SAKURA internet

GoでEPC作って本番運用している話

JANOG42 Meeting in Mie
発表日時:7月12日(木) 15:05～15:35(30分)
発表会場:多目的ホール

DAY	COMPANY	DEPARTMENT	NAME
2018/7/12（木）	さくらインターネット株式会社	IoTチーム	日下部雄也

日下部 雄也

https://twitter.com/higebu

https://github.com/higebu

- さくらインターネット2年目（2社目）
- さくらのセキュアモバイルコネクト
  - HSS/PGWの開発
  - SIMのプロファイル作成
- 好きなルータ
  - VyOS
- 好きな言語
  - Go

この話をしようと思った理由

SAKURA internet

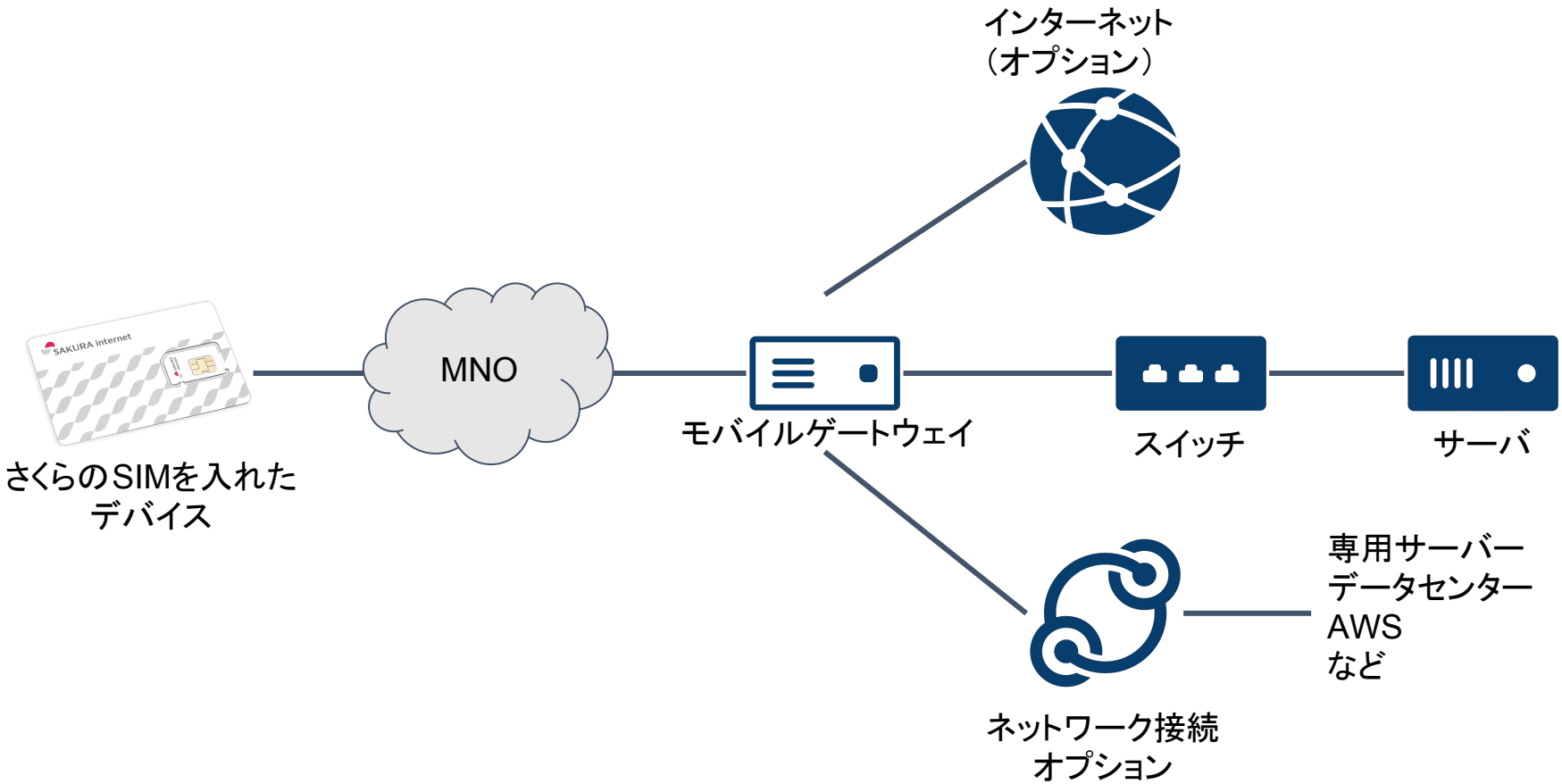
- EPCの実装について話せる仲間が欲しい
- うちではこうしていますみたいな話が聞きたい
- EPCを実装する人が増えて欲しい
- 5GCではControl PlaneがOpenAPI 3.0になるらしいので、作る人増えそう

アジェンダ

- さくらのセキュアモバイルコネクトとは
- なぜ独自にHSS、PGWを作ったのか
- なぜGoで作ったのか
- 基本的なLTEのアーキテクチャ
- HSSの基本
- Goを使ったHSSの作り方
- PGWの基本
- Goを使ったPGWの作り方
- まとめ

- お客様のデバイスから**インターネットを経由しない**
  閉域網でさくらのクラウドと通信できるサービス
- SIM1枚当たり**12円/月**
- 何回でもSIMを登録解除可能で、解除中は**0円**
- 詳しくは https://www.sakura.ad.jp/services/sim/

さくらのセキュアモバイルコネクトとは



なぜ独自にHSS、PGWを作ったのか

なぜ独自にHSS、PGWを作るのか

- ~~面白いから~~
- コスト削減
  - ○億円削減
- 機能追加の自由度
  - SIMルートとか
  - 今後も便利な機能を作っていく予定
- アンコントローラブルな箇所の削減
  - 性能、品質を自分たちでコントロールできる
  - 何かあったとき自分たちで対応できる
- 今までの常識を壊したかった

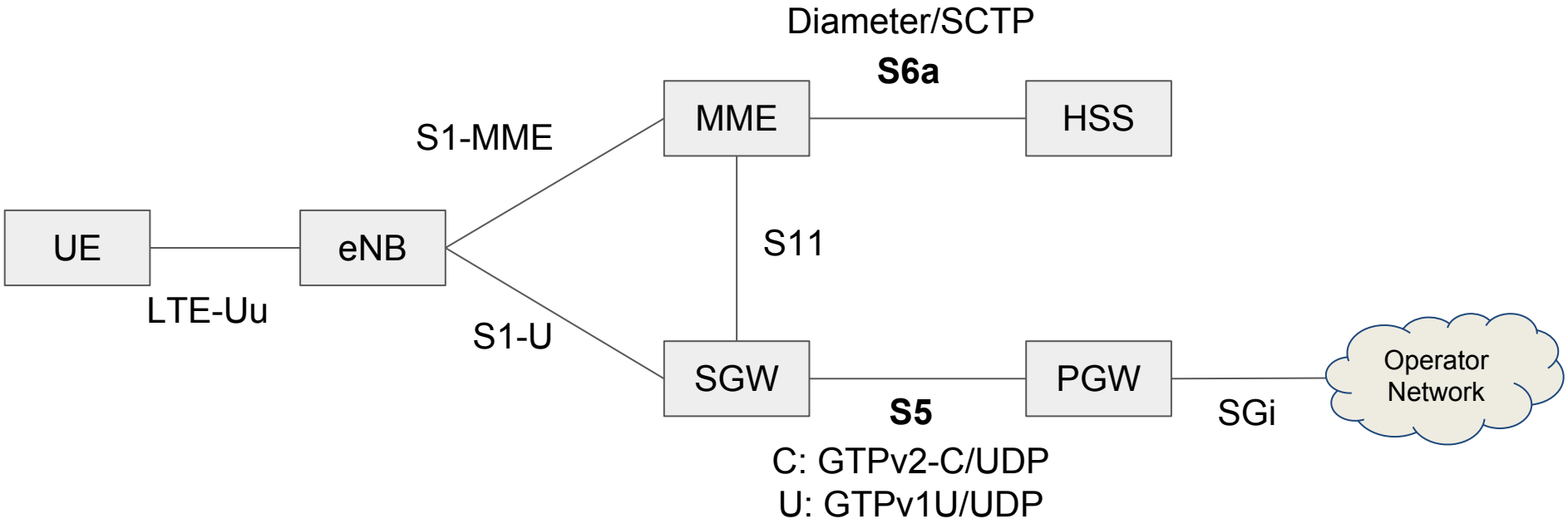
なぜGoで作ったのか

なぜGoで作ったのか

- ~~趣味~~
- シンプルな言語仕様
  - ネットワーク強いけどプログラミング初心者な人とかでもすぐに書けるようになる
- gofmtによる自動フォーマット
- それなりに高速
  - go bench でベンチマークしやすい
- 豊富なサンプル
  - Go自体、github.com/miekg/dns 、github.com/osrg/gobgp など

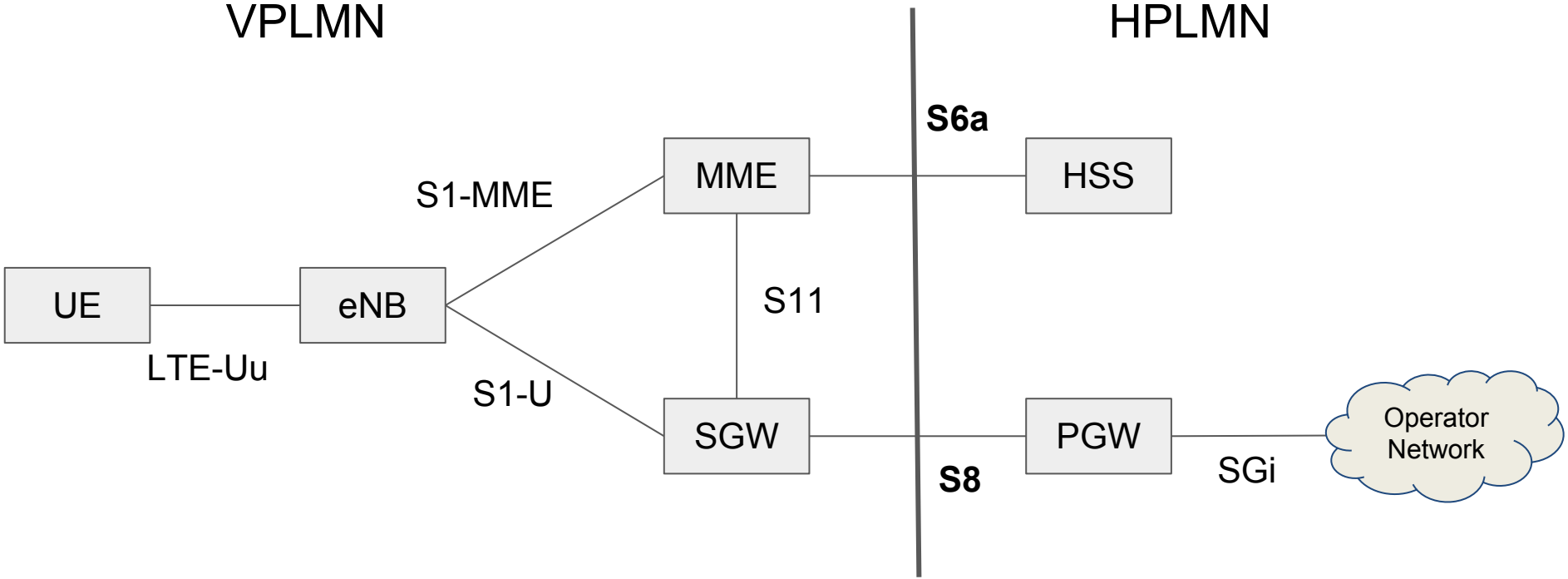
基本的なLTEのアーキテクチャ

基本的なLTEのアーキテクチャ



詳細は 3GPP TS 23.401 参照

基本的なLTEのアーキテクチャ - ローミング



詳細は 3GPP TS 23.401 と GSMA IR.88 参照

- 例）Attach procedure
- 3GPP 23.401 5.3.2.1
  E-UTRAN Initial Attach
- こういう図がたくさんあるので、
  これで流れを把握する

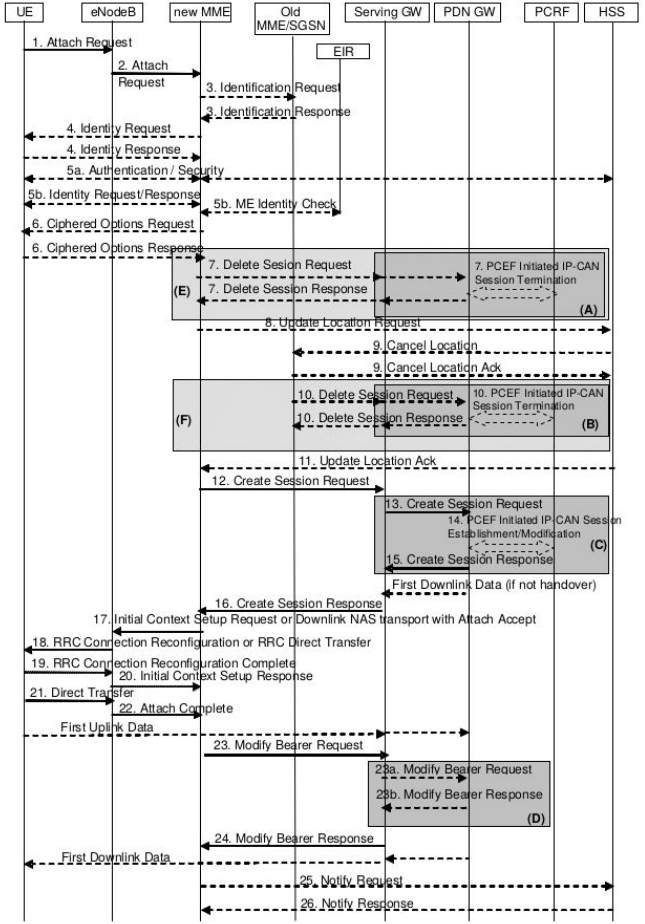


Figure 5.3.2.1-1: Attach procedure

HSSの基本

- 3GPP TS 23.401 5.1.1.9 参照
- SCTP(Stream Control Transmission Protocol)
  - RFC 4960
  - port 3868
- Diameter
  - RFC 3588
  - Diameterプロトコルガイド（本）

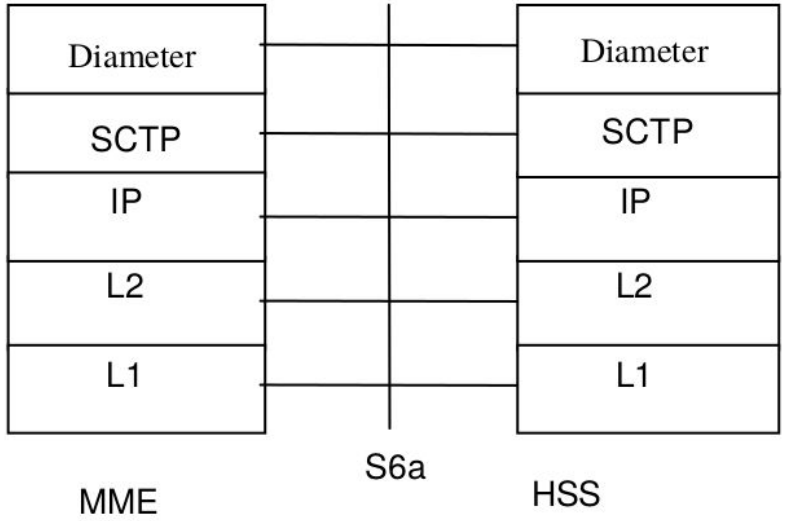


Figure 5.1.1.9-1: Control Plane for S6a interface
3GPP TS 23.401

HSSの基本 - 保持しないといけないデータ

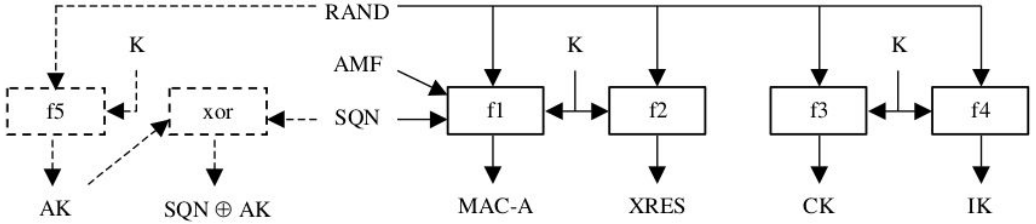
- 3GPP TS 23.401 5.7.1 参照
- 5.7は Information Storage という章で、それぞれのコンポーネントが何を保持しないといけないか書いてある
- IMSI、MME Identity、APNなど
- ここには書いていないが、AuCも実装しないといけないので、Rand、Kiなども保存する必要がある
  - AuCは認証に使う
  - LTEのセキュリティについて詳しくは 3GPP TS 33.401 参照

- 全体の流れが23.401、動作の詳細は29.272に書いてある
- 3GPP TS 23.401 の下記の章参照
  - 5.3 Authentication, security and location management
  - 5.4 Session Management, QoS and interaction with PCC functionality
  - 5.5 Handover
- 3GPP TS 29.272 の下記の章参照
  - 5 MME – HSS (S6a) and SGSN – HSS (S6d)
  - 7 Protocol Specification and Implementation
  - ここからいろいろなドキュメントに飛ばされる。。。

- 少なくとも下記のCommandに対応する必要がある
  - Authentication Information
    - Initial Attach
  - Update Location
    - Initial Attach
  - Cancel Location
    - HSSからのDetach
  - Notify
    - MMEやSGWの変更時など
  - Purge UE
    - MME上のSubcription-Dataが消えるときに来る

- Authentication InformationではAuCを作ってSIMの認証をする必要がある
  - 鍵交換アルゴリズムはMilenage（3GPP TS 35.205）
  - MilenageはFreeBSDに実装されていて参考になる
    - https://github.com/freebsd/freebsd/blob/master/contrib/wpa/src/crypto/milenage.c

7.2 Use of the algorithms on the AuC side



This figure describes the generation of the authentication and key generation values in the HLR/AuC

Goを使ったHSSの作り方

- Linuxはわりと古くからSCTP使える
  - 詳しくは man sctp
  - one-to-many と one-to-one があり、one-to-one の方はTCPと同様のAPIで使える
  - one-to-many できない環境なので one-to-one を採用
  - SCTPにはいろいろなパラメータがあるが、GSMA IR.88におすすめ設定が載っている
- Go本体はSCTPに対応していない
  - Go自体に機能追加するか、github.com/ishidawataru/sctp を使う

- さくらではGo自体に機能追加した

```
ln, err := net.Listen("sctp", ":3868")
if err != nil {
     // handle error
}
```

- Go1.9.1時点でのソース は公開している
- GoへのPR: proposal: x/net/sctp: new package #22191
  - Go本体に追加しようとしたが、SCTPはマイナーなので golang.org/x/net にしてと言われ、さらに他のIssueの対応待ちで保留になっている

- github.com/fiorix/go-diameter を使っている
- わりとメンテされているし、動く
- XMLでCommand、AVPの定義を書くスタイル（つらい）
- 2018/2/6に Adding Go-Diameter SCTP support and 3GPP S6a protocol support #78 というPRがマージされている
  - github.com/ishidawataru/sctp が使われている
  - さくらでは未検証
  - 試してみたい気持ちはあるが、Purge-UEなど足りないCommandがある

- 正しいパケットが作れているかの確認はWiresharkでやる

```
▼ Diameter Protocol
    Version: 0x01
    Length: 748
  ▶ Flags: 0x40, Proxyable
    Command Code: 316 3GPP-Update-Location
    ApplicationId: 3GPP S6a/S6d (16777251)
    Hop-by-Hop Identifier: 0x046ab000
    End-to-End Identifier: 0x2f72744d
    [Request In: 137]
    [Response Time: 0.006810000 seconds]
  ▶ AVP: Result-Code(268) l=12 f=-M- val=DIAMETER_SUCCESS (2001)
  ▶ AVP: Session-Id(263) l=74 f=-M- val=[redacted]
  ▶ AVP: Origin-Host(264) l=51 f=-M- val=[redacted].3gppnetwork.org
  ▶ AVP: Origin-Realm(296) l=41 f=-M- val=epc.[redacted].3gppnetwork.org
  ▶ AVP: Destination-Realm(283) l=41 f=-M- val=epc.mnc020.mcc440.3gppnetwork.org
  ▶ AVP: Destination-Host(293) l=55 f=-M- val=[redacted].S6a.epc.mnc020.mcc440.3gppnetwork.org
  ▶ AVP: Auth-Session-State(277) l=12 f=-M- val=NO_STATE_MAINTAINED (1)
  ▶ AVP: ULA-Flags(1406) l=16 f=VM- vnd=TGPP val=1
  ▼ AVP: Subscription-Data(1400) l=360 f=VM- vnd=TGPP
      AVP Code: 1400 Subscription-Data
    ▶ AVP Flags: 0xc0
      AVP Length: 360
      AVP Vendor Id: 3GPP (10415)
    ▼ Subscription-Data: 00000590c0000010000028af00000000000002bdc0000012...
        AVP: Subscriber-Status(1424) l=16 f=VM- vnd=TGPP val=SERVICE_GRANTED (0)
      ▶ AVP: MSISDN(701) l=18 f=VM- vnd=TGPP val=580200111111
      ▶ AVP: Network-Access-Mode(1417) l=16 f=VM- vnd=TGPP val=ONLY_PACKET (2)
      ▶ AVP: APN-Configuration-Profile(1429) l=252 f=VM- vnd=TGPP
      ▶ AVP: AMBR(1435) l=44 f=VM- vnd=TGPP
  ▶ AVP: Supported-Features(628) l=56 f=V-- vnd=TGPP
```

- 正しいパケットだからと言って動くとは限らない
- ianaの定義と被っているResult Code
- 足りないAVP
- Mbit
- 相手の運用上指定できない数値
- なぜか通らないAVP

PGWの基本

- 概要は 3GPP TS 23.401 5.1.1.6
- 詳細は 3GPP TS 29.274
- GTPv2-C (2123/UDP)

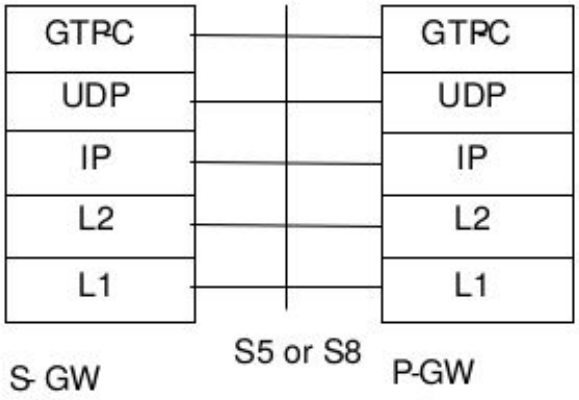


Figure 5.1.1.6-1: Control Plane for S5 and S8 interfaces
3GPP TS 23.401

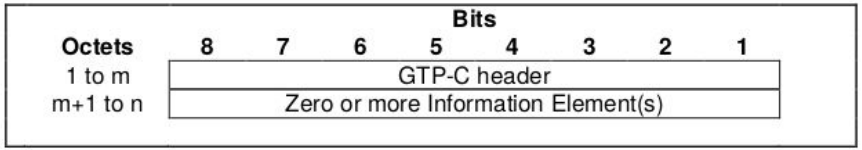


Figure 5.6-1: GTP-C Header followed by subsequent Information Elements

3GPP TS 29.274

SAKURA internet

- 概要は 3GPP TS 23.401 5.2.1.1
- 詳細は 3GPP TS 29.060
- GTPv1-U (2152/UDP)

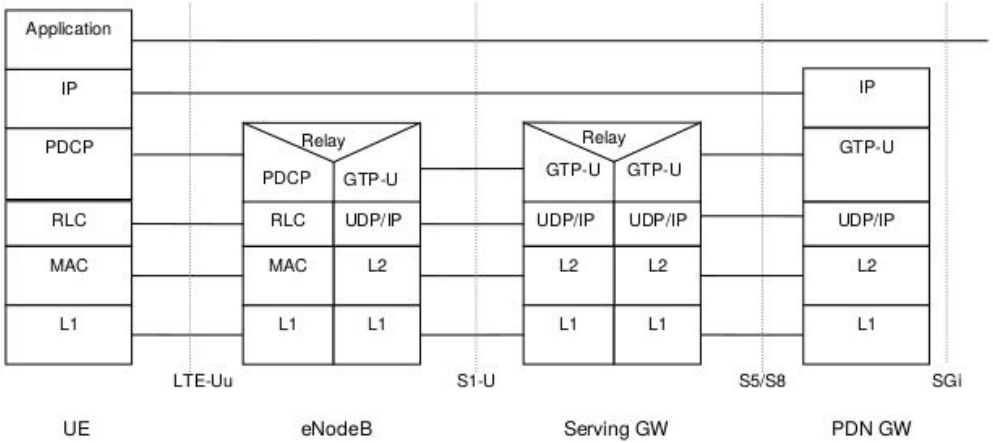


Figure 5.1.2.1-1: User Plane
3GPP TS 23.401

Octets	Bits 8	7	6	5	4	3	2	1
1	Version			PT	(*)	E	S	PN
2	Message Type							
3	Length (1st Octet)							
4	Length (2nd Octet)							
5	Tunnel Endpoint Identifier (1st Octet)							
6	Tunnel Endpoint Identifier (2nd Octet)							
7	Tunnel Endpoint Identifier (3rd Octet)							
8	Tunnel Endpoint Identifier (4th Octet)							
9	Sequence Number (1st Octet)[1) 4)]							
10	Sequence Number (2nd Octet)[1) 4)]							
11	N-PDU Number[2) 4)]							
12	Next Extension Header Type[3) 4)]							

Figure 2: Outline of the GTP Header
3GPP TS 29.060

PGWの基本 - 保持しないといけないデータ

- リストは 3GPP TS 23.401 5.7.4
- どう使うかは 3GPP TS 29.274
- IMSI、IMEI、IPアドレスなど

- 全体の流れが23.401、動作の詳細は 29.274 に書いてある
- 3GPP TS 23.401 の下記の章参照
  - 5.3 Authentication, security and location management
  - 5.4 Session Management, QoS and interaction with PCC functionality
  - 5.5 Handover
- 3GPP TS 29.274 の全部参照
  - ここからいろいろなドキュメントに飛ばされる。。。

- Control Planeは下記の5つを実装しておけば良いはず
  - Echo Request/Response
    - SGWとの相互監視のため
  - Create Session Request/Response
    - Attachのため
  - Modify Bearer Request/Response
    - SGWの変更を伴うハンドオーバーのため
  - Delete Session Request/Response
    - Dettachのため
  - Delete Bearer Request/Response
    - PGWからベアラを削除するため

- User PlaneはGTPv1-Uの下記のMessage Typeに対応する必要がある
  - Echo
    - SGWとの相互監視のため
  - G-PDU
    - UEからのパケット、UEへのパケットを通すため

Goを使ったPGWの作り方

- GoでGTPv2-Cを実装しているOSSを見たことはない
- 普通にUDPサーバ、クライアントを実装すれば良い
- golang.org/pkg/net で十分
- とにかくたくさんのIE（Information Element）のパーサを書かないといけない（つらい）

SAKURA internet

- 正しいパケットを作れているかはWiresharkで確認

```
▸ User Datagram Protocol, Src Port: 2123, Dst Port: 2123
▾ GPRS Tunneling Protocol V2
  ▸ Flags: 0x48
    Message Type: Create Session Response (33)
    Message Length: 111
    Tunnel Endpoint Identifier: 0x80592c20 (2153327648)
    Sequence Number: 0x0017c805 (1558533)
    Spare: 0
  ▸ PDN Address Allocation (PAA) :
  ▸ Bearer Context : [Grouped IE]
  ▸ APN Restriction : value 0
  ▸ Recovery (Restart Counter) : 12
  ▸ Cause : Request accepted (16)
  ▸ Fully Qualified Tunnel Endpoint Identifier (F-TEID) : S5/S8 PGW GTP-C interface, TEID/GRE Key: 0x6aefe60f, IPv4 [redacted]
  ▸ Protocol Configuration Options (PCO) :
    [Response To: 1]
    [Response Time: 0.005785000 seconds]
```

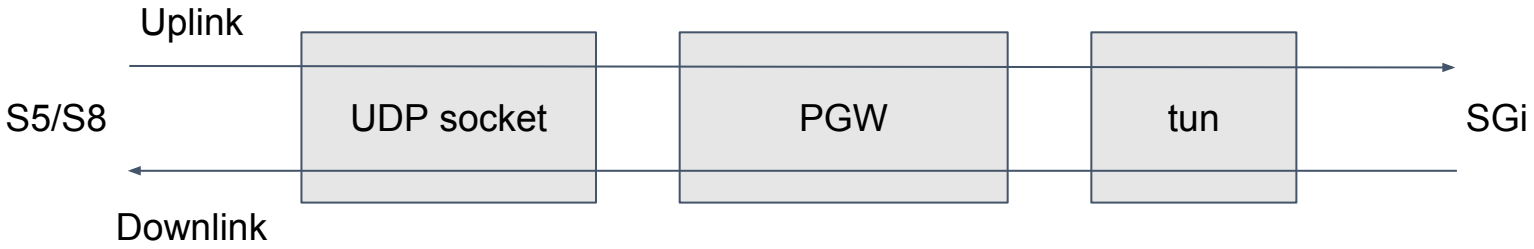
- 正しいパケットだからと言って動くとは限らない2
- 足りないIE
- Instance
- PCO、3GPP TS 24.008に書いてあるけど

- Linux kernel 4.7以降のgtpモジュールを使うか、tunインターフェースを作って実装するのが簡単
- 現在はtunインターフェース使う方で動いている
- 最近、github.com/google/gopacket にGTPのパーサが追加された
  - New protocol GTP #417
  - 使ってはいないけど参考になるのでは

- Linux kernel 4.7以降のgtpモジュールを使う場合
- netlink経由でgtpトンネルを作ることができる
- github.com/vishvananda/netlink を使うのが簡単
  - 2017/5/7から使えるようになっている
    - Add support for GPRS Tunnelling Protocol(GTP) #229
    - サンプルコード
  - さくらではこれより前から実装していたので、gtpモジュールの元になった libgtpnl を参考に実装した
  - netlinkのデバッグは大変だが nltrace が便利だった

- tunインターフェースを使う場合（流れ）
- Uplink
  - SGWから2152/UDPに来たGTPv1-Uのパケットのペイロードをtunインターフェースに入れる
- Downlink
  - tunインターフェースから読み込んだパケットをGTPv1-Uでカプセリングして SGWに投げる

- tunインターフェースを使う場合（実装）
- SGW方面はただのUDPサーバ、クライアント
- github.com/vishvananda/netlink でtunインターフェースを作って読み書きする



- tunインターフェースを使う場合（実装）
- github.com/vishvananda/netlink でtunインターフェース

```
link := &netlink.Tuntap{
      LinkAttrs: netlink.LinkAttrs{Name: "gtp0"},
      Mode:      netlink.TUNTAP_MODE_TUN,
      Flags:     netlink.TUNTAP_DEFAULTS | netlink.TUNTAP_NO_PI,
      Queues:    1,
}
if err := netlink.LinkAdd(link); err != nil {
      // handle error
}
if err := netlink.LinkSetUp(link); err != nil {
      // handle error
}
```

- link.Fds が []*os.File になっているので、link.Fds[0] をRead/Writeする

- 性能について
- 高負荷な状況ではコンテキストスイッチが問題になる
- sendmmsg/recvmmsg 使うとppsが改善できる
  - 詳しくはCloudflare先生の How to receive a million packets per second
  - Goだと golang.org/x/net/ipv4 でできる
    - サンプルコード
  - GTPのルータで使っている
  - モバイルゲートウェイではtunインターフェース側がボトルネックになる
- 今のところそこまで困っていない

- 参考: さくらのクラウド上での性能測定結果
  - CPU 20 core、224 GB RAM、10Gbps
  - 2台のサーバ間でgtpトンネルを作ってiperf（iperf 3.2）
- UDP（32 byteのパケット）
  - gtpモジュール: ~150Kpps
  - Go実装: ~70Kpps
- TCP
  - gtpモジュール: ~3Gbps
  - Go実装: ~1.5Gbps

- 1台で数Gbps以上出す必要が出てきたら試さないといけないかなと思っているものたち
  - 独自カーネルモジュール
  - DPDK
  - XDP
  - FD.io
- 他に良さそうなものがあったら教えてください

まとめ

まとめ

- EPCは案外作れるので皆さんも作りましょう
- 作っている方がいたら声をかけてください

参考資料

- RFC
  - RFC 4960 SCTP
  - RFC 3588 Diamter
- 3GPP
  - 3GPP TS 23.401
  - 3GPP TS 24.008
  - 3GPP TS 29.272
  - 3GPP TS 29.274
  - 3GPP TS 33.401
  - 3GPP TS 35.205
- GSMA
  - GSMA IR.88
- Go
  - github.com/ishidawataru/sctp
  - github.com/fiorix/go-diameter
  - github.com/vishvananda/netlink
  - golang.org/x/net/ipv4
- C
  - Milenage
  - libgtpnl
  - nltrace

おまけ: 参考になるOSSのEPC実装

- http://www.openairinterface.org/
  - CでEPC全部実装しているがSPGWになっている
  - 日本だと富士通さんがコアメンバー
- https://github.com/travelping/ergw
  - ErlangでGGSN/PGW実装している
- http://nextepc.org/
  - CでMME,SGW,PGW,HSS,PCRFを実装している