ANDPAD

Rubyでつくるパケットキャプチャツール

2025/01/17 東京Ruby会議12 前夜祭

@ydah

株式会社アンドパッド 開発本部 SWE

$ whoami

- ydah(わいだー)
- GitHub: @ydah / 旧Twitter: @ydah_
- 株式会社アンドパッドSWE
- Kyobashi.rb創設メンバ、Ruby関西メンバ
- 大阪Ruby会議04のチーフオーガナイザ
- #LR_parser_gangs
- 「終まで飲めば #rubyfamily」
- 新米Rubyコミッター(2024/12~)

LTなので大事なことをはじめに言います

関西Ruby会議08

場所: 先斗町歌舞練場

開催日: 2025年6月28日（土）

共催: Ruby関西、Kyoto.rb、Kobe.rb
Kyobashi.rb、AKASHI.rb、Ruby舞鶴
Ruby Tuesday、Shinosaka.rb、naniwa.rb

本日のトークのゴール

ネットワークプログラミングを

完全に理解する

だがしかし、このトーク

LT(5分)

本日のトークのゴール

Linuxネットワークプログラミングを

雰囲気だけでも完全に理解する

今日のコード

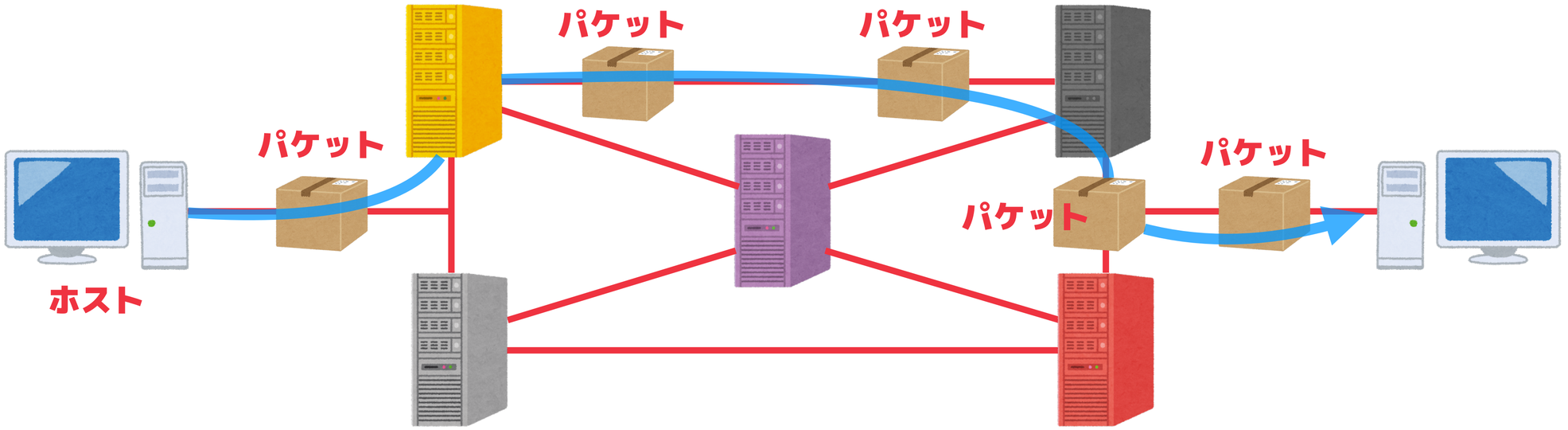
https://github.com/ydah/redhound

まず必要最低限の知識

(多分、スキップする)

パケットとは

ネットワーク上でデータを送受信する際に小さく分割されたデータの単位。大きなデータを効率的かつ信頼性を持って送るために、データはパケットに分割され、宛先に届いた後に再構成される。

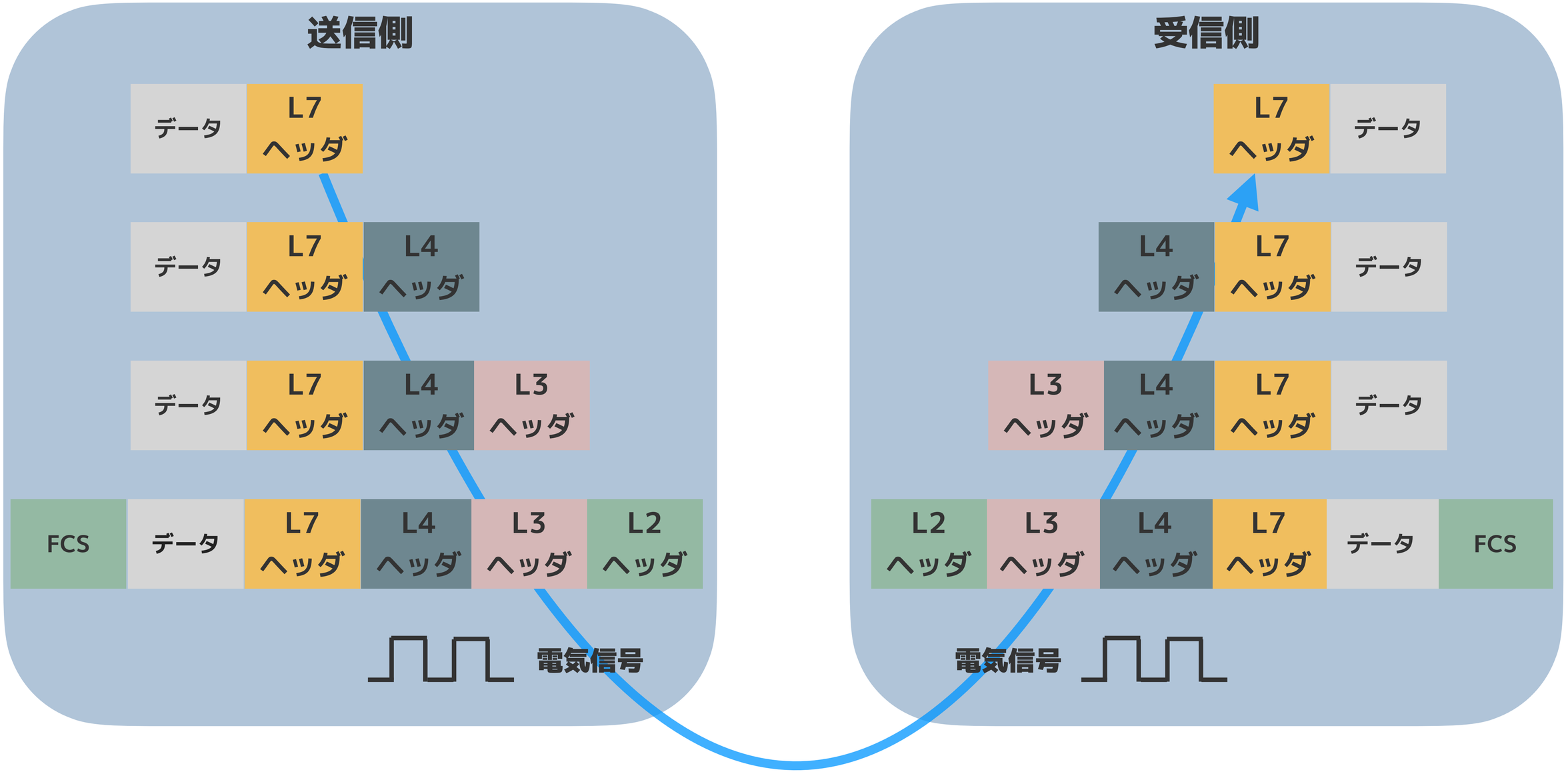


OSI参照モデル

コンピューターが通信するために利用するネットワークの機能を7つの階層に分類したもの。階層ごとに通信プロトコルが定義されている。

第7層 (L7)	アプリケーション層	アプリケーション間の通信 （HTTP、FTP）
第6層 (L6)	プレゼンテーション層	データ形式の変換（暗号化、圧縮）
第5層 (L5)	セッション層	通信の開始・維持・終了 （セッション管理）
第4層 (L4)	トランスポート層	エンドツーエンドの通信 （TCP/UDP、セグメント）
第3層 (L3)	ネットワーク層	ネットワーク間のルーティング （IPアドレス、パケット）
第2層 (L2)	データリンク層	隣接ノード間の通信 （MACアドレス、フレーム）
第1層 (L1)	物理層	電気信号やビットの伝送 （ケーブル、コネクタ）

OSI参照モデルとデータ(カプセル化/非カプセル化)



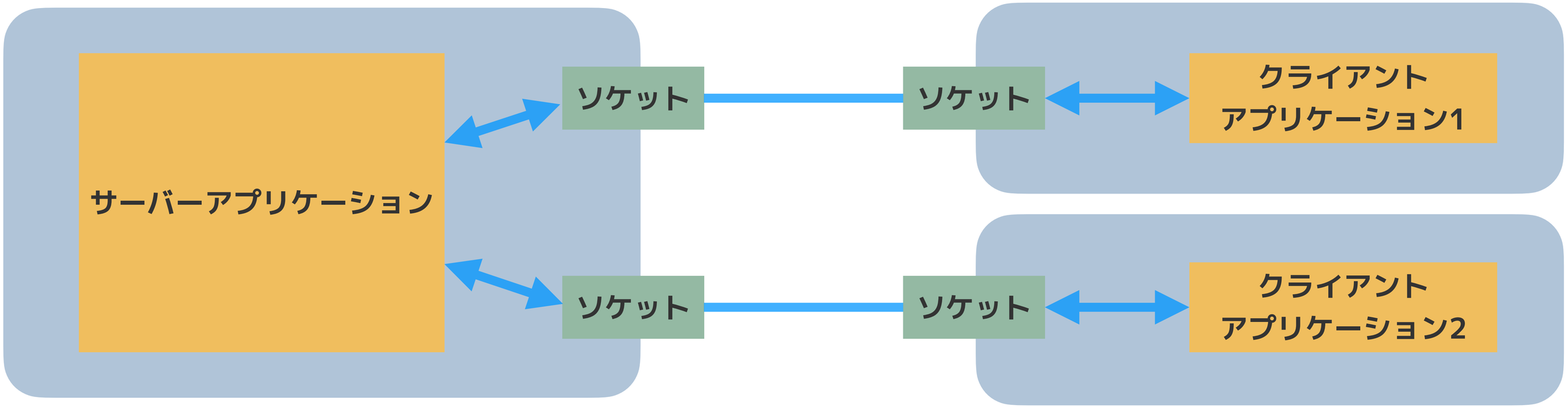
ざっくりとした理解でおk

ここからつくりかた

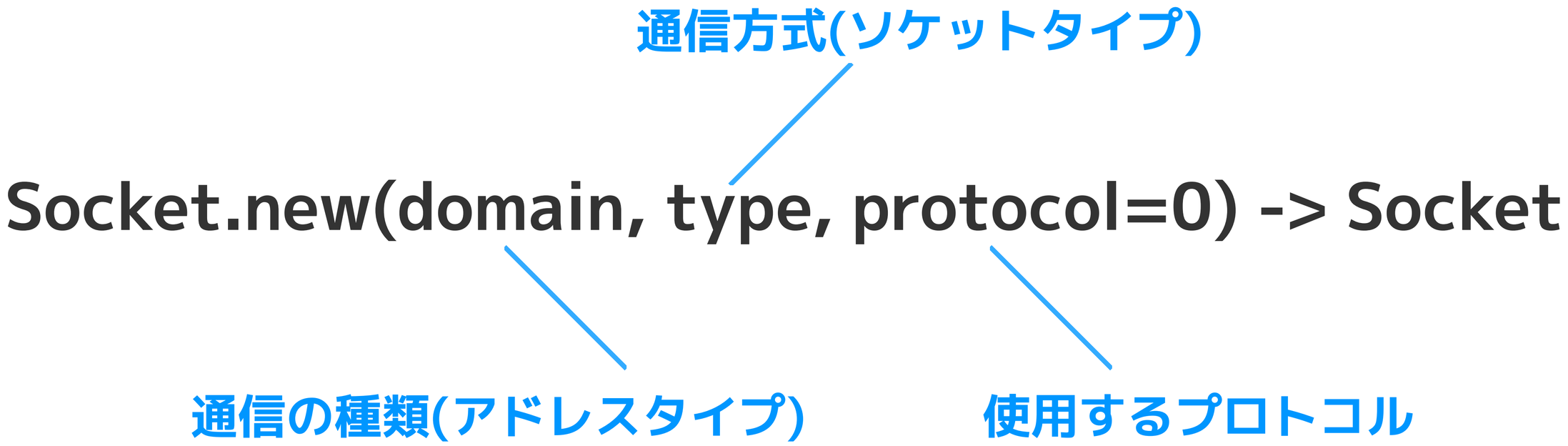
まずはソケットをつくる

ソケットとは

ネットワーク通信に使うインターフェース。アプリケーション間でデータの送受信を行うために、OSが提供するAPIで、通信のエンドポイントとして機能する。



ソケットをつくる



パケットキャプチャツールが必要とするソケット #トル

データリンク層のヘッダが扱えて
全てのプロトコルを受信できればおk

組み合わせはこんな感じ

扱いたいもの	ソケットの設定		
	アドレスタイプ	ソケットタイプ	プロトコル
UDPデータ部以上	AF_INET	SOCK_DGRAM	IPPROTO_UDP(17) or 0
TCPデータ部以上	AF_INET	SOCK_STREAM	IPPROTO_TCP(6) or 0
UDPヘッダ部以上	AF_INET	SOCK_RAW	IPPROTO_UDP(17)
TCPヘッダ部以上	AF_INET	SOCK_RAW	IPPROTO_TCP(6)
ICMPヘッダ部以上	AF_INET	SOCK_RAW	IPPROTO_ICMP(1)
IPヘッダ部以上	AF_PACKET	SOCK_DGRAM	htons(ETH_P_IP)
ARPヘッダ部以上	AF_PACKET	SOCK_DGRAM	htons(ETH_P_ARP)
Ethernetヘッダ部以上	AF_PACKET	SOCK_RAW	htons(ETH_P_IP)

組み合わせはこんな感じ

扱いたいもの	ソケットの設定		
	アドレスタイプ	ソケットタイプ	プロトコル
UDPデータ部以上	AF_INET	SOCK_DGRAM	IPPROTO_UDP(17) or 0
TCPデータ部以上	AF_INET	SOCK_STREAM	IPPROTO_TCP(6) or 0
UDPヘッダ部以上	AF_INET	SOCK_RAW	IPPROTO_UDP(17)
TCPヘッダ部以上	AF_INET	SOCK_RAW	IPPROTO_TCP(6)
ICMPヘッダ部以上	AF_INET	SOCK_RAW	IPPROTO_ICMP(1)
IPヘッダ部以上	AF_PACKET	SOCK_DGRAM	htons(ETH_P_IP)
ARPヘッダ部以上	AF_PACKET	SOCK_DGRAM	htons(ETH_P_ARP)
Ethernetヘッダ部以上	AF_PACKET	SOCK_RAW	htons(ETH_P_IP)

なのでこうする

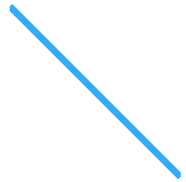
```
require 'socket'

ETH_P_ALL = 768 # NOTE: htons(ETH_P_ALL) ⇒ linux/if_ether.h
Socket.new(Socket::AF_PACKET, Socket::SOCK_RAW, ETH_P_ALL)
```

ソケットをインターフェースに
バインドする

バインドする

```
Socket#bind(my_sockaddr) -> 0
```



ソケットアドレス構造体を pack した文字列

こうする

```
PACKED_ETH_P_ALL = [ETH_P_ALL].pack('S').unpack1('S>')
s = Socket.new(Socket::AF_PACKET, Socket::SOCK_RAW, ETH_P_ALL)
sll = [Socket::AF_PACKET, PACKED_ETH_P_ALL, mr_ifindex]
s.bind(sll.pack('SS>a16'))
```

こうする

```
PACKED_ETH_P_ALL = [ETH_P_ALL].pack('S').unpack1('S>')
s = Socket.new(Socket::AF_PACKET, Socket::SOCK_RAW, ETH_P_ALL)
sll = [Socket::AF_PACKET, PACKED_ETH_P_ALL, mr_ifindex]
s.bind(sll.pack('SS>a16'))
```

ソケットを作って

こうする

```
PACKED_ETH_P_ALL = [ETH_P_ALL].pack('S').unpack1('S>')
s = Socket.new(Socket::AF_PACKET, Socket::SOCK_RAW, ETH_P_ALL)
sll = [Socket::AF_PACKET, PACKED_ETH_P_ALL, mr_ifindex]
s.bind(sll.pack('SS>a16'))
```

sockaddr_ll構造体に詰める

struct sockaddr_ll(1)

```
struct sockaddr_ll {
    unsigned short sll_family;   /* Always AF_PACKET */
    unsigned short sll_protocol; /* Physical-layer protocol */
    int            sll_ifindex;  /* Interface number */
    unsigned short sll_hatype;   /* ARP hardware type */
    unsigned char  sll_pkttype;  /* Packet type */
    unsigned char  sll_halen;    /* Length of address */
    unsigned char  sll_addr[8];  /* Physical-layer address */
};
```

(1) https://man7.org/linux/man-pages/man7/packet.7.html

struct sockaddr_ll(1)

```
struct sockaddr_ll {
    unsigned short sll_family;   /* Always AF_PACKET */
    unsigned short sll_protocol; /* Physical-layer protocol */
    int            sll_ifindex;  /* Interface number */
    unsigned short sll_hatype;   /* ARP hardware type */
    unsigned char  sll_pkttype;  /* Packet type */
    unsigned char  sll_halen;    /* Length of address */
    unsigned char  sll_addr[8];  /* Physical-layer address */
};
```

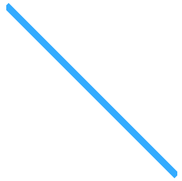
この3つに詰める

(1) https://man7.org/linux/man-pages/man7/packet.7.html

どうやって？

Cの構造体のように詰める方法

Array#pack(template) -> String



自身のバイナリとしてパックするためのテンプレート

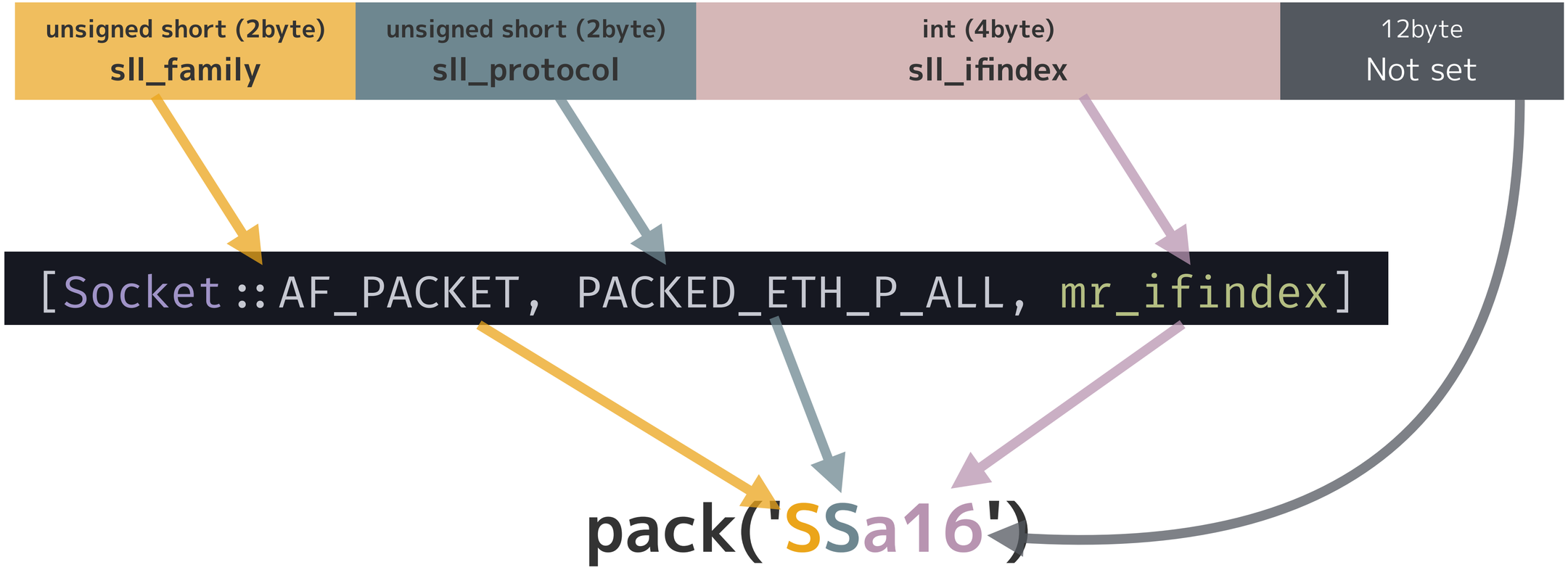
struct sockaddr_ll(1)

```
struct sockaddr_ll {
    unsigned short sll_family;   /* Always AF_PACKET */
    unsigned short sll_protocol; /* Physical-layer protocol */
    int            sll_ifindex;  /* Interface number */
    unsigned short sll_hatype;   /* ARP hardware type */
    unsigned char  sll_pkttype;  /* Packet type */
    unsigned char  sll_halen;    /* Length of address */
    unsigned char  sll_addr[8];  /* Physical-layer address */
};
```

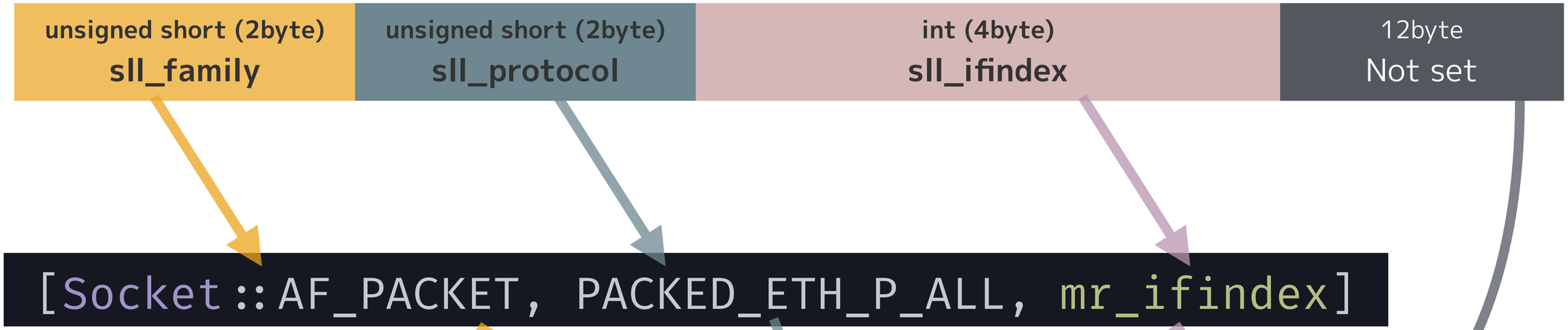
ushort, ushort, intの順

(1) https://man7.org/linux/man-pages/man7/packet.7.html

Cの構造体のように詰める方法



Cの構造体のように詰める方法



Arrayに設定したい値を詰める

Cの構造体のように詰める方法

バイナリとしてパックした文字列にする

```
[Socket::AF_PACKET, PACKED_ETH_P_ALL, mr_ifindex]
```

```
.pack('SSa16')
```

こうする

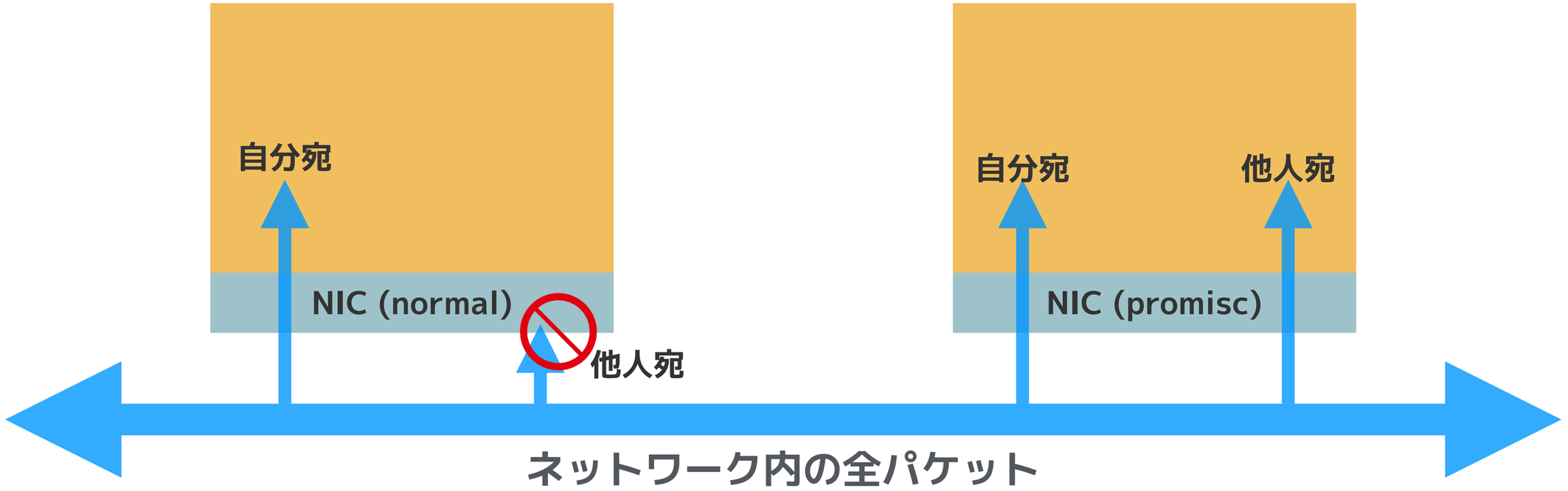
```
PACKED_ETH_P_ALL = [ETH_P_ALL].pack('S').unpack1('S>')
s = Socket.new(Socket::AF_PACKET, Socket::SOCK_RAW, ETH_P_ALL)
sll = [Socket::AF_PACKET, PACKED_ETH_P_ALL, mr_ifindex]
s.bind(sll.pack('SS>a16'))
```

ソケットに割り当てる

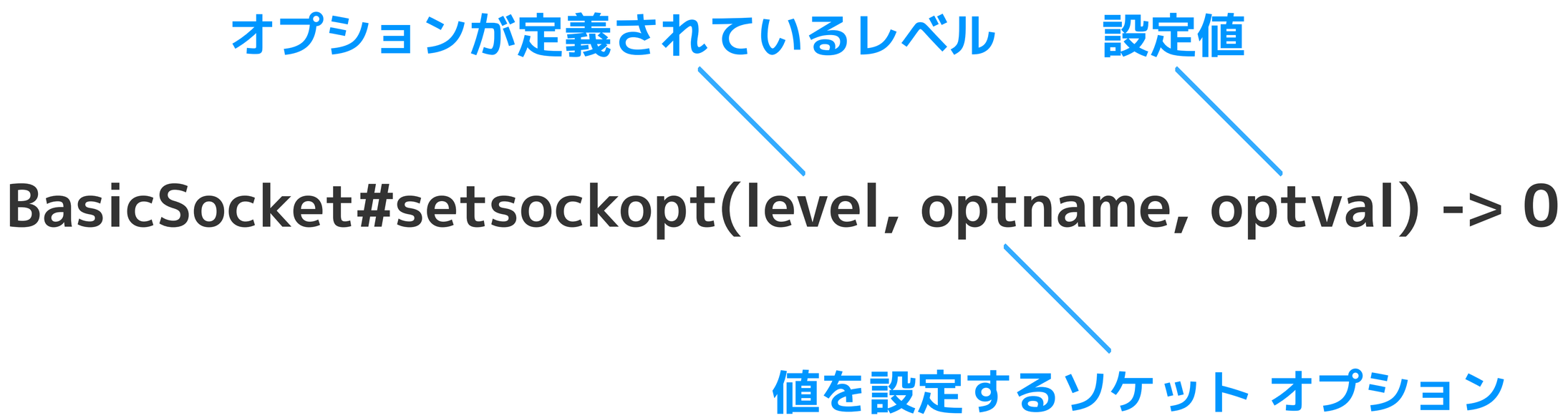
プロミスキャスモードを指定

プロミスキャスモードとは

ネットワークインターフェースの動作モードの一つで、電気的に受信したすべてのデータを読み込むモード。宛先が自分のMACアドレスになっていないフレームも受信する。



設定する



やり方

パケットソケットのオプション　バインドを追加

```
s.setsockopt(SOL_PACKET, PACKET_ADD_MEMBERSHIP, mq_req)
```

プロミスキャスモードの設定を渡す

やり方

パケットソケットのオプション　バインドを追加

```
s.setsockopt(SOL_PACKET, PACKET_ADD_MEMBERSHIP, mq_req)
```

プロミスキャスモードの設定を渡す

packet_mreq構造体に詰めて渡す

struct packet_mreq(1)

```
struct packet_mreq {
  int             mr_ifindex;    /* interface index */
  unsigned short  mr_type;       /* action */
  unsigned short  mr_alen;       /* address length */
  unsigned char   mr_address[8]; /* physical-layer address */
};
```

(1) https://man7.org/linux/man-pages/man7/packet.7.html

それぞれはこう詰める

```ruby
def mr_ifindex
  i = Socket.getifaddrs.find { |ifaddr| ifaddr.name == @ifname }&.ifindex
  [[i].pack('c')].pack('a4')
end

def mr_type
  PACKET_MR_PROMISC = 0x0001 # NOTE: netpacket/packet.h
  [PACKET_MR_PROMISC].pack('S')
end

def mr_alen
  [0].pack('S')
end

def mr_address
  [0].pack('C') * 8
end
```

それぞれはこう詰める

```
def mr_ifindex
  i = Socket.getifaddrs.find { |ifaddr| ifaddr.name == @ifname }&.ifindex
  [[i].pack('c')].pack('a4')
end

def mr_type
  PACKET_MR_PROMISC = 0x0001 # NOTE: netpacket/packet.h
  [PACKET_MR_PROMISC].pack('S')
end

def mr_alen
  [0].pack('S')
end

def mr_address
  [0].pack('C') * 8
end
```

Socket.getifaddrs でI/F名から index を取得する

それぞれはこう詰める

```
def mr_ifindex
  i = Socket.getifaddrs.find { |ifaddr| ifaddr.name == @ifname }&.ifindex
  [[i].pack('c')].pack('a4')
end

def mr_type
  PACKET_MR_PROMISC = 0x0001 # NOTE: netpacket/packet.h
  [PACKET_MR_PROMISC].pack('S')
end

def mr_alen
  [0].pack('S')
end

def mr_address
  [0].pack('C') * 8
end
```

PACKET_MR_PROMISC を設定

それぞれはこう詰める

```
def mr_ifindex
  i = Socket.getifaddrs.find { |ifaddr| ifaddr.name == @ifname }&.ifindex
  [[i].pack('c')].pack('a4')
end

def mr_type
  PACKET_MR_PROMISC = 0×0001 # NOTE: netpacket/packet.h
  [PACKET_MR_PROMISC].pack('S')
end

def mr_alen
  [0].pack('S')
end

def mr_address
  [0].pack('C') * 8
end
```

mr_alen と mr_address は0埋め

ソケットは出来たので

パケットの受信

パケットを受信する

フラグ

```
Socket#recvfrom(maxlen, flags=0) -> [String, Addrinfo]
```

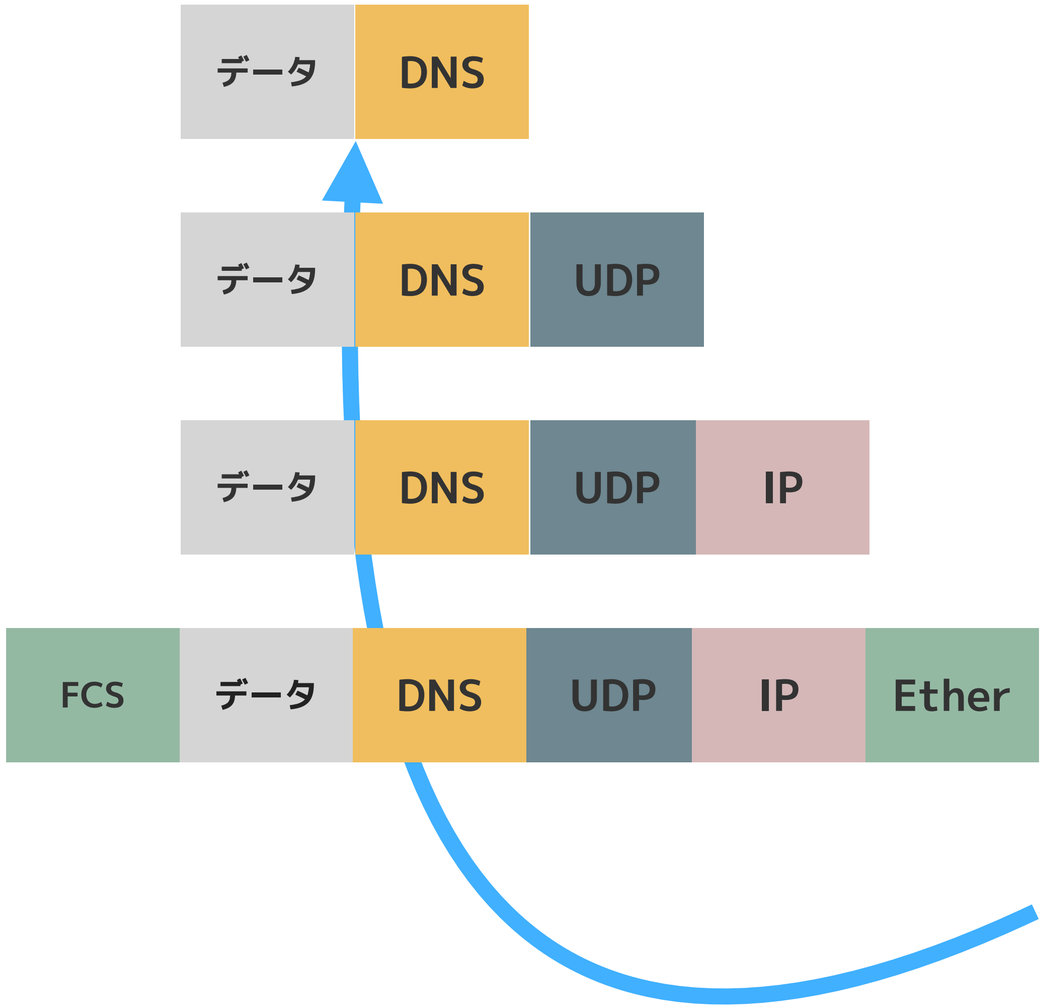
ソケットから受けとるデータの最大値

ループを回しつつ受信してアナライズしていく

```
loop do
  msg, = s.recvfrom(2084)
  analyze(msg)
end
```

ヘッダを解析していく

ヘッダを解析していく例：UDPぐらいまで



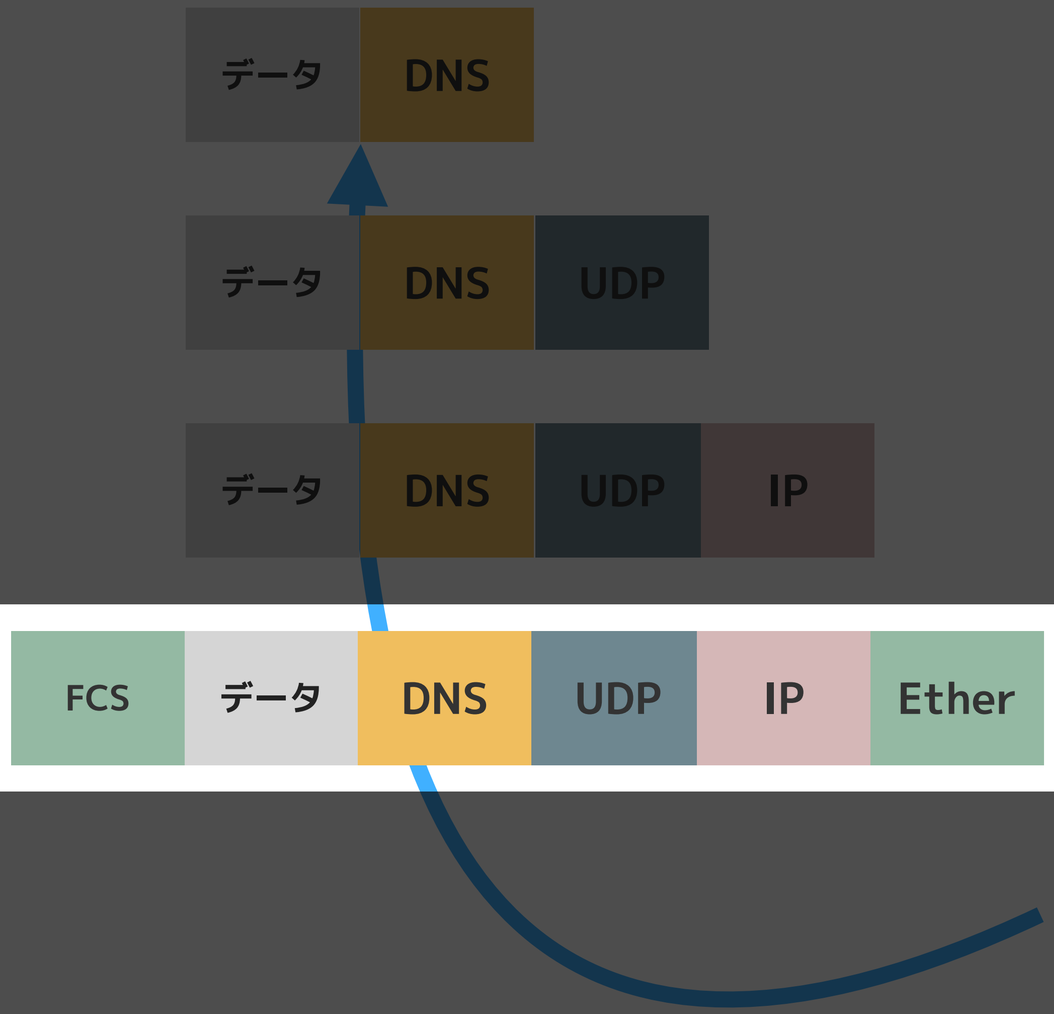
L7: メッセージ

L4: セグメント/データグラム

L3: パケット

L2: フレーム

ヘッダを解析していく例：UDPぐらいまで



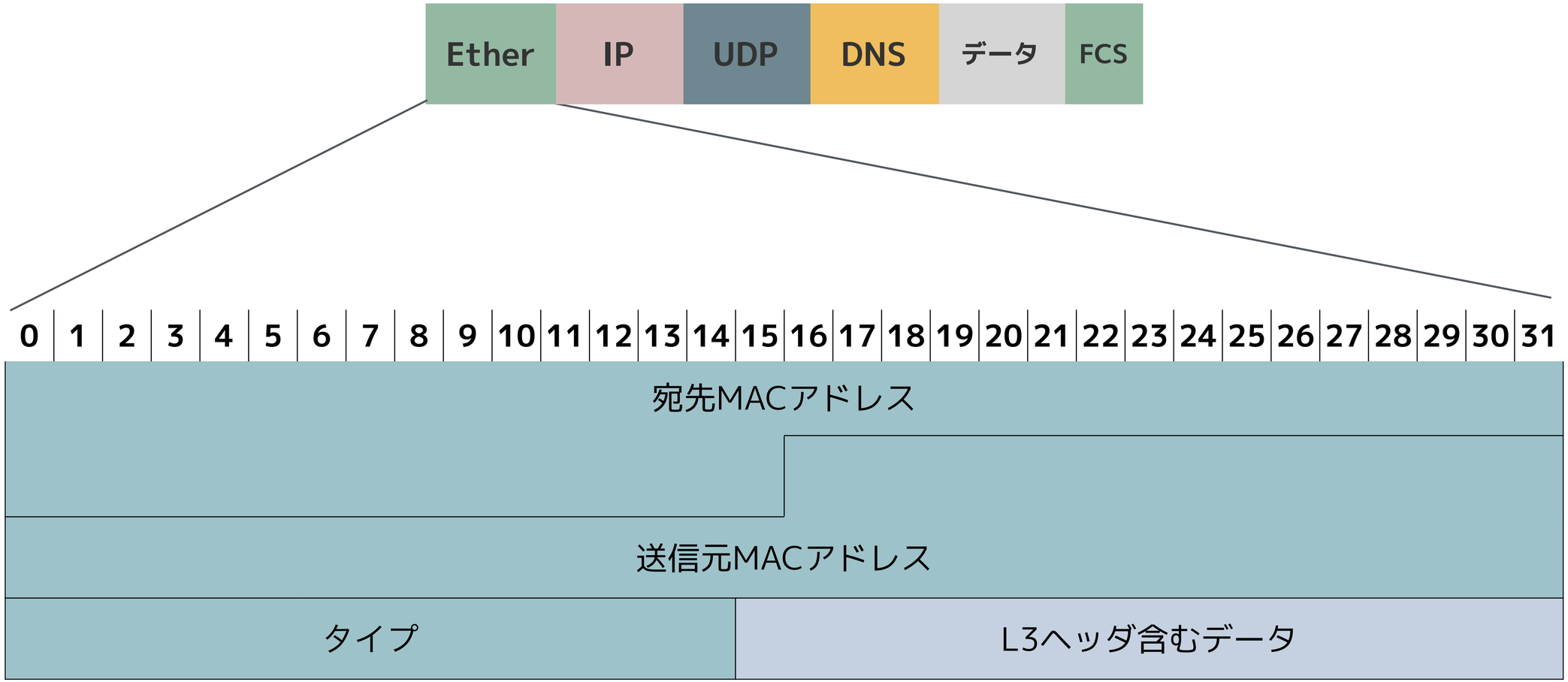
L7: メッセージ

L4: セグメント/データグラム

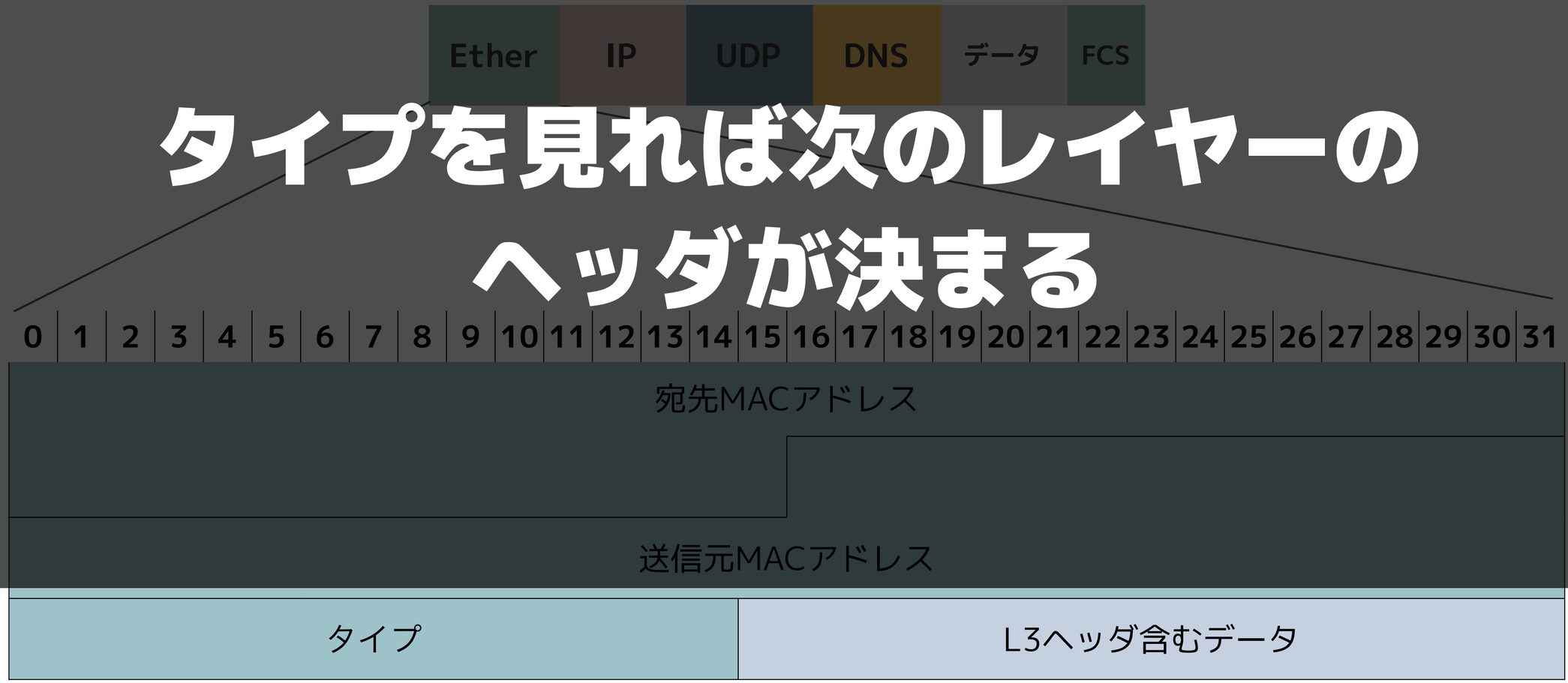
L3: パケット

L2: フレーム

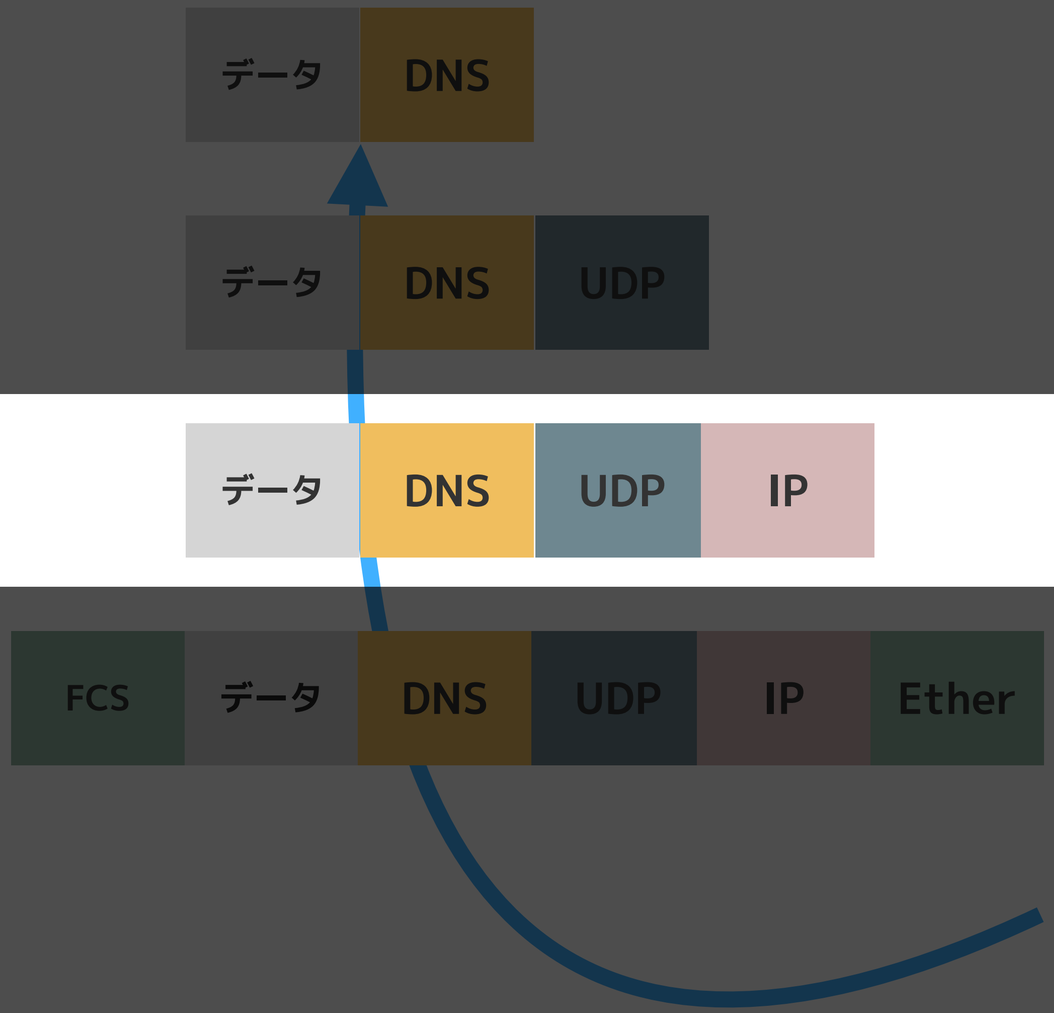
イーサネットIIヘッダ



イーサネットIIヘッダ



ヘッダを剥がしつつ出力するだけ



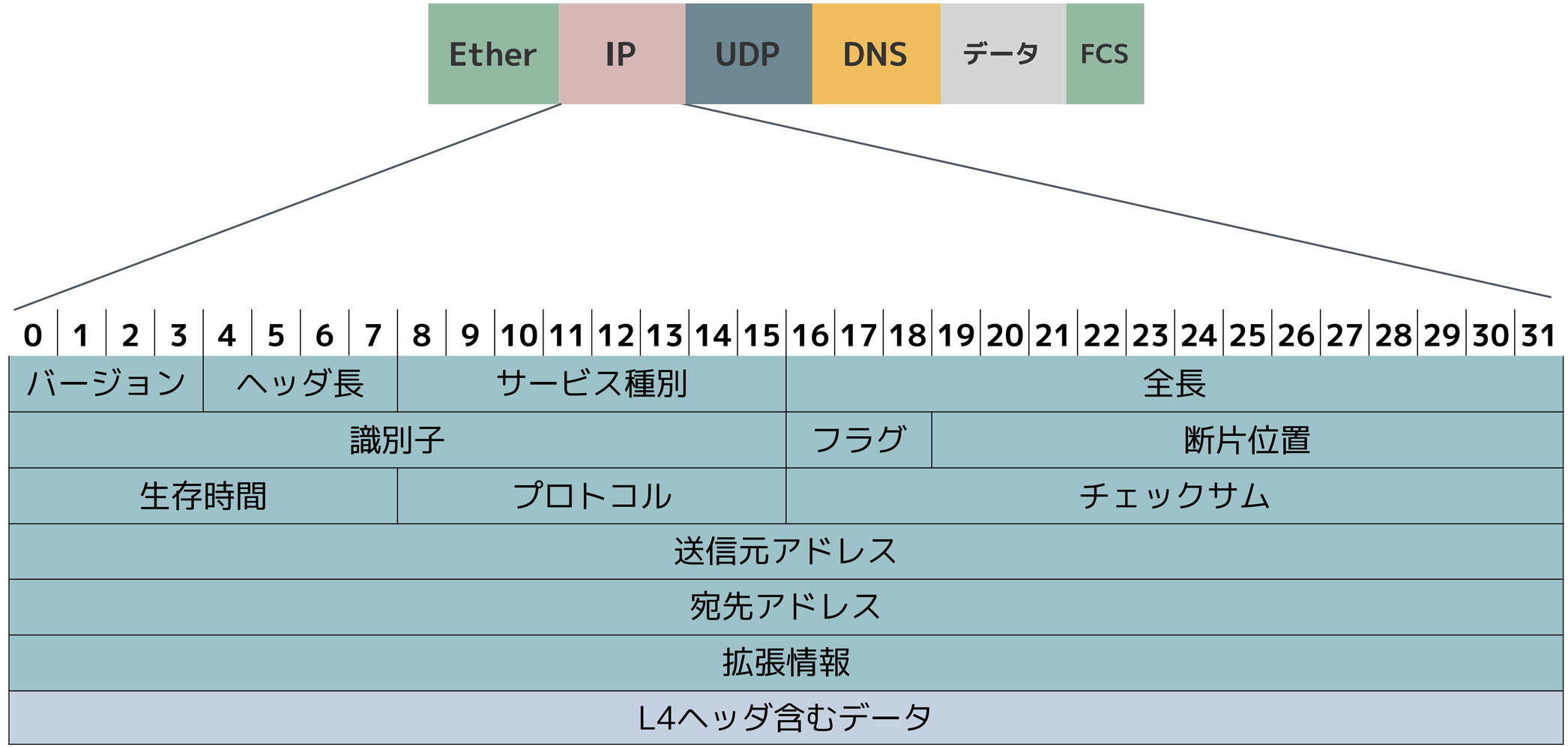
L7: メッセージ

L4: セグメント/データグラム

L3: パケット

L2: フレーム

IPv4ヘッダ



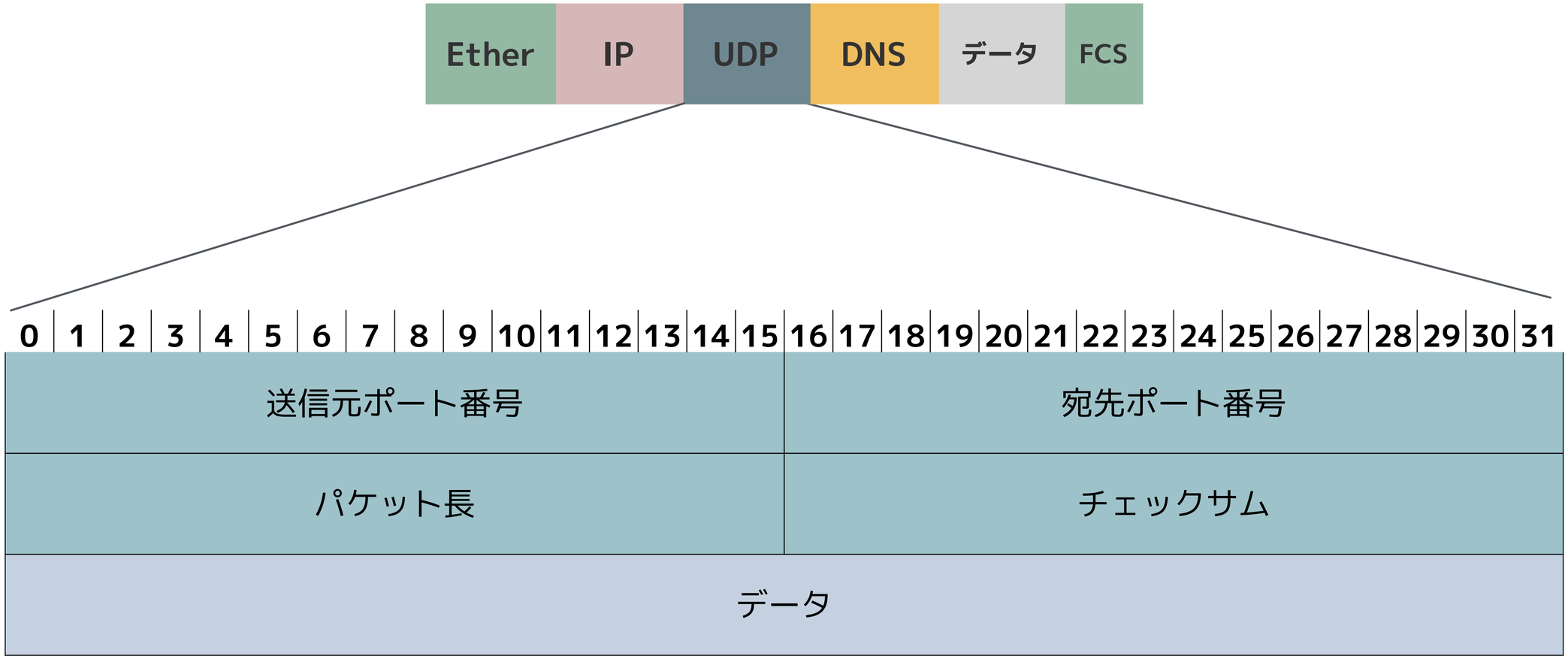
IPv4ヘッダ

プロトコルを見れば次のレイヤーのヘッダが決まる

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
バージョン				ヘッダ長				サービス種別								全長															
識別子																フラグ			断片位置											
生存時間								プロトコル								チェックサム															
送信元アドレス																															
宛先アドレス																															
拡張情報																															
L4ヘッダ含むデータ																															

ヘッダを剥がしつつ出力するだけ

UDPヘッダ



あとはサポートするヘッダ用の解析処理を書いていけば…

完成！！！！！

おわりにかえて

- ソケット作って、I/Fへバインドして、プロミスキャスモードを設定して、あとはループ内で受信して解析していくだけでおk
- 解析も基本はヘッダをレイヤーごとに解析していけばおk
- Macでは動作しないんや…すまんな…
- Wiresharkで見れるpcap形式での出力も実装はある！が、発表時間はない…
- みんなもやろうネットワークプログラミング！