメント分割されたIPv4/IPv6パケットを再構築する
pkt-analyze	テキスト入力されたパケットを解析
pkt-correct	テキスト入力されたパケットのチェックサムを再計算して再出力
pkt-check	テキスト入力されたパケットの簡易チェックをして再出力
pkt-filter	テキスト入力されたパケットをフィルタして再出力(※１)
pkt-change	テキスト入力されたパケットを変換して再出力(※１)
pkt-pingrep	テキスト入力されたパケットからpingの応答パケットを出力する

pkttoolsを使いこなそう

(ツール一覧その2)

ソケットをオープンしてデータを送受信する簡易ツールが tools というディレクトリ に置いてあります (使いかたはREADME参照)
IPv4/IPv6, TCP/UDP, 送信/受信の組合せで, 以下の合計16通りのツールがあります

```
tcp-send    / tcp6-send    / udp-send    / udp6-send
tcp-recv    / tcp6-recv    / udp-recv    / udp6-recv
tcp-request / tcp6-request / udp-request / udp6-request
tcp-reply   / tcp6-reply   / udp-reply   / udp6-reply
```

(パケットのサンプルデータ)

ARPやICMPパケットのサンプルデータが samples というディレクトリに置いてあります

まずは試してみよう

(Linux)

```
$ unzip pkttools-1.14.zip
$ cd pkttools-1.14
$ make
$ su
# ./pkt-recv -i eth0
```

(Windows)

pkttools-1.14-win.zip をダウンロードし展開

```
C:¥> pkt-recv
         0       ...
         1       ...
         2       ...
C:¥> pkt-recv -i 1
```

使いかたの例

(添付のREADMEも参照してください)

(パケットの受信)

```
% pkt-recv -i eth0
```

(パケットをキャプチャして解析して表示)

```
% pkt-recv -i eth0 | pkt-analyze
```

(パケットをキャプチャして各フィールド値を表示)

```
% pkt-recv -i eth0 -a
```

(パケットを一旦保存し解析．さらに別インターフェースに強制送信)

```
% pkt-recv -i eth0 > capture.txt
% cat capture.txt | pkt-analyze
% cat capture.txt | pkt-send -i eth1
```

使いかたの例

(添付のREADMEも参照してください)

(eth0 → eth1 にパケットをブリッジする)

```
% pkt-recv -i eth0 | pkt-send -i eth1
```

(IPパケットのみブリッジ)

```
% pkt-recv -i eth0 ETHERNET.TYPE==0x0800 | pkt-send -i eth1
```

(宛先が192.168.1.Xのパケットのみブリッジ)

```
% pkt-recv -i eth0 ”IP.DST_ADDR>=192.168.1.0” ¥
        | pkt-txt2txt ”IP.DST_ADDR<=192.168.1.255” | pkt-send -i eth1
```

(TTL値を書き換えてチェックサム再計算してブリッジ)

```
% pkt-recv -i eth0 IP.TTL=0xFF | pkt-correct | pkt-send -i eth1
```

使いかたの例

(添付のREADMEも参照してください)

(パケットをpcapフォーマットで保存し，あとで Wireshark で開く)

```
% pkt-recv -i eth0 | pkt-txt2pcap > capture.pcap
% wireshark capture.pcap
```

(Wiresharkでキャプチャし保存したパケット(capture.pcap)をeth0に強制送信)

```
% cat capture.pcap | pkt-pcap2txt | pkt-send -i eth0
```

(pcapngからpcapフォーマットへの変換)

```
% cat capture.pcapng | pkt-pcapng2txt | pkt-txt2pcap > capture.pcap
```

使いかたの例

(添付のREADMEも参照してください)

（１個のパケットをバイナリデータとして保存しバイナリエディタで開く）

```
% pkt-recv -i eth0 -l 1 | pkt-txt2bin > capture.bin
% hexedit capture.bin
```

（バイナリエディタで作成・編集したパケットを送信）

```
% hexedit capture.bin
% cat capture.bin | pkt-bin2txt | pkt-send -i eth0
```

（キャプチャしたパケットを改造してチェックサムを再計算し再送信）

```
% pkt-recv -i eth0 > capture.txt
% vi capture.txt
% cat capture.txt | pkt-correct | pkt-send -i eth0
```

10分ほど時間をとりますので，
自由にいろいろ試してみてください

ARPやICMPのパケットをキャプチャして
見てみよう!

注意: pkt-recv, pkt-send はスーパーユーザで実行する必要があります
(その他のツールは一般ユーザでも実行できます)

pkttoolsはパケットの直接送受信に
「RAWソケット」を使っています

RAWソケットとは何か？

ネットワーク・パケットをそのまま生(RAW)で送受信する，Linuxの機能

RAWソケットの使いかたを見てみる
pkttools-1.8.zipを展開して，rawsock.c を見てみよう
(注:pkttools-1.14でなくpkttools-1.8を参照してください)

以下の関数があります

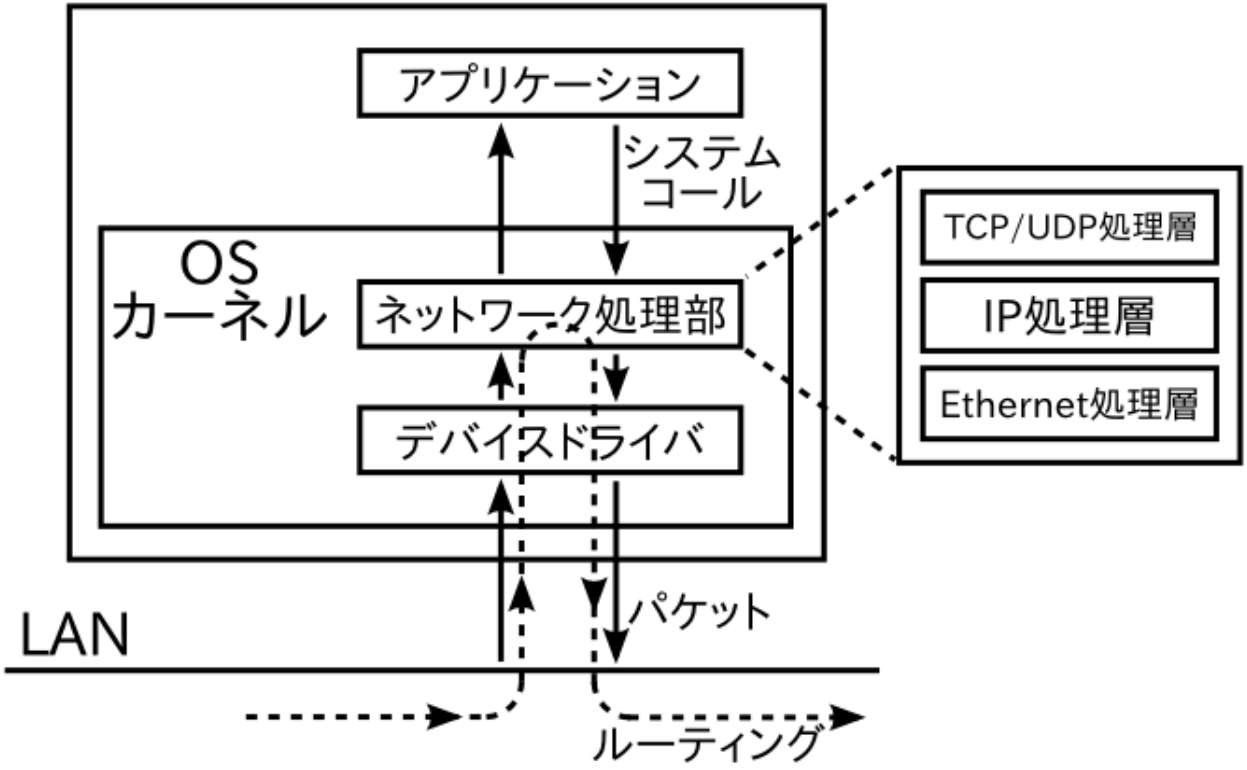
(パケット受信用のRAWソケットのオープン)
```
int rawsock_open_recv(char *ifname, unsigned long flags, int *bufsizep)
```

(パケット送信用のRAWソケットのオープン)
```
int rawsock_open_send(char *ifname, unsigned long flags)
```

(RAWソケットを利用した，パケットの受信)
```
int rawsock_recv(int fd, char *recvbuf, int recvsize, struct timeval *tm)
```

(RAWソケットを利用した，パケットの送信)
```
int rawsock_send(int fd, char *sendbuf, int sendsize)
```

パケットの流れ


アプリケーション
システム
コール
OS
カーネル
ネットワーク処理部
デバイスドライバ
TCP/UDP処理層
IP処理層
Ethernet処理層
パケット
LAN
ルーティング

パケットの受信の実験

ここからは，RAWソケットを使った送受信の実験用プログラムを C言語で書いて試してみます

サンプルコードを提示してありますので，C言語がわからないという人も，とりあえず入力して試してみてください

やっぱしC言語わからない！という人は，かわりにpkttoolsを使って実習を進めてもOKです

RAWソケットを利用して
簡単なパケットキャプチャを作ってみよう

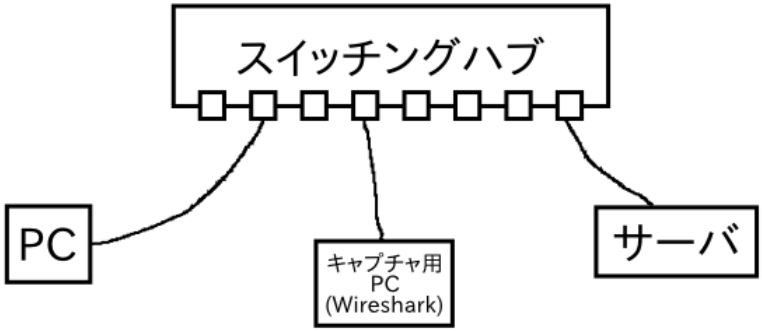
```
-------- simplerecv.c --------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include ”rawsock.h”

int error_exit(char *message)
{
  fprintf(stderr, ”Error: %s¥n”, message);
  exit(1);
}

int main(int argc, char *argv[])
{
  int fd, size;
  char buffer[65536];
  fd = rawsock_open_recv(argv[1], 0, NULL);
  size = rawsock_recv(fd, buffer, sizeof(buffer), NULL);
  if (size > 0) write(1, buffer, size);
  return 0;
}
```

コンパイルして実行してみる

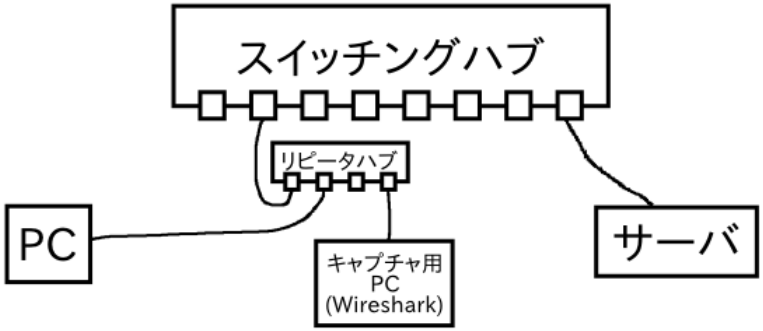
(注:pkttools-1.8.zipのrawsock.cを利用する)

```
$ wget http://kozos.jp/software/pkttools-1.8.zip
$ unzip pkttools-1.8.zip
$ cp pkttools-1.8/rawsock.c .
$ gcc simplerecv.c rawsock.c -o simplerecv -Wall
$ su
# ./simplerecv eth0 > packet.bin
(ここで何かパケットを送受信する)
# exit
$ hexdump -C packet.bin
```

リピータハブでキャプチャする



(a) これではすべてのパケットをキャプチャできない



(b) リピータハブを間に挟むことで，すべてのパケットをキャプチャする

パケットの解析(Ethernetヘッダ)

出力されるバイナリデータ(パケット)は，
先頭がEthernetヘッダになっています

■ 演習1−1
「Ethernet ヘッダ」あたりでググッてEthernetヘッダのフォーマット図を見て，キャプチャしたパケットに照らし合わせてみよう!

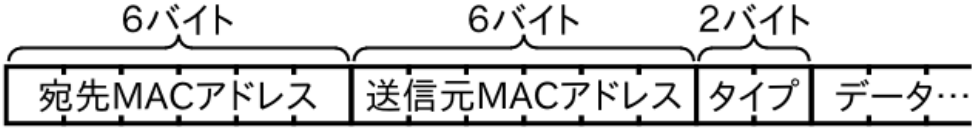
■ 演習1−2
Ethernetヘッダを解析するための構造体が定義されたヘッダファイルが /usr/include以下にあります．探してみよう!

■ 演習1−3
その構造体を，キャプチャしたバイナリデータ(パケット)に照らし合わせてみよう!

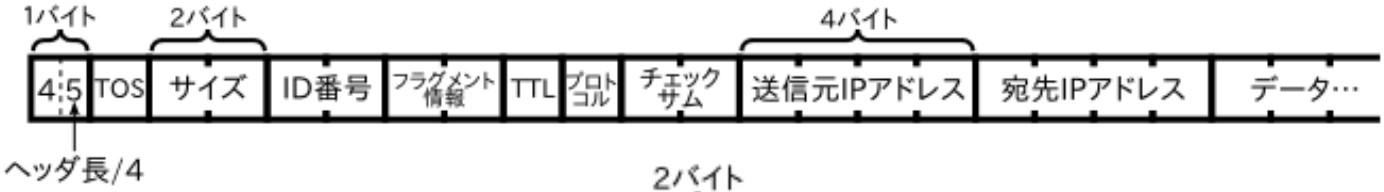
■ 演習1−4
ググッて出てきたフォーマット図を，構造体と照らし合わせてみよう!

ヘッダ構造

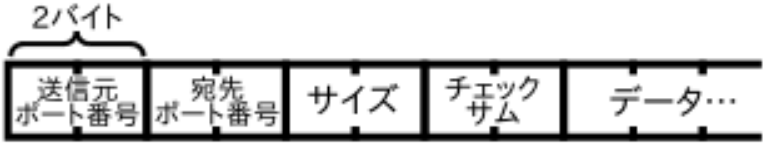
Ethernet



IP



UDP



ICMP

ARP

(固定長部分)

(可変長部分:EthernetとIPの場合)

パケットの解析(Ethernetヘッダ)

出力されるバイナリデータ(パケット)は，
先頭がEthernetヘッダになっています

■ 演習1−5
ヘッダファイルから，IPパケットの場合のEthernetのタイプフィールドの値を 調べてみよう！ もしくはググッて調べてみよう！ キャプチャしたパケットに照らし合わせてみよう！

■ 演習1−6
IPパケットのみキャプチャするように改造して，IPパケットをキャプチャしてみよう！
もしくはpkttoolsのフィルタ機能で，IPパケットだけをキャプチャしてみよう！ (簡易フィルタを使う方法と，pkt-filterを使う方法があります)

Interface誌2014年8月号
(http://www.kumikomi.net/interface/contents/201408.php)

パケットの解析(IPヘッダ)

パケットがIPパケットならば，Ethernetヘッダの
次はIPヘッダになっています

■ 演習2−1
「IP ヘッダ フォーマット」あたりでググッてIPヘッダのフォーマット図を見て，キャプチャしたパケットに照らし合わせてみよう!

■ 演習2−2
IPヘッダを解析するための構造体が定義されたヘッダファイルが/usr/include以下にあります．探してみよう!

■ 演習2−3
その構造体を，キャプチャしたバイナリデータ(パケット)に照らし合わせてみよう!

■ 演習2−4
ググッて出てきたフォーマット図を，構造体と照らし合わせてみよう!

パケットの解析(IPヘッダ)

パケットがIPパケットならば，Ethernetヘッダの
次はIPヘッダになっています

■ 演習2−5
ヘッダファイルから，UDPパケットの場合のIPヘッダのプロトコル値を 調べてみよう！ もしくはググッて調べてみよう！ キャプチャしたパケットに照らし合わせてみよう！

■ 演習2−6
UDPパケットのみキャプチャするように改造して，UDPパケットをキャプチャしてみよう！

もしくはpkttoolsのフィルタ機能で，UDPパケットだけをキャプチャしてみよう！ (簡易フィルタを使う方法と，pkt-filterを使う方法があります)

パケットキャプチャの改造

フィルタ機能を追加しよう

■ 演習3−1
特定のパケット(ARPやIPなど)のみをキャプチャするパケットキャプチャにしてみよう.
もしくはpkttoolsで, 特定のパケットのみキャプチャしてみよう.

■ 演習3−2
特定のプロトコル(TCPなど)のみをキャプチャするパケットキャプチャにしてみよう.
もしくはpkttoolsで, 特定のプロトコルのみキャプチャしてみよう.

■ 演習3−3
特定のアドレス宛のパケットのみをキャプチャするパケットキャプチャにしてみよう. (アドレスが完全一致の場合と, アドレスの一部のみ一致の場合を考える)
もしくはpkttoolsで, やってみよう.

パケットキャプチャの改造

ヘッダ上のフィールドの値を読み書きする例
pkttools-1.8 の filter.c / change.c を参考にしよう

(IPヘッダの例)

```
static int filter_ip(int *f, char *buffer, int size)
{
  ...
  struct ip *iphdr;
  ...
  iphdr = (struct ip *)pktbuf;
  ...
  if (ntohl(iphdr->ip_dst.s_addr) == 0xC0A80102) ...
  ...
    switch (iphdr->ip_p) {
    case IPPROTO_ICMP: ...
    ...
    case IPPROTO_TCP:  ...
  ...
```

パケットの解析(UDPヘッダ)

パケットがUDPパケットならば，IPヘッダの
次はUDPヘッダになっています

■ 演習4−1
「UDP ヘッダ フォーマット」あたりでググッてUDPヘッダのフォーマット図を見て，キャプチャしたパケットに照らし合わせてみよう！

■ 演習4−2
UDPヘッダを解析するための構造体が定義されたヘッダファイルが /usr/include以下にあります．探してみよう！

■ 演習4−3
その構造体を，キャプチャしたバイナリデータ(パケット)に照らし合わせてみよう！

■ 演習4−4
ググッて出てきたフォーマット図を，構造体と照らし合わせてみよう！

パケットの解析(ARPヘッダ)

パケットがARPパケットならば，Ethernetヘッダの
次はARPヘッダになっています

■ 演習5−1
「ARP ヘッダ フォーマット」あたりでググッてARPヘッダのフォーマット図を見て，キャプチャしたパケットに照らし合わせてみよう!

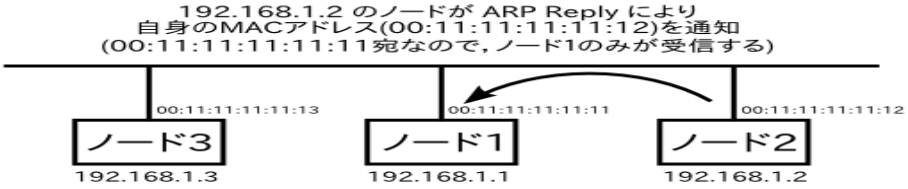
■ 演習5−2
ARPヘッダを解析するための構造体が定義されたヘッダファイルが/usr/include以下にあります．探してみよう!

■ 演習5−3
その構造体を，キャプチャしたバイナリデータ(パケット)に照らし合わせてみよう!

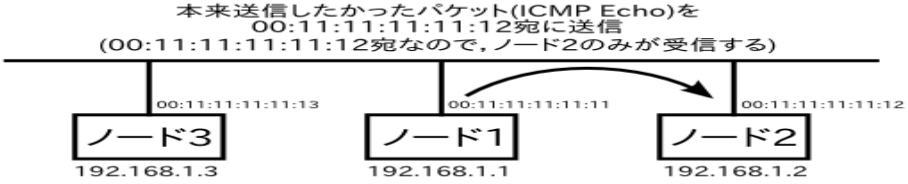
■ 演習5−4
ググッて出てきたフォーマット図を，構造体と照らし合わせてみよう!

ARPのシーケンス

(a) ARP Request による問い合わせ



(b) ARP Reply による応答



(c) ICMP Echo の送信

PCAPフォーマットの利用

PCAPフォーマットとは何か？

キャプチャしたパケットを保存するためのフォーマット
(Wiresharkなどが対応)

PCAPの使いかたを見てみる
pkttools-1.8 の pcap.c を見てみよう

以下の関数があります

```
(PCAPフォーマットの読み込み)
int pkt_pcap_read(FILE *fp, char *p, int size,
                  int *capsizep, int *origsizep, struct timeval *tm)

(PCAPフォーマットの書き込み)
int pkt_pcap_write(FILE *fp, char *p,
                   int capsize, int origsize, struct timeval *tm)
```

PCAPフォーマットの構造

PCAPフォーマットで出力する簡単なパケットキャプチャを作ってみよう

simplerecv.cを以下のように改造する → recv2pcap.c

```
-------- recv2pcap.c --------
#include ”pcap.h”
  ...
  struct timeval t;
  ...
  while (1) {
    size = rawsock_recv(fd, buffer, sizeof(buffer), &t);
    if (size > 0) {
      pkt_pcap_write(stdout, buffer, size, size, &t);
      fflush(stdout);
    }
  }
```

コンパイルして実行してみる

```
$ gcc recv2pcap.c rawsock.c pcap.c -o recv2pcap -Wall
$ su
# ./recv2pcap eth0 > packet.pcap
(ここで何かパケットを送受信する)
^C (Ctrl-Cで停止)
# exit
$ wireshark packet.pcap
```

バイナリエディタでPCAPファイルを見てみよう

```
$ hexedit packet.pcap
```

パケットの送信の実験

RAWソケットを利用して
簡単なパケット送信ツールを作ってみよう

```
-------- simplesend.c --------
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include ”rawsock.h”

int error_exit(char *message)
{
  fprintf(stderr, ”Error: %s¥n”, message);
  exit(1);
}

int main(int argc, char *argv[])
{
  int fd, size;
  char buffer[65536];
  fd = rawsock_open_send(argv[1], 0);
  size = read(0, buffer, sizeof(buffer));
  if (size > 0) rawsock_send(fd, buffer, size);
  return 0;
}
```

コンパイルして実行してみる

```
$ gcc simplesend.c rawsock.c -o simplesend -Wall
$ hexedit packet.bin
$ su
# cat packet.bin | ./simplesend eth0
```

送信の実験

バイナリデータ(パケット)をバイナリエディタで
作成して送信できます

■ 演習6−1
ARPパケットをバイナリエディタで作成して，演習用PCに送信して
ARPテーブルに登録してみよう!

■ 演習6−2
ICMP Echo パケットをバイナリエディタで作成して，演習用PCに
送信して応答させてみよう!

■ 演習6−3
UDPパケットをバイナリエディタで作成して，隣のひとのPCに送信
してみよう!

ツールを作成してみよう

簡易アナライザの作成

簡易アナライザを作成してみよう!

- パケットを受信して, Ethernetヘッダの解析結果を出力する
- IPパケットならば, IPヘッダの解析結果を出力する

簡易ブリッジの作成

簡易ブリッジを作成してみよう!

- 受信用のRAWソケットと, 送信用のRAWソケットを, 別々のインターフェースにbind()してオープンする
- 受信用のソケットはプロミスキャスモード(自分宛だけでなくすべてのパケットを 受信するモード)にする (rawsock.c の PACKET_MR_PROMISC の処理を参照)
- 受信用のRAWソケットでパケットを受信したら，送信用のRAWソケットに送信する. このとき, パケットの解析結果を出力する
- eth0 → eth1 方向と eth1 → eth0 方向の2つのプロセスを立ち上げる

ping応答ツールの作成

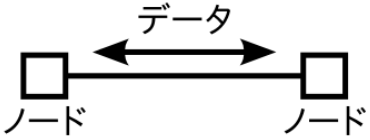
すべてのpingに応答するツールを作成してみよう!

- RAWソケットを受信用と送信用にオープンする
- パケットを受信し, ARP Request ならば ARP Reply を応答する
- ICMP Echo ならば ICMP Echo Reply を応答する

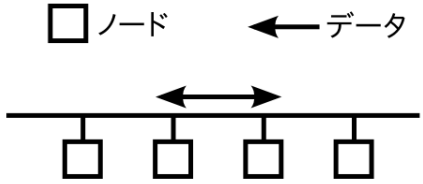
簡易アナライザについて考えてみよう

- EthernetヘッダとIPヘッダの特徴を考えてみよう
- EthernetヘッダとIPヘッダで，ソフトウェアが処理しやすいのはどちらか？ ハードウェアが(ビットシーケンスとして)処理しやすいのはどちらか？
- LANコントローラの役割を考えてみよう
- Ethernetの役割を考えてみよう
  LANコントローラは何をやっているのか？
- なぜEthernetでは先頭に宛先MACアドレスがあるのか？
- そもそも，なぜ通信はパケット単位で送られるのか？

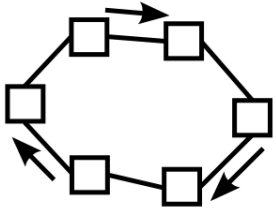
P2P通信



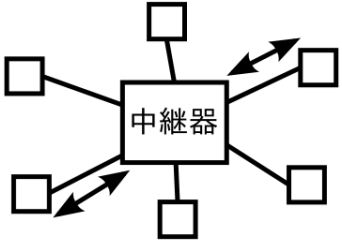
多ノード間通信



(a) バス型



(b) リング型



(c) スター型

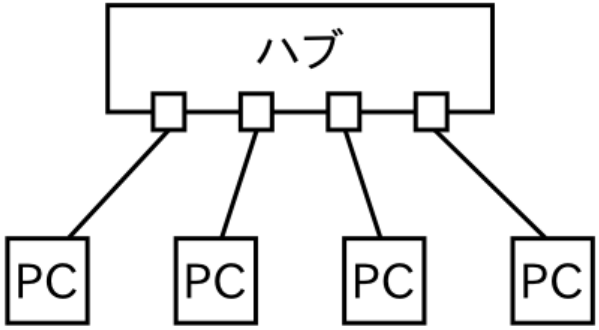
多ノード間通信のために必要なこと

- 通信をパケット単位に分割する
- パケットに宛先を明示する

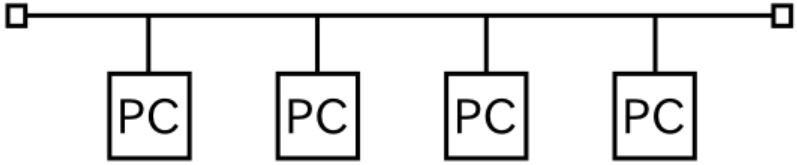
Ethernetは, バス型のトポロジでパケット単位での通信により 多ノード間通信を実現するもの

(じゃあIPは何のためのもの?)

実際の装置構成とネットワーク図



(a) 実際の装置構成



(b) ネットワーク構成図

簡易ブリッジについて考えてみよう

- 送信先のインターフェースを複数にして，宛先ごとに送信先を切替えるような 動作を実装できないか？
- MACアドレスベースで行う場合と，IPアドレスベースで行う場合の違いを考える
- IPアドレスベースで行う場合は，アドレス完全一致でなく，アドレスの一部のみ 一致で判断できないか？
- IPの役割を考えてみよう
  MACアドレスとは別に，なぜIPアドレスがあるのか？
  MACアドレスだけでは通信できないのだろうか？
- なぜIPアドレスは32ビットなのか？
  (MACアドレスは6バイトもあるのに!)

広域ネットワークのために必要なこと

- Ethernetでの通信範囲を分割してLANにする
- LAN間でパケットを中継する
  そしてその中継ルールは, 簡易に書けなければならない
  (全世界のノード情報の保持が必要, とかじゃダメ)

IPはルーティングのルールを, IPアドレスとマスクによって LAN単位でまとめて簡単に書けるようにしたもの

簡易ルータへの挑戦

- 簡易ブリッジとping応答ツールを組み合わせて，簡易ルータにできないか？
- MACアドレスの管理と書き換えが必要
- 他に何をすればルータになるのか，考えてみよう (EthernetのまとまりでLANを分割したい)

作成したツールをOSSとして公開してみよう

- ソースコード整理, 流用元のライセンス確認
- ライセンスどうするか？
- ドキュメントを書こう
- githubやホームページ上で公開など
- 名前をどうするか？ (けっこう重要です)

パケット
大運動会

ルール

- 指示されたパケットを素早く作ってサーバに投げつけてください
  ※ 作る方法は自由です！自分なりに工夫してください！
- ソースMACでプレイヤーを識別して得点が入ります
- サーバのIPアドレスは，XXX.XXX.XXX.XXX です

禁止事項

- スイッチングHUBのMACテーブルの汚染(不正なソースMACのパケットを投げる)
- 他人のMACアドレスの利用によるなりすまし (ソースMAC，ARPなど)
- 大量のパケットを投げつける行為(設問的に必要なものは除く)
- その他，競技ルールを外れた攻撃

第0波(練習)

■ ARP問題01

競技の最初に, その後にパケットをサーバに送りつけるためのサーバのMACアドレスを 知る必要があります. ARP Request をサーバに対して投げて, MACアドレスを知って ください.

■ ARP問題02

標準ではMACアドレスがプレイヤー名として登録されますが, サーバに対して ARP Reply を投げて, ARPの後ろの空きにプレイヤー名を入れておくと, プレイヤー名がその名前で書き変わります.

第1波

■ ICMP問題01
サーバに対する ICMP Echo

■ UDP問題01
サーバのポート1000番に対するUDP.

■ TCP問題01
サーバに対してのポート10000からのSyn/Ack

■ IP問題01
サーバに対しての Land Attack

■ ETH問題01
Ethernetのタイプ値が 0xFFFF で, Ethernetヘッダの後に "This is Okashina Frame." という文字列が入っているへんなEthernetフレーム

第2波

■ ICMP問題02
サーバに対する ICMP Echo Reply なのだけど，IPのチェックサムが0xabcd，ICMPのチェックサムが0xef01になっている．

■ UDP問題02
サーバのポート0番に対するUDP．ただしTTLがゼロ．

■ TCP問題02
サーバに対してのポート19999からポート20000へのTCPパケットで，Syn/Ack/Psh/Rst/Finフラグが全部立っているというわけのわからないもの．

■ IP問題02
中身はIPなのだけど，IPのバージョン番号が「5」になっているという わけのわからないサーバに対してのICMPパケット

■ ETH問題02
VLANタグが2重についている，サーバ宛のICMP Echoのパケット (VLAN ID が10と20)

第3波

■ ARP問題03
IPv6の ARP Reply (なんだそりゃ)

■ ICMP問題03
サーバに対する ICMP Echo Reply のコード値の部分に以下の文字列を1文字ずつ 入れて送る "Shinko ha kohada no chiisai monode, natsu no jikini taberu kotoga dekiru."

■ UDP問題03
IPのチェックサム値が 0x1234 で, UDPのチェックサム値が 0x5678 になっている UDPパケット.

■ TCP問題03
サーバに対してのSyn/Ackなのだけど, URGフラグのON/OFFで以下の文字列を 1ビットずつ送る. "Natsuyasumi"

■ IP問題03
IPヘッダサイズが最大になっていて, ヘッダの空きがオプションとかではなくて "This packet has a large header and key." という文字列で単に埋められているという サーバに対してのICMPパケット

■ ETH問題03
宛先MACアドレスの下4バイトにサーバのIPアドレスが入っているような サーバ宛の ICMPパケット.

おつかれさまでした！