FRR Scripting は何ができるのか

さくらインターネット
合田　和也

2026/06/19 ENOG Meeting 90

自己紹介

- 名前: 合田 和也 (Goda Kazuya)

- 経歴:
  - 2012/04 ~ 某 ISP にてソフトウェアルーターの開発
  - 2025/02 ~ さくらインターネットにてクラウドのネットワーク機能の開発

- 新潟県民歴:
  - 出生地が柏崎
  - 小学5年生〜高校卒業まで長岡に住んでました

FRR に Scripting があるのを知っていますか？

- User Guide をみてたら Scripting という項目が
  - on_rib_process_dplane_results というフックが使えるらしい
- Developer’s Guide の libfrr にも Scripting という項目が
  - 利用例 として route_map match で Script を呼び出だしている
  - ちなみに User Guide の route_map には Scripting の記述は存在しない
- この渾沌とした感じは面白そうな感じがする

本日の内容

- FRR Scripting の概要
  - 目指しているもの、できること
  - 2 つのフック
- 実際に使ってみよう
  - GBP によるフィルタリングを FRR Scripting を使って実装する
- まとめ - FRR Scripting は何ができるのか
  - 実際に使った上でもう一度考える

FRR Scripting は何を目指しているのか

- GSoC 2020[1]、GSoC 2021[2] の説明より
  - Luaスクリプトをフックとして呼び出し、コード変更なしに動作を変えること
    - ユーザースペースのネットワークルーティングスタック向け eBPF 実装

- 2026/06 現在 2 つのフックポイントがある
  - on_rib_process_dplane_results
  - route-map match

[1] https://frrouting.github.io/frr-gsoc/year-2020/projects/lua-userscripting.html
[2] https://frrouting.github.io/frr-gsoc/year-2021/projects/lua-hook-points.html

FRR Scripting は何を目指しているのか

- GSoC 2020[1]、GSoC 2021[2] の説明より
  - Luaスクリプトをフックとして呼び出し、コード変更なしに動作を変えること
    - ユーザースペースのネットワークルーティングスタック向け eBPF 実装

- 2026/06 [illegible] がある
  - on_rib_process_dplane_results
  - route-map match

本日はコレを中心に話す

[1] https://frrouting.github.io/frr-gsoc/year-2020/projects/lua-userscripting.html
[2] https://frrouting.github.io/frr-gsoc/year-2021/projects/lua-hook-points.html

route-map match フック

- route-map の match にスクリプトが指定できる

```
route-map LUASAMPLE permit 10
  match script rtscript
exit
```

- 実行される Lua スクリプトの例

```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)

    if prefix.network == "172.16.10.4/24" then
        return { action = RM_NOMATCH }
    else
        attributes["metric"] = attributes["metric"] + 7
        return { action = RM_MATCH_AND_CHANGE, attributes = attributes }
    end
end
```

route-map match で呼ばれる スクリプトから扱える情報

- 以下の情報が見える
  - Prefix
    - Prefix の情報
  - Peer
    - Prefix を受信した Peer の情報
  - Attribution
    - Prefix につく Attribution の一部
    - MED や LP など一部書き換えることができる

カテゴリ	フィールド	内容
Base	remote_as	リモートAS番号
	local_as	ローカルAS番号
	remote_id	リモート Router-ID
	local_id	ローカル Router-ID
	remote_address	リモートアドレス
	local_address	ローカルアドレス
	state	BGPセッション状態
	description	ピアの説明
	capabilities	ネゴシエート済みケーパビリティ
	flags	ピアフラグ
	password	BGP認証パスワード
	uptime	セッション確立からの経過時間
	last_readtime	最後にメッセージを受信した時刻
	last_resettime	最後にリセットした時刻
stats	open_in / open_out	OPENメッセージ 受信/送信数
	update_in / update_out	UPDATEメッセージ 受信/送信数
	update_time	最終UPDATE時刻
	keepalive_in / keepalive_out	KEEPALIVEメッセージ 受信/送信数
	notify_in / notify_out	NOTIFYメッセージ 受信/送信数
	refresh_in / refresh_out	ROUTE-REFRESHメッセージ 受信/送信数
	dynamic_cap_in / dynamic_cap_out	Dynamic Capabilityメッセージ 受信/送信数
	times_established	セッション確立回数
	times_dropped	セッション切断回数
timer	hold	
	keepalive	
	connect	
	route_advertisement	

Peer の情報

フィールド	内容	R	W
metric	MED	✅	✅
ifindex	Next-Hop インターフェースインデックス	✅	
aspath	AS_PATH	✅	
localpref	Local Preference	✅	✅

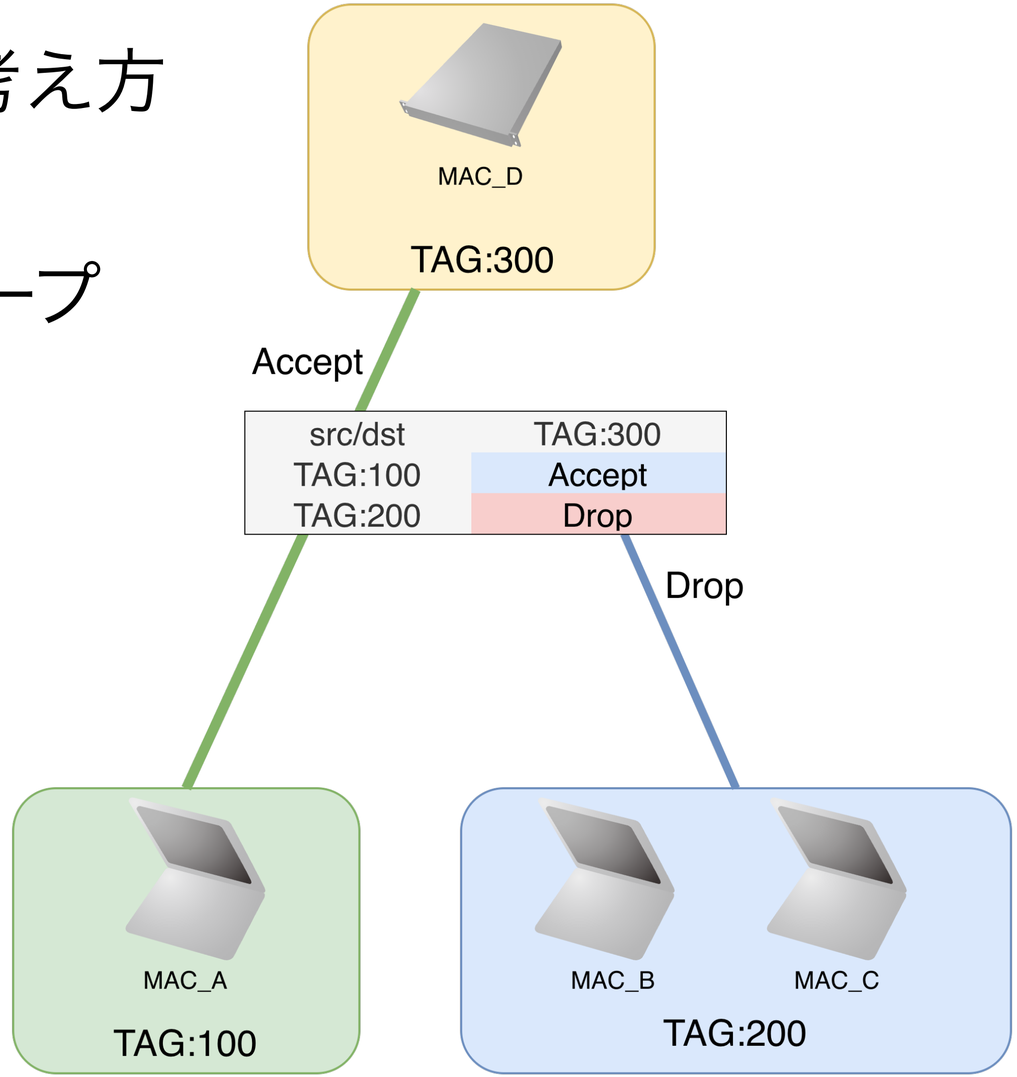
Attribution 情報

使ってみよう

~ GBP によるポリシーベースのフィルタ実装 ~

GBP - Group Base Policy とは

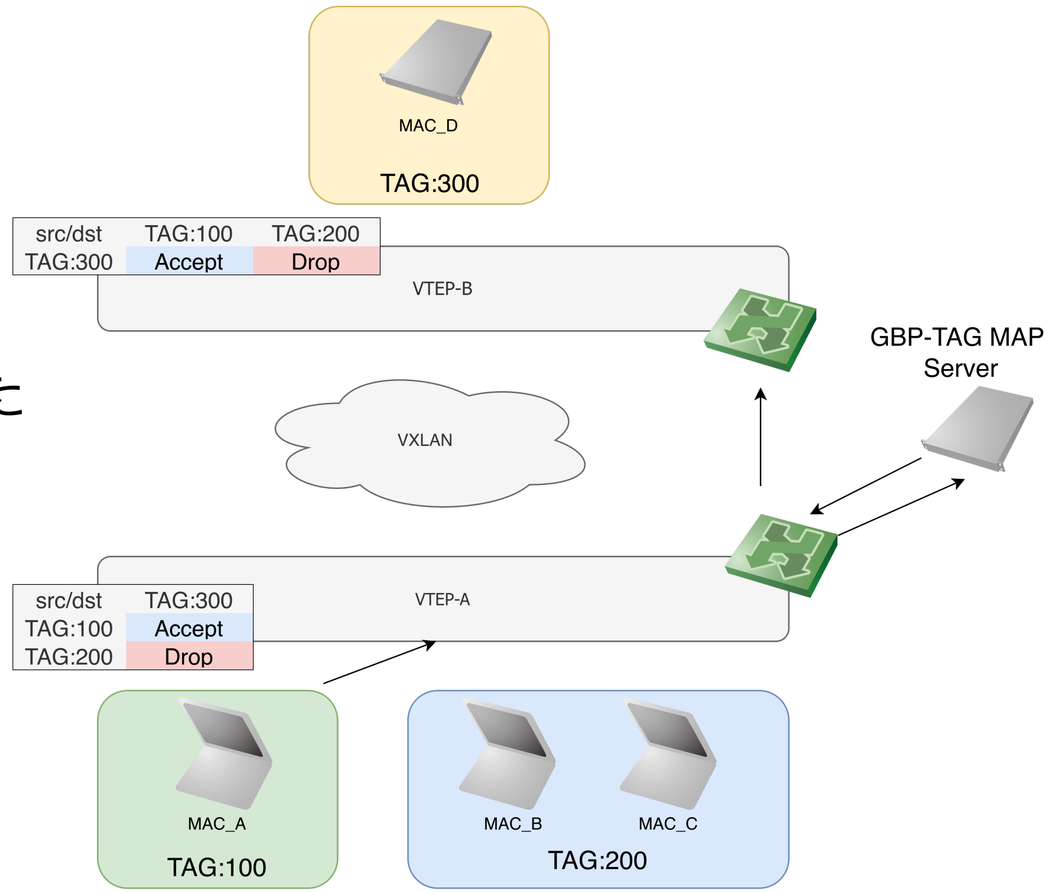
- ポリシーをグループ単位の関係で表現する考え方
- 例: 通信の可否を IP アドレスではなくグループ (TAG) ベースで行う
  - TAG:100 と TAG:300 は ACCEPT
  - TAG:200 と TAG:300 は DROP
- GBP の実現方法は複数ある
  - TAG をVXLAN のヘッダに埋め込むなど
    - https://datatracker.ietf.org/doc/html/draft-lemon-vxlan-gpe-gbp



実現方法として EVPN + GPI Extended-Community

- 端末と GBP-TAG の紐付きが事前にわかっていれば:
  - TAG をパケットに埋め込む必要がない
    - 端末-A は TAG:100 に紐づく
    - TAG:100 としてのポリシーを適用
  - Egress でフィルターをかけられる

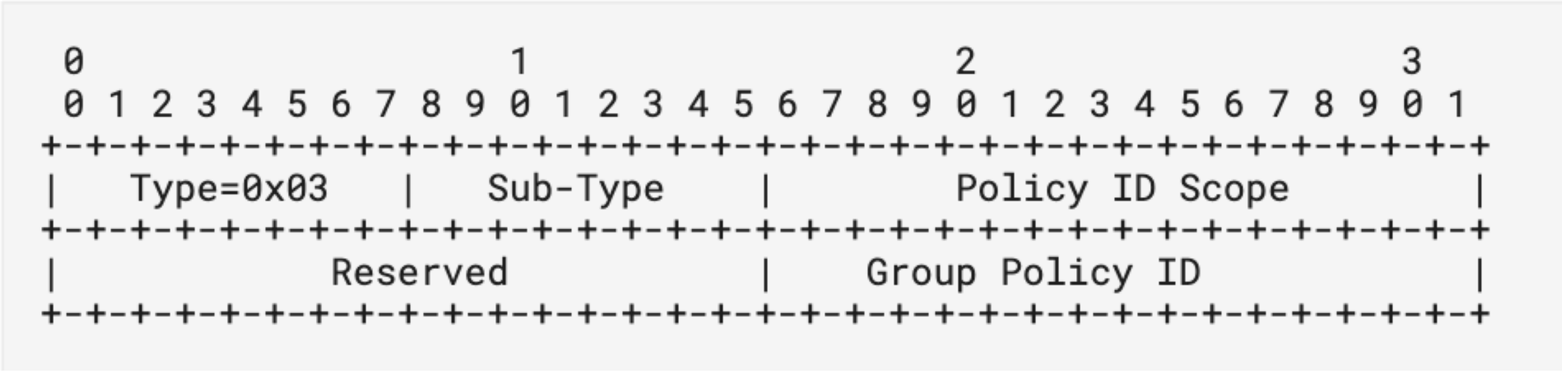
- 端末 (今回は MAC で識別) と GBP-TAG の紐付けかた
  - EVPN をコントロールプレーンとして利用し配布
    - EVPN は MAC_A は VTEP-A にいるという情報を広告する
  - Type2 (MAC) 経路に GBP-TAG の情報をつけて広告
    - これによりMAC と GBP-TAG が紐づけられる
  - Group Policy ID BGP Extended Community で TAG を紐付け
    - **これを Scripting により動的に追加する**



Group Policy ID BGP Extended Community

draft-wlin-bess-group-policy-id-extended-community-03

- GPI Extended Community とは
  - GBP のタグ (ID) を BGP Extended-Community で配送する仕組み
  - Sub-Type は 0x17 となる
    - ただし draft の RFC なので正式ではない

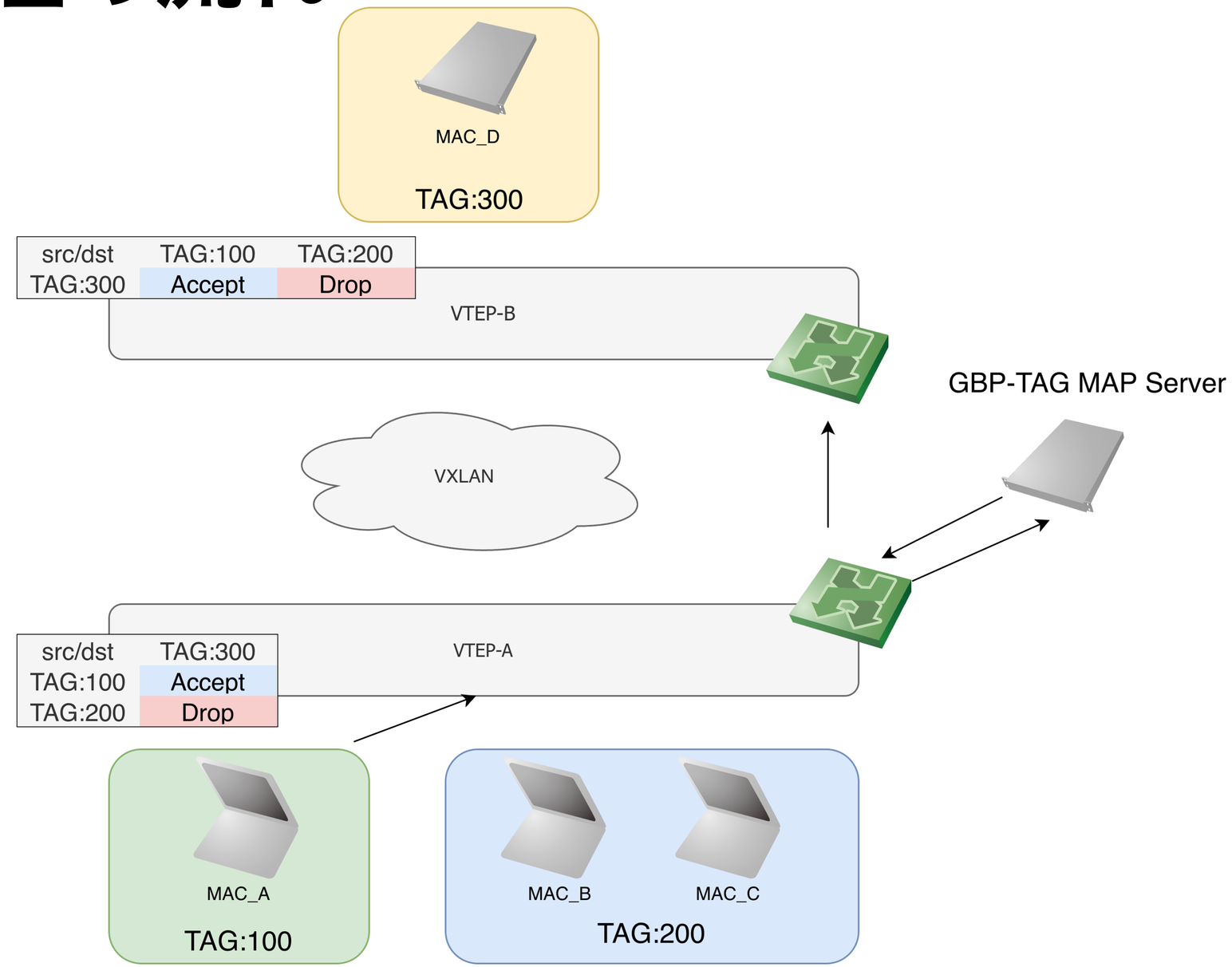


補足: 具体的なポリシー適用方法と操作

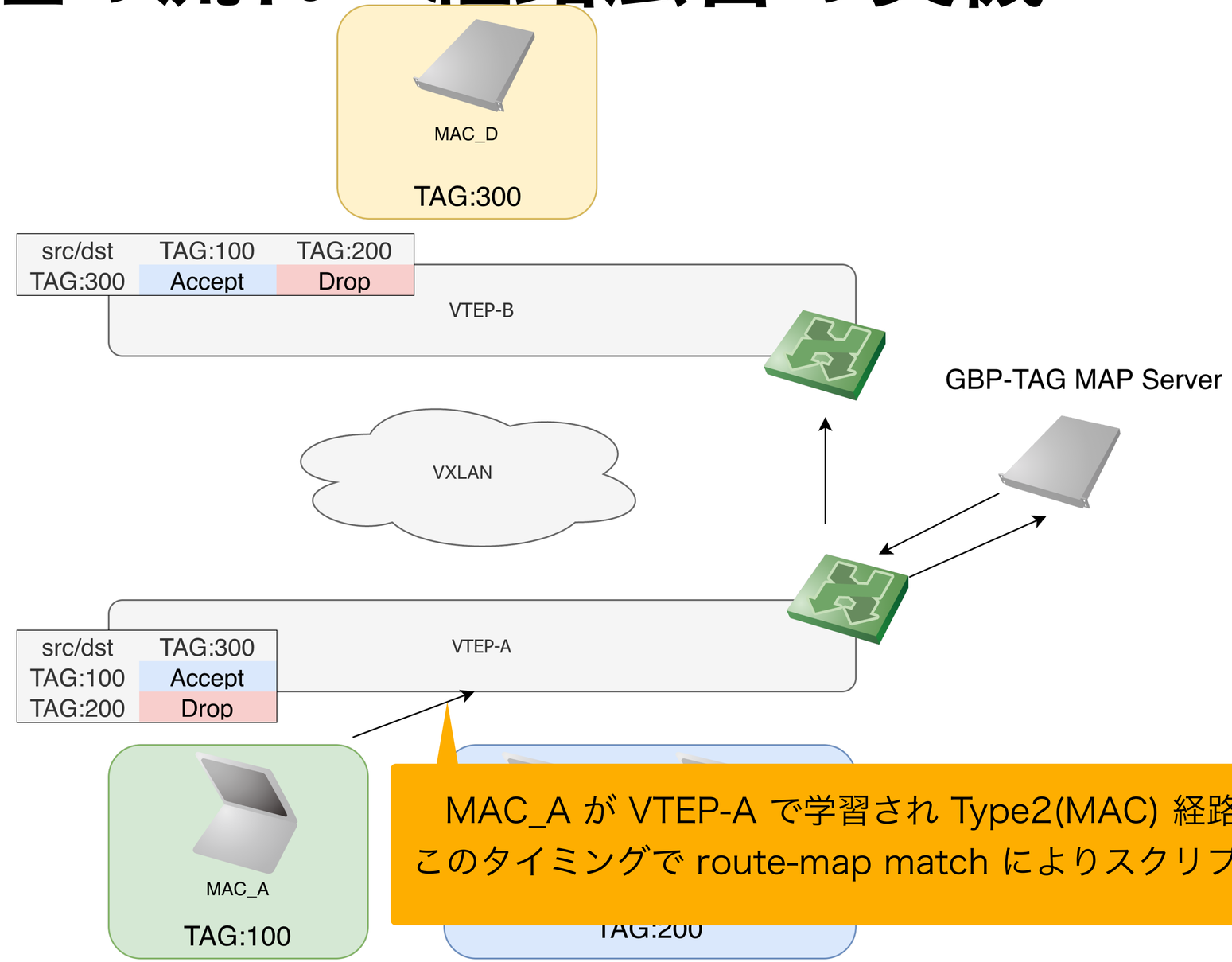
- nftable の set でグループを表現
  - グループに紐づくリソースを列挙
- この set を対象にポリシーを記述
  - グループベースのポリシーとなる
- Type2 経路受信時の挙動
  - TAG と対応する set に MAC を追加する

```
table bridge gbp_filter {
    set tag_100 {
        type ether_addr
        elements = { aa:bb:cc:dd:ee:01 }
    }

    set tag_200 {
        type ether_addr
        elements = { aa:bb:cc:dd:ee:01
                     aa:bb:cc:dd:ee:02 }
    }

    set tag_300 {
        type ether_addr
        elements = { aa:bb:cc:dd:ee:00 }
    }

    chain FORWARD {
        type filter hook forward priority 0; policy accept;

        # --- TAG:100 ⟷ TAG:300 : permit ---
        ether saddr @tag_300 ether daddr @tag_100 counter accept
        ether saddr @tag_100 ether daddr @tag_300 counter accept

        # --- TAG:200 ⟷ TAG:300 : deny ---
        ether saddr @tag_300 ether daddr @tag_200 counter drop
        ether saddr @tag_200 ether daddr @tag_300 counter drop
    }
}
```

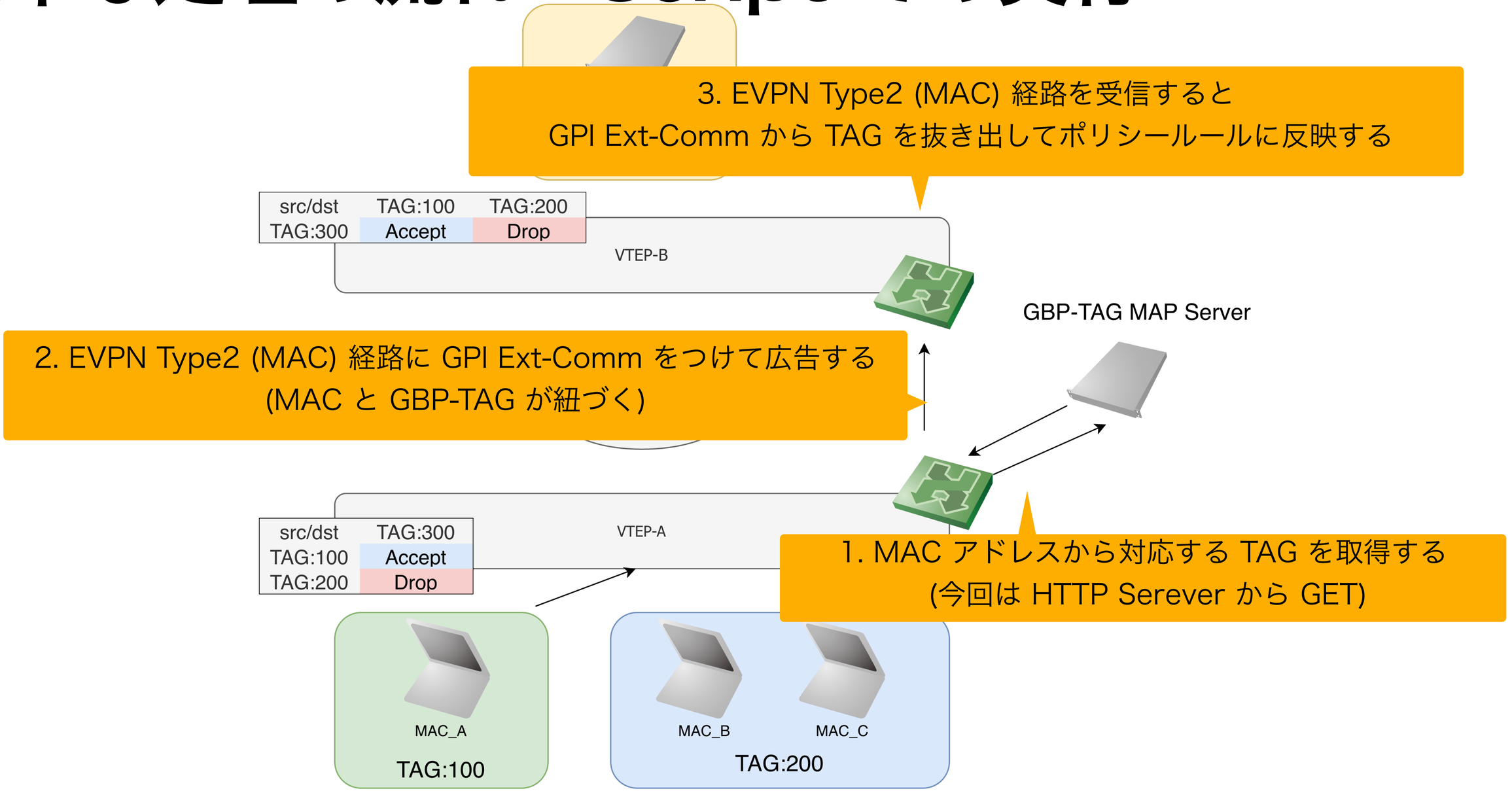
簡単な処理の流れ


MAC_D
TAG:300
src/dst
TAG:300
TAG:100
Accept
TAG:200
Drop
VTEP-B
GBP-TAG MAP Server
VXLAN
src/dst
TAG:100
TAG:200
TAG:300
Accept
Drop
VTEP-A
MAC_A
TAG:100
MAC_B
MAC_C
TAG:200

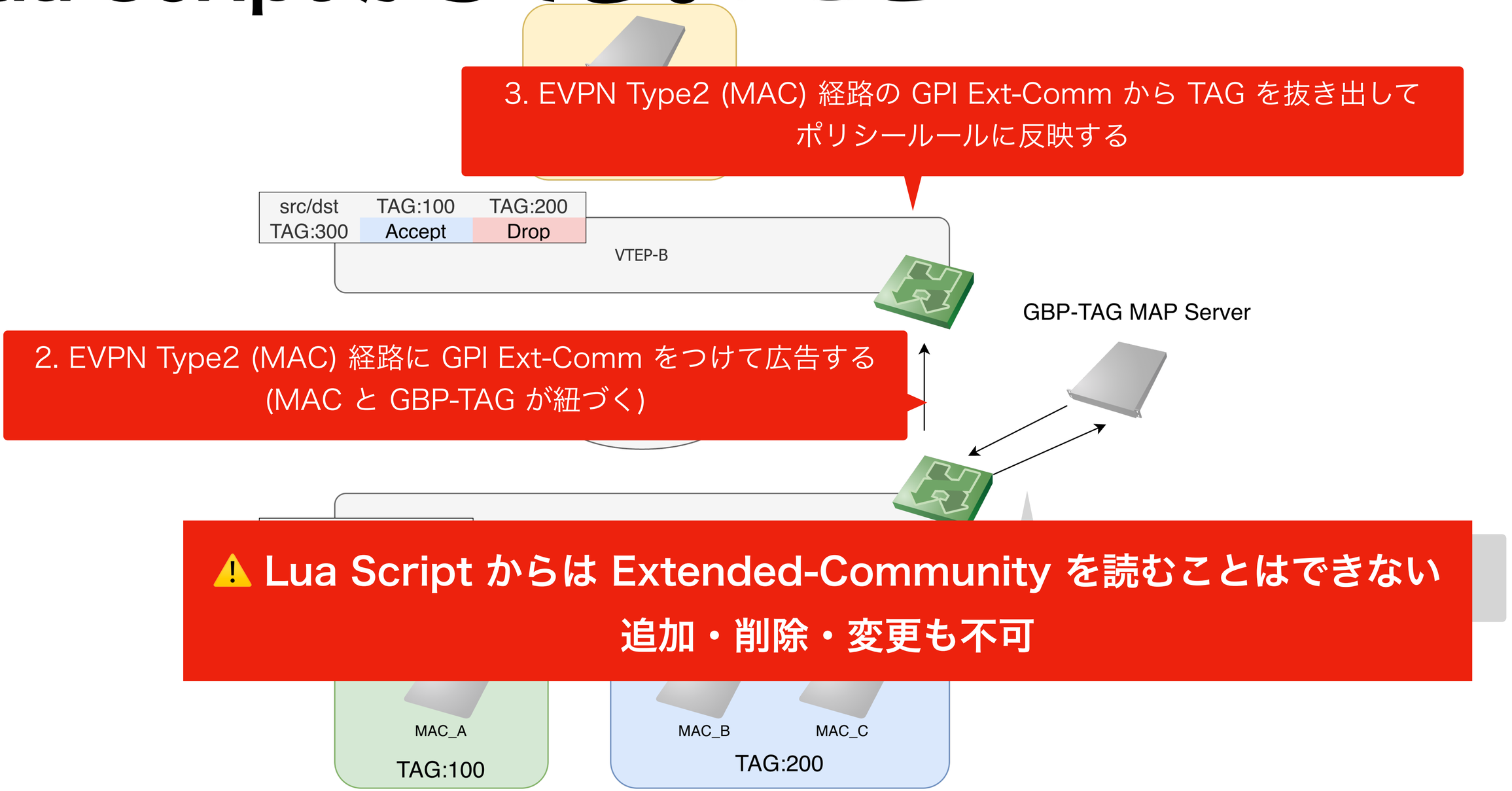
簡単な処理の流れ - 経路広告の契機



簡単な処理の流れ - Script での実行



Lua Script からできないこと



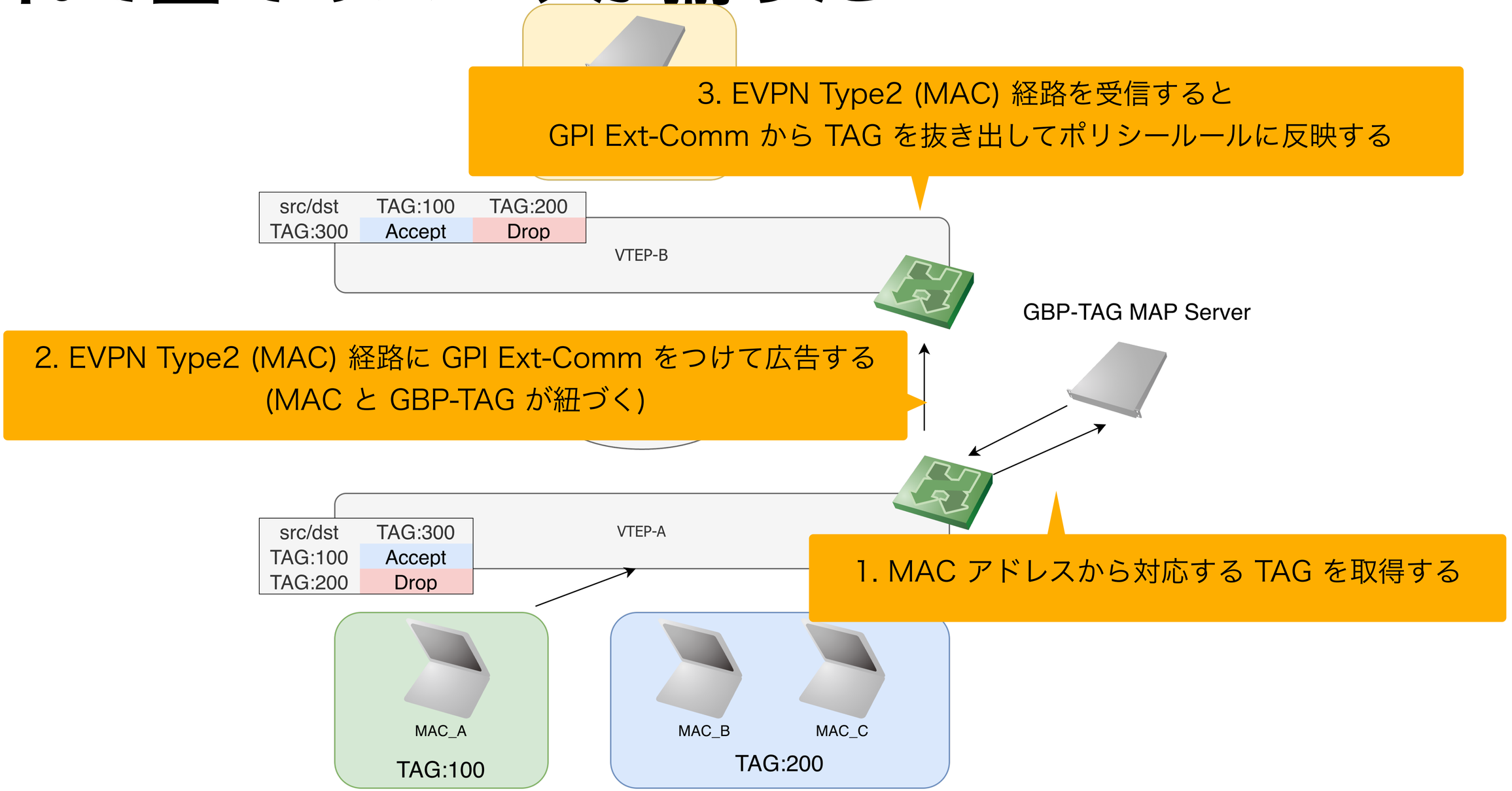
Extended Community を操作できるように実装

- Lua Script からの使い方

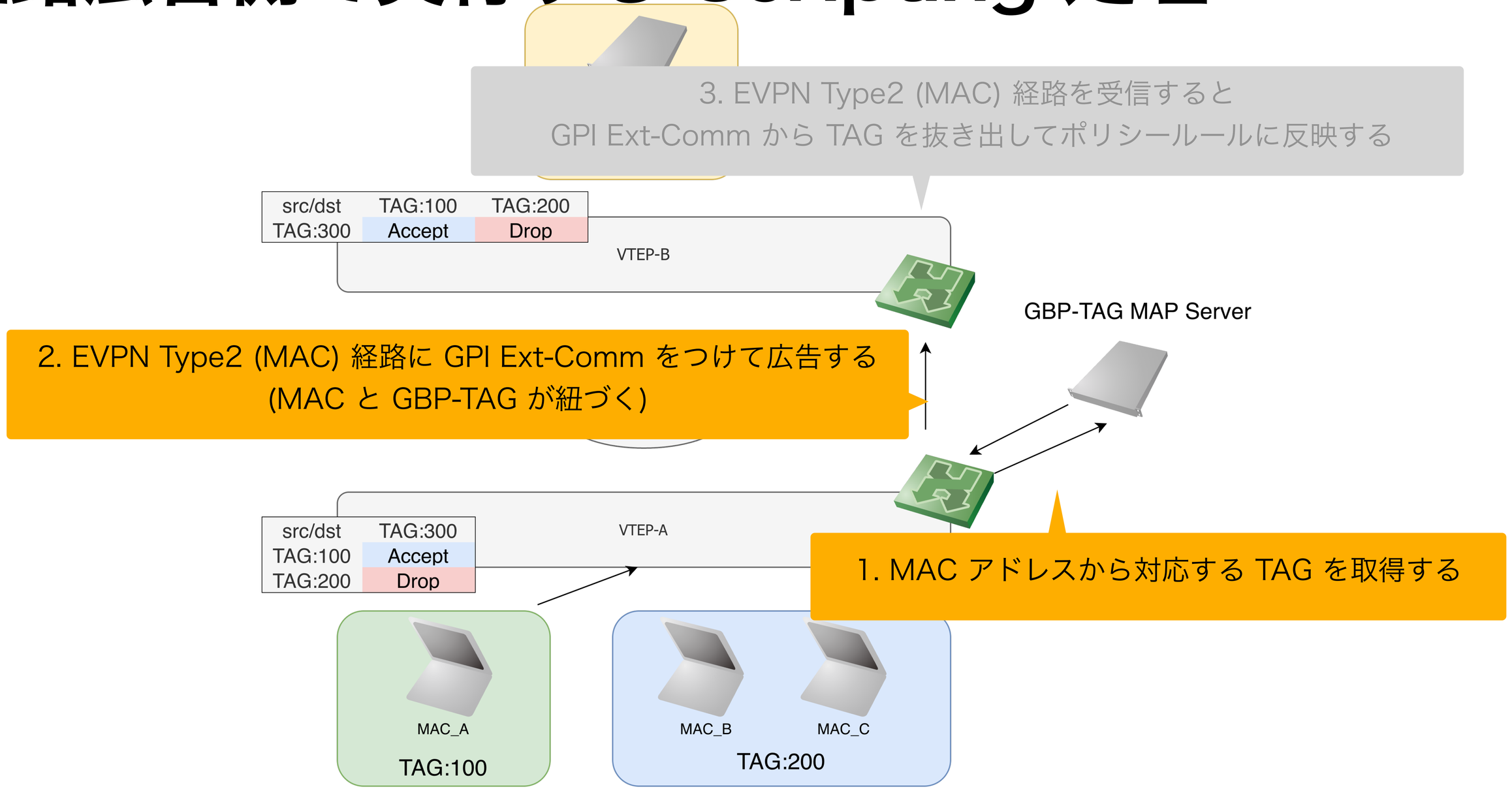
```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)

    ecoms = {}
    for _, ec in ipairs(attributes.ext_community) do
      table.insert(ecoms, ec)
    end

    local gbp_ecom = string.pack(">BBHHH", 0x03, 0x17, 0x0000, 0x0000, tonumber(obj.sgt))
    table.insert(ecoms, gbp_ecom)

    attributes.ext_community = ecoms

    return { action = RM_MATCH_AND_CHANGE, attributes = attributes }
end
```

https://github.com/gokzy/frr/commit/9e8d3c112e30a538aba510958a4e846d7abc523d

これで全てのパーツが揃った



経路広告側で実行する Scripting 処理



経路広告側で実行するスクリプト (一部抜粋)

```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)
    local type2_mac_only = "^%[(%d+)%]:%[(%d+)%]:%[(%d+)%]:%[([0-9a-fA-F:]+)%]$"
    local evpn_type, flags, prefix_len, mac = tostring(prefix.network):match(type2_mac_only)
    -- snip --
    local body, code = http.request("http://10.254.254.254:8080/sgt?mac=" .. mac)

    local obj = json.decode(body)

    ecoms = {}
    for _, ec in ipairs(attributes.ext_community) do
      table.insert(ecoms, ec)
    end

    local gbp_ecom = string.pack(">BBHHH", 0x03, 0x17, 0x0000, 0x0000, tonumber(obj.sgt))
    table.insert(ecoms, gbp_ecom)

    attributes.ext_community = ecoms

    return { action = RM_MATCH_AND_CHANGE, attributes = attributes }
end
```

経路広告側で実行するスクリプト (一部抜粋)

```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)
    local type2_mac_only = "^%[(%d+)%]:%[(%d+)%]:%[(%d+)%]:%[([0-9a-fA-F:]+)%]$"
    local evpn_type, flags, prefix_len, mac = tostring(prefix.network):match(type2_mac_only)
    -- snip --
    local body,

    local obj = json.decode(body)

    ecoms = {}
    for _, ec in ipairs(attributes.ext_community) do
      table.insert(ecoms, ec)
    end

    local gbp_ecom = string.pack(">BBHHH", 0x03, 0x17, 0x0000, 0x0000, tonumber(obj.sgt))
    table.insert(ecoms, gbp_ecom)

    attributes.ext_community = ecoms

    return { action = RM_MATCH_AND_CHANGE, attributes = attributes }
end
```

Prefix が文字列で渡されるため、正規表現で広告される MAC アドレスを抜き出す

例. [2]:[0]:[48]:[aa:bb:cc:dd:ee:00]

経路広告側で実行するスクリプト (一部抜粋)

```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)
    local type2_mac_only = "^%[(%d+)%]:%[(%d+)%]:%[(%d+)%]:%[([0-9a-fA-F:]+)%]$"
    local evpn_type, flags, prefix_len, mac = tostring(prefix.network):match(type2_mac_only)
    -- snip --
    local body, code = http.request("http://10.254.254.254:8080/sgt?mac=" .. mac)

    local obj = json.decode(body)

    ecoms = {}
    for _, ec
      table.insert(ecoms, ec)
    end

    local gbp_ecom = string.pack(">BBHHH", 0x03, 0x17, 0x0000, 0x0000, tonumber(obj.sgt))
    table.insert(ecoms, gbp_ecom)

    attributes.ext_community = ecoms

    return { action = RM_MATCH_AND_CHANGE, attributes = attributes }
end
```

HTTP Server から MAC に対応する TAG を取得する

経路広告側で実行するスクリプト (一部抜粋)

```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)
    local type2_mac_only = "^%[(%d+)%]:%[(%d+)%]:%[(%d+)%]:%[([0-9a-fA-F:]+)%]$"
    local evpn_type, flags, prefix_len, mac = tostring(prefix.network):match(type2_mac_only)
    -- snip --
    local body, code = http.request("http://10.254.254.254:8080/sgt?mac=" .. mac)

    local obj = json.decode(body)

    ecoms = {}
    for _, ec in ipairs(attributes.ext_community) do
      table.insert(ecoms, ec)
    end

    local gbp_ecom = string.pack(">BBHHH", 0x03, 0x17, 0x0000, 0x0000, tonumber(obj.sgt))
    table.insert(ecoms, gbp_ecom)

    attribut

    return {
end
```

Type=3, SubType=0x17

GBP TAG を埋め込む

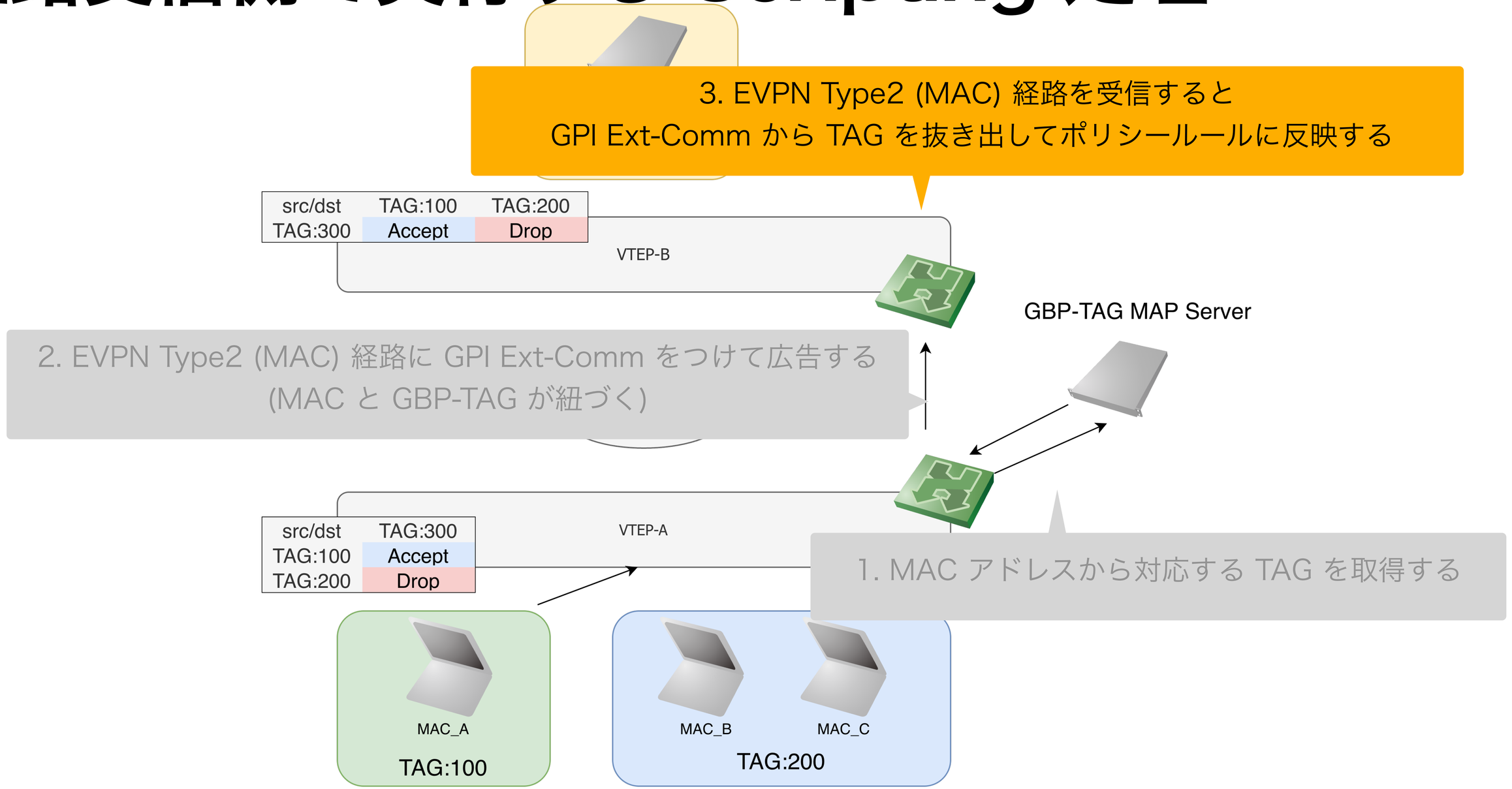
GPI Extended Community を追加する

経路広告側で実行するスクリプト (一部抜粋)

```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)
    local type2_mac_only = "^%[(%d+)%]:%[(%d+)%]:%[(%d+)%]:%[([0-9a-fA-F:]+)%]$"
    local evpn_type, flags, prefix_len, mac = tostring(prefix.network):match(type2_mac_only)
    -- snip --
    local body, code = http.request("http://10.254.254.254:8080/sgt?mac=" .. mac)

    local obj = json.decode(body)

    ecoms = {}
    for _, ec in ipairs(attributes.ext_community) do
      table.insert(ecoms, ec)
    end

    local gb                                    7, 0x0000, 0x0000, tonumber(obj.sgt))
    table.ins                                   ecoms, gbp_ecom)

    attributes.ext_community = ecoms

    return { action = RM_MATCH_AND_CHANGE, attributes = attributes }
end
```

Ext-Comm を更新して return する

経路受信側で実行する Scripting 処理



経路受信側で実行するスクリプト (一部抜粋)

```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)
    local type2_mac_only = "^%[(%d+)%]:%[(%d+)%]:%[(%d+)%]:%[([0-9a-fA-F:]+)%]$"
    local evpn_type, flags, prefix_len, mac = tostring(prefix.network):match(type2_mac_only)

    gpi_ec = nil
    for _, ec in ipairs(attributes.ext_community) do
      local type_ = ec:byte(1)
      local subtype = ec:byte(2)
      if type_ == 0x03 and subtype == 0x17 then
          gpi_ec = ec
          break
      end
    end

    local type_, subtype, scope, reserved, sgt = string.unpack(">BBHHH", gpi_ec)
    os.execute("nft add element bridge gbp_filter stag_" .. tostring(sgt) .. " { " .. mac .. " }")

    return { action = RM_NOMATCH }
end
```

経路受信側で実行するスクリプト (一部抜粋)

Ext-Comm リストから GPI Ext-Comm を探す

```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMA
    local type2_mac_onl
    local evpn_type, fl                                    ac_only)

    gpi_ec = nil
    for _, ec in ipairs(attributes.ext_community) do
      local type_ = ec:byte(1)
      local subtype = ec:byte(2)
      if type_ == 0x03 and subtype == 0x17 then
          gpi_ec = ec
          break
      end
    end

    local type_, subtype, scope, reserved, sgt = string.unpack(">BBHHH", gpi_ec)
    os.execute("nft add element bridge gbp_filter stag_" .. tostring(sgt) .. " { " .. mac .. " }")

    return { action = RM_NOMATCH }
end
```

経路受信側で実行するスクリプト (一部抜粋)

```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)
    local type2_mac_only = "^%[(%d+)%]:%[(%d+)%]:%[(%d+)%]:%[([0-9a-fA-F:]+)%]$"
    local evpn_type, flags, prefix_len, mac = tostring(prefix.network):match(type2_mac_only)

    gpi_ec = nil
    for _, ec in ipairs(attributes.ext_community) do
      local type_ = ec:byte(1)
      local subtype = ec:byte(2)
      if type_ == 0x03 and subtype == 0x17 then
          gpi_ec = ec
          break
      end
    end

    local type_, subtype, scope, reserved, sgt = string.unpack(">BBHHH", gpi_ec)
    os.execute("nft add element bridge gbp_filter stag_" .. tostring(sgt) .. " { " .. mac .. " }")

    return { action = RM_NOMATCH }
end
```

GPI Ext-Comm から TAG を抜き出す

経路受信側で実行するスクリプト (一部抜粋)

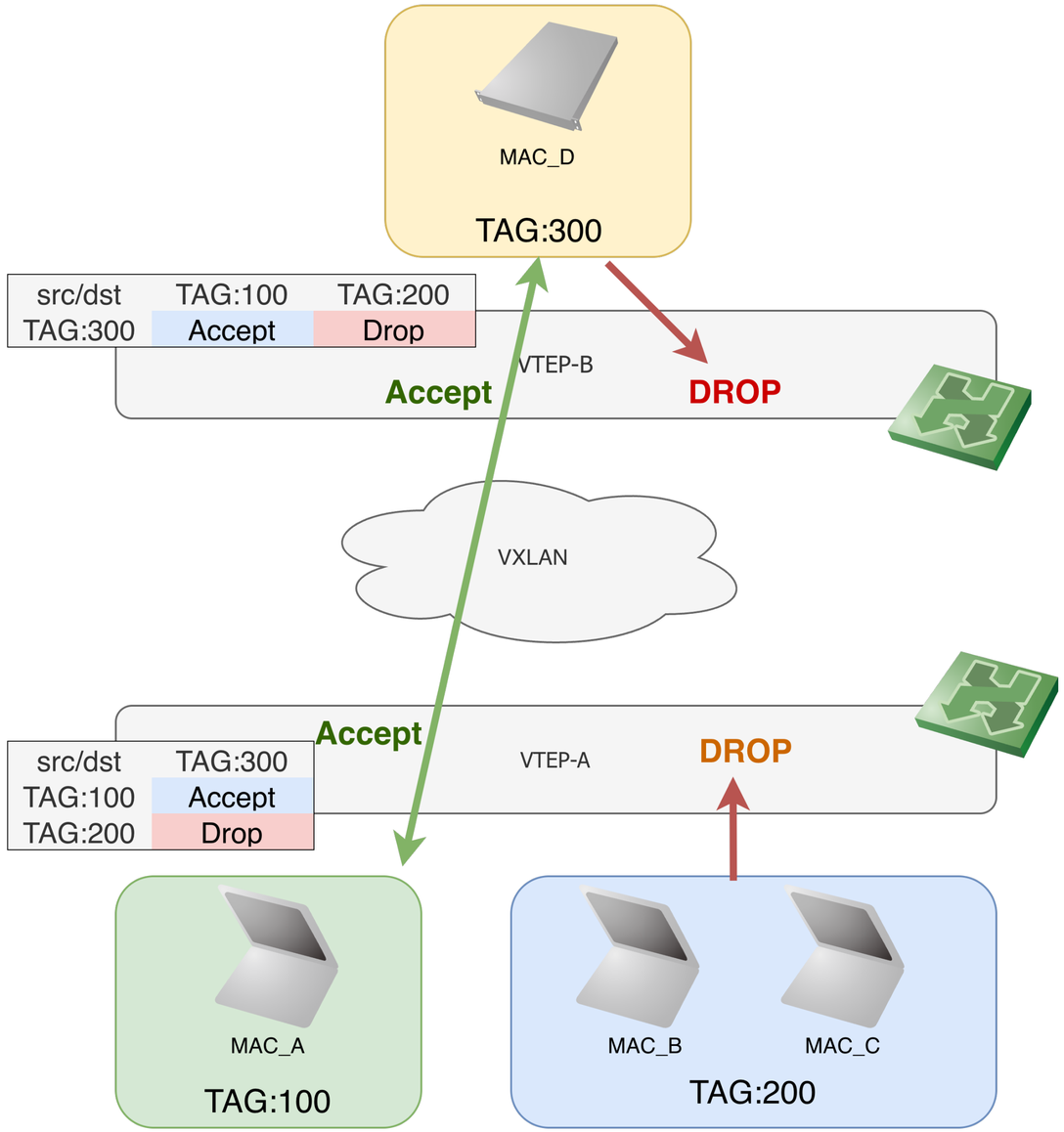
```
function route_match(prefix, attributes, peer,
    RM_FAILURE, RM_NOMATCH, RM_MATCH, RM_MATCH_AND_CHANGE)
    local type2_mac_only = "^%[(%d+)%]:%[(%d+)%]:%[(%d+)%]:%[([0-9a-fA-F:]+)%]$"
    local evpn_type, flags, prefix_len, mac = tostring(prefix.network):match(type2_mac_only)

    gpi_ec = nil
    for _, ec in ipairs(attributes.ext_community) do
      local type_ = ec:byte(1)
      local subtype = ec:byte(2)
      if type_ == 0x03 and subtype == 0x17 then
          gpi_ec = ec
          break
      end
    end

    local type_, subtype, scope, reserved, sgt = string.unpack(">BBHHH", gpi_ec)
    os.execute("nft add element bridge gbp_filter stag_" .. tostring(sgt) .. " { " .. mac .. " }")

    return { action = RM_NOMATCH }
end
```

nftables にフィルターを追加する

デモ

- MAC_D から MAC_A に ping を送信
  1. VTEP-A から VTEP-B に MAC_A の経路が広告
     - ここでは MAC_A の eth1 の link-up 契機
     - この経路には GBP-TAG:100 も含まれる
  2. VTEP-B が経路を受信し nftable に反映する
  3. MAC_D から ping を送信
  4. TAG:100 <=> TAG:300 は Accept なので通る

- MAC_D から MAC_B に ping を送信
  1. VTEP-A から VTEP-B に MAC_B の経路が広告
     - ここでは MAC_B の eth1 の link-up 契機
     - この経路には GBP-TAG:200 も含まれる
  2. VTEP-B が経路を受信し nftable に反映する
  3. MAC_D から ping を送信
  4. TAG:200 <=> TAG:300 は Drop なので通らない



実現できていないこと

- 動的なフィルターの削除
  - EVPN 経路の withdraw 時にフィルターを削除したいが route-map は呼ばれない
    - そもそも route-map は経路情報更新イベントを通知するためのものではないので

- on_rib_process_dplane_results フックを使えばできそうだが
  - D-Plane から MAC の経路が削除されるときに Lua Script が呼ばれる
    - 削除される MAC や VTEP-IP などが Lua Script から読むことができる
  - しかし MAC と GBP-TAG の紐付き情報が得られないのでスマートに削除ができない
    - zebra のフックイベントなので bgpd が持つ ext-comm の情報は持ってない

- デーモン間のインタフェースが「コード変更なしに動作を変える」ことの制約になる
  - クロスデーモンで Cookie 的なものをつけれる何らかの方法があると良い
  - DB 等の外部リソースで解決でも良いが

まとめ - FRR Scripting は何ができるのか

主に route-map match 視点

- PoC などには使いやすい
  - 今回のように Extended-Community を気軽に追加する
- 外部リソースを参照して実行時に動作を変える用途に
  - 外部監視系と連携して MED を変更するなど
- 現状の FRR Scripting はまだまだ実験的な実装
  - もう少し整備されないと使い所が難しそう

Backup

参考: FRR Scripting の現状

- まだまだ実験的な実装
  - おそらく多くの Linux ディストリのパッケージでは無効化されている
    - ビルド時に --enable-scripting の指定が必要
- 2021年以来 フックも2箇所から増えておらず開発も活発ではない
  - 昨年 Lua 5.4 対応なども入っているのでメンテされてないわけではない
- ちょっと触っただけでバグが見つかる
  - https://github.com/gokzy/frr/commit/2b83bf41e9cfd8a4eabf2bfd2280b056ca3a4fe3
  - https://github.com/gokzy/frr/commit/33a188c5b49825a6ef62dba9bb74fb655d29f4c1