



OTG-MCP触ってみた

～トラフィック定義を自然言語で書く～

NTTフィールドテクノ 奥村 陸

NTT ネットワークイノベーションセンタ 中野寛二

2026年6月19日

JANOG57.5 Interim Meeting

目次

1. 自己紹介
2. Open Traffic Generator (OTG) とは
3. OTGの利用方法
4. OTG-MCPとは
5. OTG-MCP実施結果
6. まとめ

1. 自己紹介



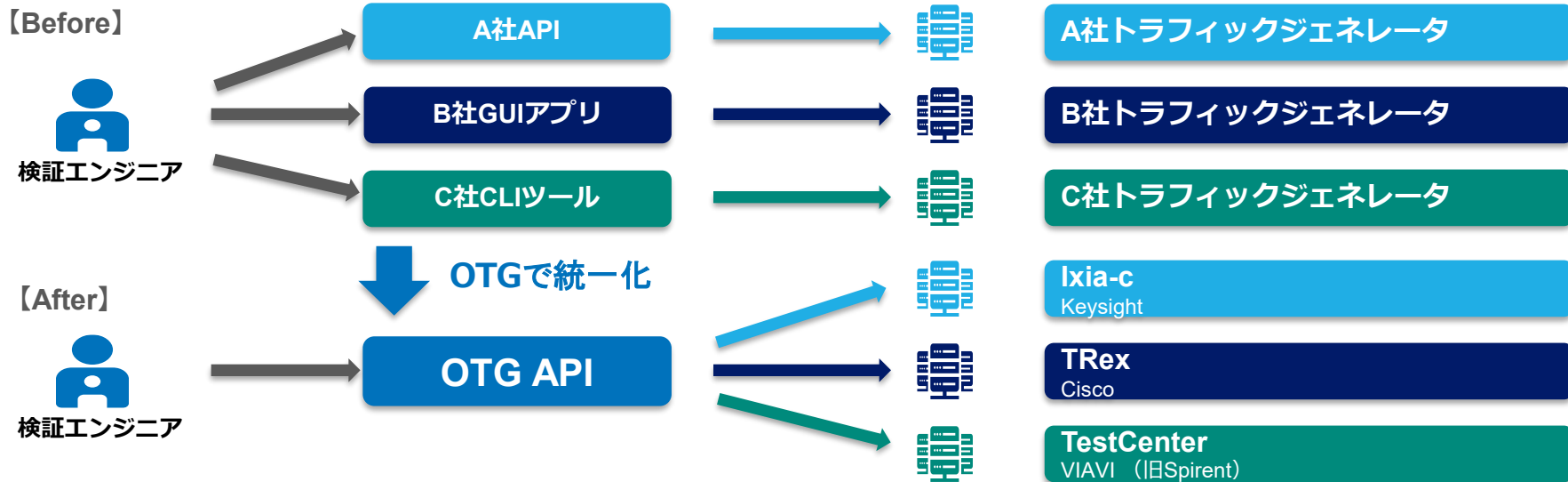
奥村 陸

NTTフィールドテクノ
(NTT ネットワークイノベーション
センターにOJT派遣中)

- **業務内容:** ネットワーク装置の**検証自動化**とその手法の高度化を担当しています
- **今日の発表:** **業務での自動化手法高度化検討の一環**として、OTG/OTG-MCPを触ってみた結果や所感を共有します
- JANOG初登壇です。お手柔らかにお願いします

2. Open Traffic Generator (OTG) とは

- ベンダごとに独自のAPI/GUIアプリ/CLIで操作していたトラフィックジェネレータを、**Open Traffic Generator (OTG)**（オープン・ベンダ中立・宣言的・intentベースな共通API仕様）で統一化できる
- KeysightのコンテナベースでOTG APIの全機能を利用可能な**Ixia-c**を始めとして、Ciscoの**TRex**やVIAVI（旧Spirent）の**TestCenter**などのトラフィックジェネレータでもOTGを利用することができる



3. OTGの利用方法

- OTG API経由でトラフィックジェネレータを制御する際には、Python/Goでトラフィック定義を記載するsnappi/gosnappiやCLIツールのotgenなどのクライアントを利用できる
- トラフィック定義等の設定はJSONやYAMLにシリアライズできるため、コードベースでの管理がしやすく、装置やクライアント間で共有することもできる
- snappiでのトラフィック定義は宣言的で比較的書きやすいが、snappiの記法やOTGのデータモデルの習熟コストがかかる

snappi 記法 (Python)

```
cfg = api.config()
cfg.ports.port(name='p1')
cfg.ports.port(name='p2')
f1 = cfg.flows.flow(name='flow_p1p2')[-1]
f1.tx_rx.port.tx_name = 'p1'
f1.tx_rx.port.rx_name = 'p2'
f1.size.fixed = 128; f1.rate.pps = 1000
f1.duration.fixed_packets.packets = 1000
eth, ip, udp = f1.packet.ethernet().ipv4().udp()
ip.src.value, ip.dst.value = '10.0.0.1', '10.0.0.2'
udp.dst_port.values = [4000, 4044, 4060]
```

シリアライズ

デシリアライズ

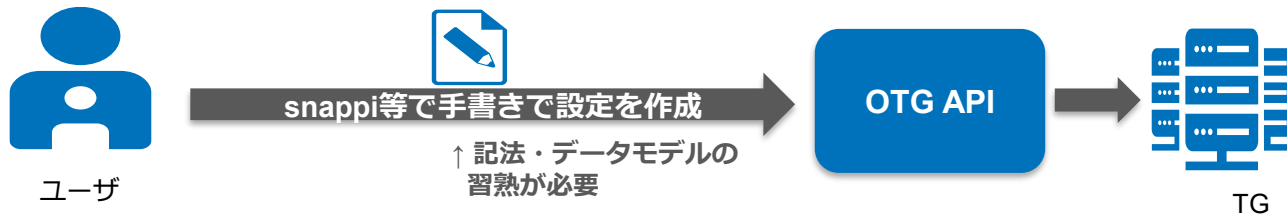
シリアライズ後の YAML

```
flows:
- name: flow_p1p2
  tx_rx: {port: {tx_name: p1, rx_name: p2}}
  size: {fixed: 128}
  rate: {pps: '1000'}
  duration: {fixed_packets: {packets: 1000}}
  packet:
  - ethernet: {}
  - ipv4:
      src: {value: 10.0.0.1}
      dst: {value: 10.0.0.2}
  - udp: {dst_port: {values: [4000, 4044, 4060]}}
```

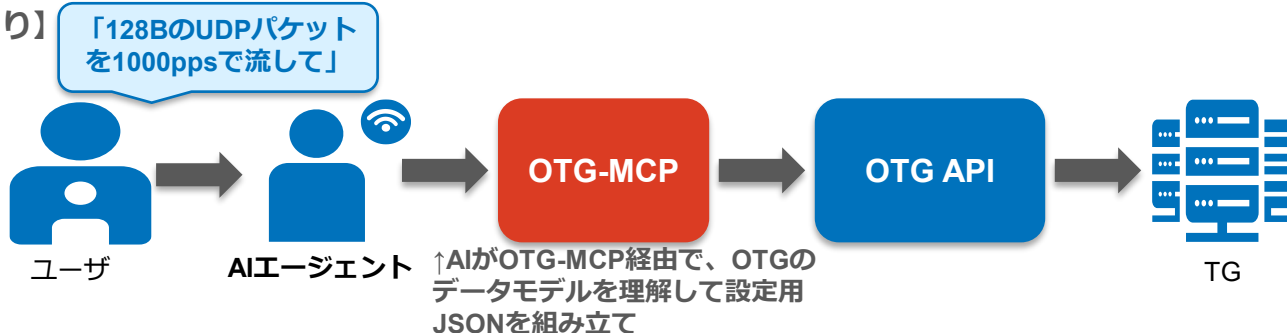
snappiの記法やOTGのデータモデルへの習熟は必要

4. OTG-MCPとは

- **OTG-MCP**はOTG API経由でOTG対応のトラフィックジェネレータを**AIエージェント**から操作できるようにするMCPサーバで、個人リポジトリ上で公開されている
- OTG-MCPを利用することで、ユーザはsnappiの記法やデータモデルに**習熟することなく、自然言語**でトラフィックの意図を伝えるだけでAIエージェントにOTG-MCP経由でトラフィックジェネレータを操作させることができる



[OTG-MCPあり]

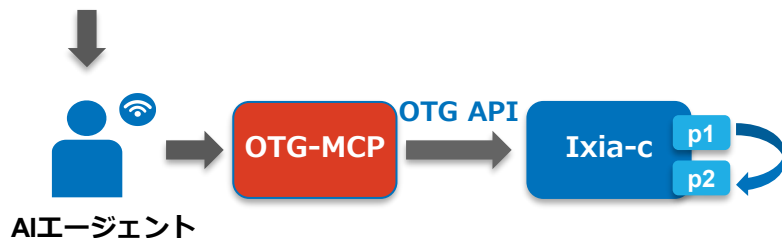


5. 実施結果①

- Ixia-cのポートをループバック接続した環境上で、実際にOTG-MCPを操作した結果を共有する
- AIエージェントへのプロンプトではOTGのデータモデルに関する情報は特に指示せず、**送信元MAC/宛先MACのみを指定**してUDPパケットの生成を依頼した
- **指示していない要素(src/dst IP、ポート、フレームサイズ、レート、Duration)はAIが補完**し、指示通りのトラフィック生成に成功した

プロンプト:
以下のパケットを p1 から p2 に流してください。

- プロトコル: UDP
- 送信元 MAC: 02:1a:2b:3c:4d:01
- 宛先 MAC: 02:1a:2b:3c:4d:02



パケットキャプチャでの確認結果

パラメータ	値	区分
Ethernet II src→dst	02:1a:2b:3c:4d:01 → 02:1a:2b:3c:4d:02	指定
IPv4 src→dst	192.168.1.1 → 192.168.1.2	補完
UDP src→dst port	1234 → 5678	補完
Frame Size	128 byte	補完
Rate	100 pps	補完
Duration	10 packets	補完

→プロンプトで指定したパラメータは指定通り
+ 未指定パラメータはAIが補完

5. 実施結果②

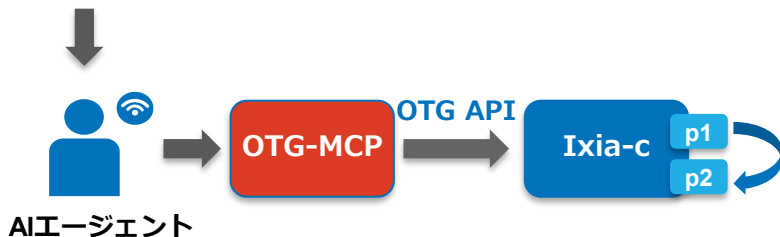
- AIエージェントにSRv6 L3VPNを実現するパケットの生成を依頼した
- 利用したバージョンのOTGでは未対応のSRv6のSRH（拡張ヘッダ）を作るようなプロンプトを投げた際には、AIが機転を利かせ、OTGの**customフィールド（任意のバイト列を直接指定可能）**を利用して**SRHを構築**する結果が見られた
- パケットキャプチャからも構造上正しいパケットが生成されていることを確認できた

プロンプト:

SRv6 で L3VPN を実現するパケットを生成してください。

SR List = `[2001:db8:1::1, 2001:db8:2::1, 2001:db8:3::100]`（論理パス順）、

内側は IPv4 + UDP のフレームを p1 から送信してください。



パケットキャプチャでの確認結果

パラメータ	値	区分
Ethernet II src→dst	00:00:00:00:00:01 → 00:00:00:00:00:02	補完
Outer IPv6 src	2001:db8::1	補完
Outer IPv6 dst	2001:db8:1::1	補完
SRH Segment[0]	2001:db8:3::100	指定
SRH Segment[1]	2001:db8:2::1	指定
SRH Segment[2]	2001:db8:1::1	指定
Inner IPv4 src→dst	192.168.1.1 → 192.168.2.1	補完
UDP src→dst	1234 → 5678	補完

→SRHはAIが直接バイト列を記載して指定通りのパケットを生成

- **OTG**:トラフィックジェネレータの**オープン・ベンダ中立・宣言的・インテントベース**なAPI仕様
- **OTG-MCP**:トラフィックジェネレータを**AIエージェント**からOTG API経由で操作できるMCPサーバ
- 指定していない**パラメータの補完**や、未対応のヘッダに**バイト列の直接指定**での対応など、AIエージェントが意図を理解して機転を利かせる様子を確認できた
- トラフィック生成の補助へ活用可能と考えられるが、AIの自律的な検証実行などの本格的な使用に向けては、**個人リポジトリ**で公開されているMCPサーバであることや、OTG-MCPの各トラフィックジェネレータへの対応が十分ではないことが課題である

OTG-MCP

OTG APIをAIが叩けるMCPサーバ

✓ 強み

- 自然言語で制御可能でOTG特有の記法習熟等が不要
- AIエージェントが試行錯誤しながら、指示したトラフィックを作成可能

⚠ 課題

- 個人リポジトリで公開されているMCPサーバでメンテナンス性に課題
- RESTインタフェースのみに対応しているため、gRPCを利用する装置（TestCenter等）に接続する際に開発が必要となる

Innovating a Sustainable Future for People and Planet



参考資料

1. **環境構築** otg-mcp のインストールとMCPの接続方法を記載する
2. **作業ディレクトリ** Claude Codeの起動ディレクトリの環境を記載する
3. **otg-mcp の MCP ツール** otg-mcpの全 12 ツールの概要を記載する
4. **実施フロー** デモ全体のフローの概要を記載する
5. **デモ①** デモ①（UDPパケットの生成）の記録を記載する
6. **デモ②** デモ②（SRHの生成）の記録を記載する
7. **付録 (ツール詳細)** otg-mcpの全 12 ツールの実際の入出力例を記載する

1. 環境構築：テスト環境

- 実際にデモ・再現検証を行って**動作を確認した環境**を記載する(同等のLinux環境であれば動作する想定)

■ テスト環境

項目	デモ実施環境
OS	Ubuntu 24.04 LTS (x86_64)
vCPU / メモリ	2 コア / 4 GB
Python	3.12.3
Docker Engine	29.3.1
Containerlab	0.73.0
otg-mcp	v0.1.3a.0
snappi	1.29.0
ixia-c-one	latest (controller 1.43.0 / apiVersion 1.43.0)

■ 前提ツール

ツール	公式インストール手順
Docker Engine	https://docs.docker.com/engine/install/
Containerlab	https://containerlab.dev/install/

■ 導入確認

```
docker --version && containerlab version
```

1. 環境構築 : otg-mcp 導入

- otg-mcpが公開されているリポジトリをcloneしてPython仮想環境にインストールする

■ インストール手順

```
# 作業用ベースディレクトリ (以下 /path/to/otg-demo と表記)
mkdir -p /path/to/otg-demo && cd /path/to/otg-demo

git clone https://github.com/h4ndzdatm0ld/otg-mcp.git
cd otg-mcp

python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt -e ".[dev]"
```

■ インストール確認

```
python -m otg_mcp --help
```

※ README.mdの手順との差分
<https://github.com/h4ndzdatm0ld/otg-mcp> のREADMEに記載のインストール手順の"pip install -e ".[dev]" "では、必要な依存パッケージの一部が入らず、また互換性のない最新バージョンのパッケージがインストールされる問題があるため、requirements.txtを用いてインストールする。

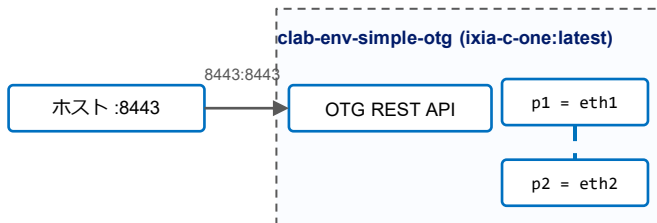
1. 環境構築：実行環境のdeploy

- **ixia-c-one 1コンテナ**をcontainerlabで起動して、OTG APIの疎通確認を行う

■ topology.yml (全文)

```
name: env-simple

topology:
  nodes:
    otg:
      kind: linux
      image: ghcr.io/open-traffic-generator/ixia-c-one:latest
      # localhost:8443 から OTG REST API を叩けるようホストにポート公開
      ports:
        - "8443:8443"      # OTG REST API
  links:
    - endpoints: ["otg:eth1", "otg:eth2"]
```



■ デプロイと成功時出力

```
# 左記topology.yml を任意のディレクトリに保存して
sudo containerlab deploy -t topology.yml

| clab-env-simple-otg | linux / ixia-c-one:latest | running | 172.20.20.x |
```

■ OTG API 疎通確認

```
curl -sk https://127.0.0.1:8443/capabilities/version
#正常時の出力例
{
  "api_spec_version": "1.*",
  "sdk_version": "1.43.0",
  "app_version": "1.43.0-7"
}
```

■ 片づける

```
sudo containerlab destroy -t topology.yml
```

1. 環境構築 : otg-mcpへの接続

- 接続先ターゲットを`otg-mcp-config.json`に、Claude CodeへのMCPサーバ登録を`mcp.json`に定義する

otg-mcp-config.json — ターゲット定義

```
{
  "targets": {
    "localhost:8443": {
      "ports": {
        "p1": {
          "location": "eth1",
          "name": "p1"
        },
        "p2": {
          "location": "eth2",
          "name": "p2"
        }
      }
    }
  }
}
```

手動起動での動作確認

```
.venv/bin/python -m otg_mcp ¥
--config-file otg-mcp-config.json ¥
--transport stdio
```

.mcp.json — MCP サーバ登録

```
{
  "mcpServers": {
    "otg": {
      "type": "stdio",
      "command": "/path/to/otg-demo/otg-mcp/.venv/bin/python",
      "args": [
        "-m", "otg_mcp",
        "--config-file", "/path/to/otg-demo/otg-mcp-config.json"
      ]
    }
  }
}
```

Claude Codeの場合の接続確認

Claude Codeの起動ディレクトリに上記の`mcp.json`を配置した後にClaude Codeを起動し、`/mcp`で正常に`otg-mcp`が接続されていることを確認する。

```
#statusがconnectedになっていることを確認する
> otg · ✓ connected · 12 tools
```

2.作業ディレクトリ：ディレクトリ構成

- エージェントが過去の実施結果を参照せずに、MCPのツールのみを用いて意図するトラフィックを生成できることを確認するため、最小構成のワークスペースを作り、その上で動作を確認した
- 各ファイルの詳細は以降のスライドで記載する

ディレクトリ構成

```
<作業ディレクトリ>/
├── state/
├── .mcp.json
└── .claude/
    ├── settings.local.json
    ├── skills/otg-demo/SKILL.md
    ├── hooks/mcp_log.sh
    └── hooks/session_save.sh
# 過去のclaudeとの対話履歴がないディレクトリ(例: /tmp/otg-demo/)
# 空 (current_demo.txt の置き場)
# MCP サーバ定義 (otg-mcp インストール先を絶対パス参照)
# model=sonnet / effortLevel=medium / 権限 / hook 登録
# /otg-demo の中身
# MCP 呼び出しログ自動保存
# session.jsonl 自動保存
```

2. 作業ディレクトリ : settings.local.json

- 使用モデルは**Claude sonnet (effortLevel: medium)**を利用し、permissionsでotg-mcpの12ツールと、デモディレクトリ作成・tcpdump・成果物書き込みに必要な最小コマンドのみを事前許可している

.claude/settings.local.json

```
{
  "model": "sonnet",
  "effortLevel": "medium",
  "permissions": {
    "allow": [
      "mcp_otg_get_schemas_for_target",
      "mcp_otg_get_config",
      "mcp_otg_list_schemas_for_target",
      "mcp_otg_set_config",
      "mcp_otg_health",
      "mcp_otg_get_metrics",
      "mcp_otg_get_available_targets",
      "mcp_otg_start_traffic",
      "mcp_otg_stop_traffic",
      "mcp_otg_start_capture",
      "mcp_otg_stop_capture",
      "mcp_otg_get_capture",
      "Bash(mkdir -p *)",
      "Bash(tcpdump *)",
      "Edit(**/config.json)",
      "Edit(**/prompt.md)",
      "Edit(**/current_demo.txt)"
    ]
  },
  "enableAllProjectMcpServers": true,
  "enabledMcpjsonServers": [
    "otg"
  ],
  "hooks": {右に記載}
}
```

hooks

```
"hooks": {
  "PostToolUse": [
    {
      "matcher": "mcp_otg_.*",
      "hooks": [
        {
          "type": "command",
          "command": "bash .claude/hooks/mcp_log.sh"
        }
      ]
    },
    {
      "matcher": "Stop",
      "hooks": [
        {
          "type": "command",
          "command": "bash .claude/hooks/session_save.sh"
        }
      ]
    }
  ]
}
```

2. 作業ディレクトリ : /otg-demo skill

- デモはデモ用のotg-demoスキルを用いて、基本的な進め方と、結果の記録方法をエージェントに与えて実行する

前半

```
# OTG Demo Skill

ユーザーからトラフィック生成リクエストを受け取ったときに呼び出す手順書です。

**目的**: 実際にトラフィックを流してキャプチャとメトリクスで動作を確認した上で、snappi / gosnappi で利用できる **検証済み OTG 設定 JSON** を生成する。

---

## ログ管理

### デモディレクトリの作成

リクエストを受け取ったら最初に実行する:

```bash
mkdir -p YYYYMMDD_HHMMSS_<概要> # 概要は英小文字・アンダースコア10文字以内
echo "YYYYMMDD_HHMMSS_<概要>" > state/current_demo.txt
```

`state/current_demo.txt` への書き込みはフックが保存先を解決するために必須。
```

後半

```
### 保存するファイル

ファイル	タイミング	内容
`prompt.md`	開始時	ユーザーの入力をそのまま保存
`capture_*.pcap`	キャプチャ取得後	`get_capture` の結果
`config.json`	検証 OK 後	検証済み OTG 設定 JSON

---

## キャプチャとメトリクスによる意図確認

トラフィック停止後、以下で意図通りかを確認する。 **検証 OK になるまで `config.json` を確定しない**。
```

```
### メトリクス確認

ユーザの意図するトラフィック定義とメトリクスが整合するかを確認する

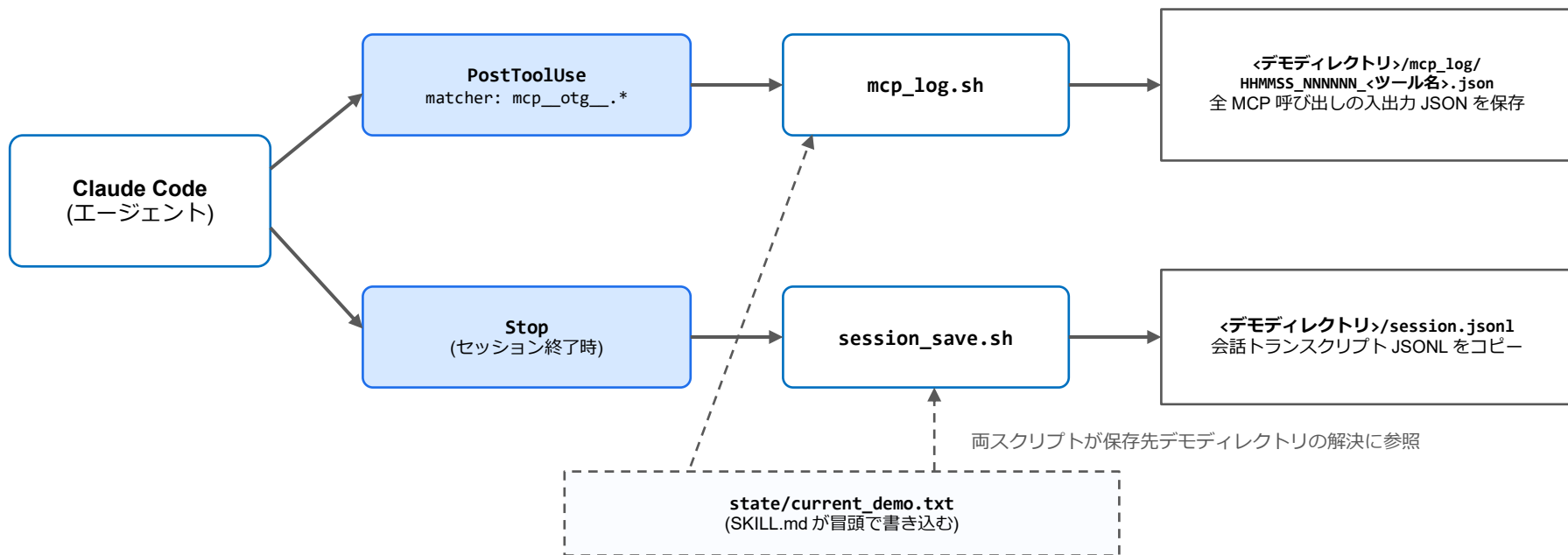
### キャプチャ確認

`get_capture` で取得した pcap を `tcpdump` などで確認し、ユーザーの指示通りのトラフィックが流れているかを確認する。

---
```

2. 作業ディレクトリ : hooks

- 結果の記録はエージェント任せにせず、Claude Codeのhooksで機械的にも保存する



2. 作業ディレクトリ : mcp_log.sh

- mcp_log.shの全文を記載する

■ mcp_log.sh

```
#!/bin/bash
# MCP tool input/output logger
# Called as PostToolUse hook; receives event JSON via stdin
set -euo pipefail
INPUT=$(cat)
TOOL_NAME=$(echo "$INPUT" | python3 -c "
import json, sys
d = json.load(sys.stdin)
print(d.get('tool_name', 'unknown'))
" 2>/dev/null || echo "unknown")
# Skip non-OTG tools
case "$TOOL_NAME" in
  mcp_otg_*);;
  *) exit 0 ;;
esac
# Determine demo directory from state file
STATE_FILE="state/current_demo.txt"
if [ ! -f "$STATE_FILE" ]; then
  exit 0
fi
DEMO_DIR=$(cat "$STATE_FILE" | tr -d '[:space:]')
if [ -z "$DEMO_DIR" ] || [ ! -d "$DEMO_DIR" ]; then
  exit 0
fi
MCP_LOG_DIR="$DEMO_DIR/mcp_log"
mkdir -p "$MCP_LOG_DIR"
TIMESTAMP=$(date +%H%M%S_%N | cut -c1-13)
OUTFILE="$MCP_LOG_DIR/${TIMESTAMP}_${TOOL_NAME#mcp_otg_}.json"
echo "$INPUT" | python3 -m json.tool > "$OUTFILE" 2>/dev/null || echo "$INPUT" > "$OUTFILE"
```

2. 作業ディレクトリ : session_save.sh

- session_save.shの全文を記載する

session_save.sh

```
#!/bin/bash
# Session transcript saver
# Called as Stop hook; copies the current session JSONL to the demo directory.
# The Stop event payload provides transcript_path, so no searching is needed.
set -euo pipefail
INPUT=$(cat)
# Not inside a demo run -> nothing to save (normal no-op)
STATE_FILE="state/current_demo.txt"
if [ ! -f "$STATE_FILE" ]; then
  exit 0
fi
DEMO_DIR=$(tr -d '[:space:]' < "$STATE_FILE")
if [ -z "$DEMO_DIR" ] || [ ! -d "$DEMO_DIR" ]; then
  exit 0
fi
# The Stop event tells us exactly which transcript this session is.
TRANSCRIPT=$(printf '%s' "$INPUT" | python3 -c 'import json, sys;
print(json.load(sys.stdin).get("transcript_path", ""))')
# Inside a demo but the transcript is missing -> a real failure, surface it.
if [ -z "$TRANSCRIPT" ] || [ ! -f "$TRANSCRIPT" ]; then
  echo "session_save: transcript_path を取得できませんでした (value='$TRANSCRIPT')" >&2
  exit 1
fi
cp "$TRANSCRIPT" "$DEMO_DIR/session.jsonl"
```

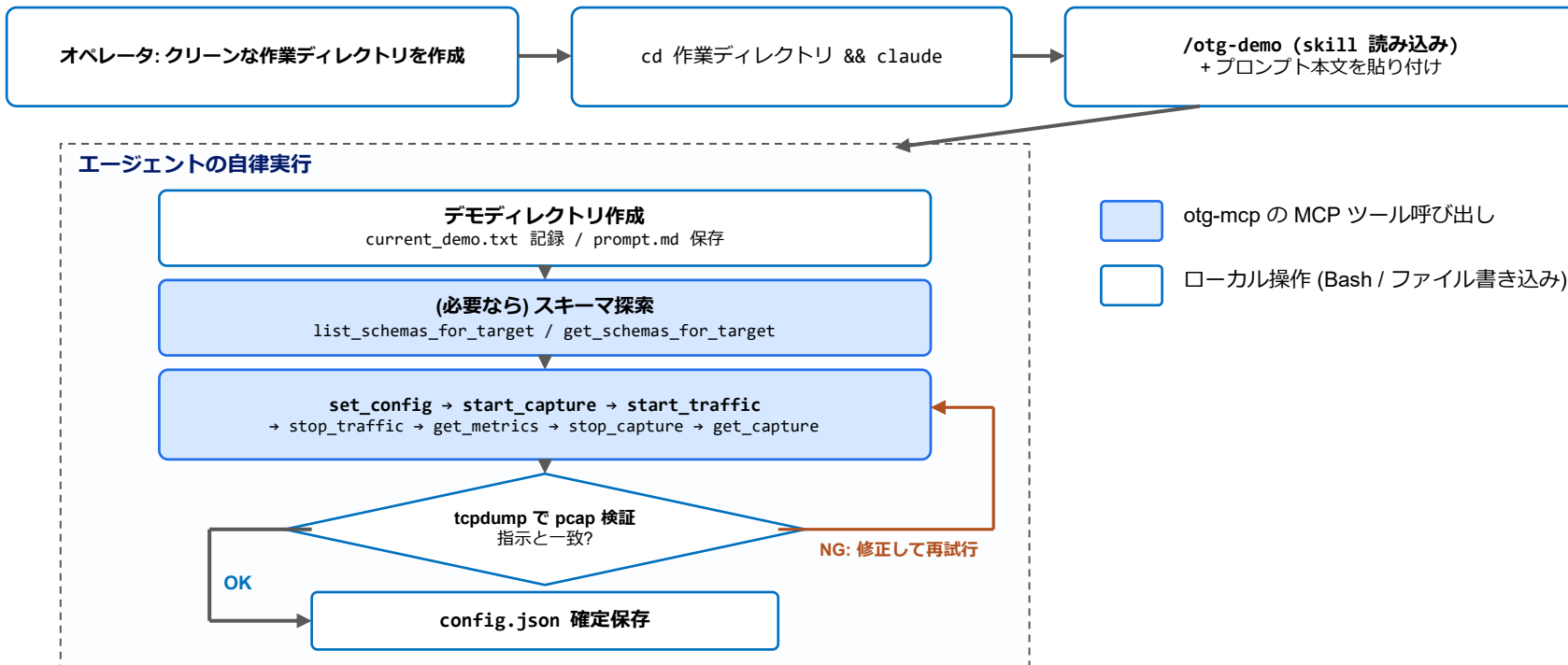
3. otg-mcp が提供する MCP ツール

- 以降の実施フロー・デモはotg-mcpの**全12ツール**の概要を一覧で示す

| ツール | 機能 |
|------------------------------|-----------------------------------|
| get_available_targets | ターゲット一覧と疎通・apiVersion の取得 |
| health | ターゲットのヘルスチェック |
| list_schemas_for_target | ターゲットの参照スキーマ (OTG データモデル) の一覧 |
| get_schemas_for_target | 特定スキーマ (例: Flow.Custom) の詳細取得 |
| set_config | OTG 設定 (ports/flows/captures) の投入 |
| get_config | 現在の OTG 設定の取得 |
| start_traffic / stop_traffic | トラフィック送信の開始 / 停止 |
| start_capture / stop_capture | パケットキャプチャの開始 / 停止 |
| get_capture | キャプチャ結果 (pcap) の取得 |
| get_metrics | ポート / フロー毎のメトリクス取得 |

4. 実施フロー — オペレータ操作と自律実行

- オペレータの操作は**3ステップのみ**。以降はpcap検証までエージェントが**自律実行**



5. デモ① : プロンプトと結果

- デモでのプロンプトと結果を記載する

■プロンプト

以下のパケットを p1 から p2 に流してください。

- プロトコル: UDP
- 送信元 MAC: 02:1a:2b:3c:4d:01
- 宛先 MAC: 02:1a:2b:3c:4d:02

■生成トラフィック

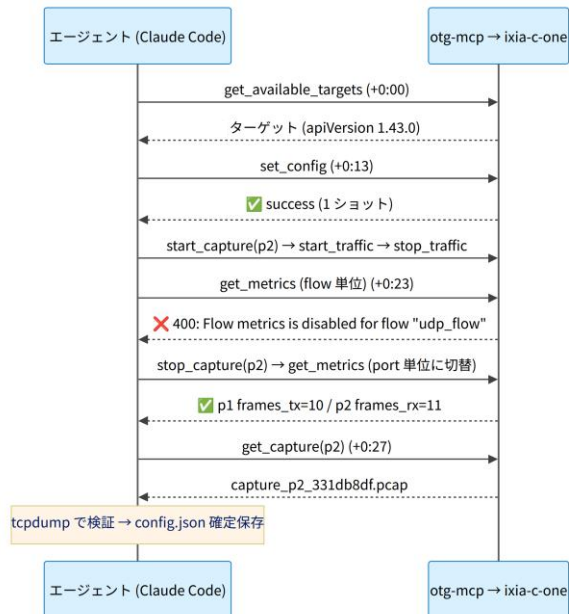
| フィールド | 値 | 区分 |
|------------------|---------------------------------------|----|
| Ethernet src→dst | 02:1a:2b:3c:4d:01 → 02:1a:2b:3c:4d:02 | 指定 |
| IPv4 src→dst | 192.168.1.1 → 192.168.1.2 | 補完 |
| UDP src→dst port | 1234 → 5678 | 補完 |
| Frame Size | 128 byte 固定 | 補完 |
| Rate | 100 pps | 補完 |
| Duration | fixed_packets: 10 | 補完 |

5. デモ① : エージェントの動作

- 所要時間は初めのツール呼び出しから、pcap取得まで約 27 秒、MCP呼び出し 9 回
- get_metrics(flow)でエラーが発生したが、portのメトリクスを取る方向に修正した

MCP 呼び出し時系列 (mcp_log/ より、+0:00 = 最初の

| 経過時間 | ツール | 内容 / 結果 |
|-------|-----------------------|---|
| +0:00 | get_available_targets | 利用可能なトラフィックジェネレータの一覧検出 (localhost:8443 apiVersion 1.43.0 等) |
| +0:13 | set_config | OTG 設定投入 — 1 発成功 (スキーマ探索なし) |
| +0:16 | start_capture | p2 でキャプチャ開始 |
| +0:18 | start_traffic | 送信開始 |
| +0:21 | stop_traffic | 送信停止 |
| +0:23 | get_metrics (flow) | ✗ Flow metrics is disabled for flow "udp_flow" (flow_metrics 未有効化のため失敗) |
| +0:23 | stop_capture | キャプチャ停止 |
| +0:26 | get_metrics (port) | ✓ port_metrics に切替えて再取得: p1 frames_tx=10 / p2 frames_rx=11 |
| +0:27 | get_capture | pcap 取得 |



5. デモ① : config.json

- 以下が実際に生成されたconfig.jsonで、このまま別のOTG API対応トラフィックジェネレータで利用可能

■ config.json — ports / captures / flows (ethernet まで) ■ (続き) ipv4 / udp / size / rate / duration

```
{
  "ports": [
    { "name": "p1", "location": "eth1" },
    { "name": "p2", "location": "eth2" }
  ],
  "captures": [
    {
      "name": "cap_p2",
      "port_names": ["p2"],
      "format": "pcap"
    }
  ],
  "flows": [
    {
      "name": "udp_flow",
      "tx_rx": {
        "choice": "port",
        "port": {
          "tx_name": "p1",
          "rx_names": ["p2"]
        }
      }
    }
  ],
  "packet": [
    {
      "choice": "ethernet",
      "ethernet": {
        "src": { "choice": "value", "value": "02:1a:2b:3c:4d:01" },
        "dst": { "choice": "value", "value": "02:1a:2b:3c:4d:02" }
      }
    }
  ],
}
```

```
    {
      "choice": "ipv4",
      "ipv4": {
        "src": { "choice": "value", "value": "192.168.1.1" },
        "dst": { "choice": "value", "value": "192.168.1.2" }
      }
    },
    {
      "choice": "udp",
      "udp": {
        "src_port": { "choice": "value", "value": 1234 },
        "dst_port": { "choice": "value", "value": 5678 }
      }
    }
  ],
  "size": { "choice": "fixed", "fixed": 128 },
  "rate": { "choice": "pps", "pps": 100 },
  "duration": {
    "choice": "fixed_packets",
    "fixed_packets": { "packets": 10 }
  }
}
]
```

6. デモ② : プロンプトと結果

- SRHはotg-mcp同梱のv1.30.0スキーマに無く、エージェントがFlow.Custom (custom.bytes)で代替して達成

■ 入カプロンプト (全文)

```
SRv6 で L3VPN を実現するパケットを生成してください。  
SR List = `[2001:db8:1::1, 2001:db8:2::1, 2001:db8:3::100]` (論理パス  
順)、  
内側は IPv4 + UDP のフレームを p1 から送信してください。
```

OTG仕様自体はv1.40.0以降Flow.Ipv6SegmentRoutingとしてネイティブ対応(otg-mcp同梱スキーマがv1.30.0のため、未対応と判断した)

■ 達成したプロトコルスタック

Ethernet

src=00:00:00:00:00:01 → dst=00:00:00:00:00:02, type=IPv6

IPv6

src=2001:db8::1 → dst=2001:db8:1::1
NextHdr=43 (Routing), HopLimit=64

SRH (RFC 8754) — custom.bytes 56 byte でエージェントが手組み

NextHdr=4 (IPIP), HdrExtLen=6, RoutingType=4,
SegmentsLeft=2, LastEntry=2, Flags=0, Tag=0
Segment[0]=2001:db8:3::100
Segment[1]=2001:db8:2::1
Segment[2]=2001:db8:1::1

IPv4

src=192.168.1.1 → dst=192.168.2.1, proto=17, ttl=64

UDP src=1234 → dst=5678

Payload (24 byte)

6. デモ② : エージェントの動作①

- 所要時間は初めのツール呼び出しからpcap取得まで約2分52秒、MCP呼び出し19回
- custom.bytesの記法とキャプチャポートの設定誤りで2回失敗しているが、自己修正している

■ MCP 呼び出し時系列 (mcp_log/ より、+0:00 = 最初の MCP 呼び出し)

| 経過時間 | アクション | 結果 / 学び |
|-------------|--|---|
| +0:00 | get_available_targets + health | 利用可能なトラフィックジェネレータの取得と、状態確認 |
| +0:11 | list_schemas_for_target (応答 45 KB) | 参照スキーマ (OTG v1.30.0) を総ざらいし SRH のネイティブ表現が無い ことを確認 |
| +0:26 | get_schemas_for_target (custom header 詳細) | custom.bytes (任意バイト列の直接指定) を拡張ヘッダの手段として確定 |
| +1:27 | set_config (1 回目) | ❌ 失敗 — hex 文字列にスペースを含めた。エラー: encoding/hex: invalid byte: U+0020 |
| +1:41 | set_config (2 回目) | ✅ 成功 (packets=100、キャプチャは cap_p1) |
| +1:45~+2:07 | start_capture → start_traffic → stop_traffic | 送信実行 |
| +2:07 | get_metrics | ✅ 100/100 送受信成功 (frames_tx=100, frames_rx=100, bytes=16600) |
| +2:10~+2:13 | stop_capture → get_capture(p1) | ⚠️ pcap が 0 byte — TX 側ポートではキャプチャされない |
| +2:34 | set_config (3 回目) | ✅ キャプチャを cap_p2 (RX 側) に切替、packets=10 に縮小して再走 |
| +2:37~+2:48 | start_capture → start_traffic → stop_traffic → get_metrics | ✅ 10/10 送受信成功 |
| +2:50~+2:52 | stop_capture → get_capture(p2) | ✅ pcap 1844 byte / 10 frames 取得完了 |

6. デモ② : エージェントの動作②

- シーケンス図と確認された工夫点を記載する



■ 自律的に解決した工夫点

① スキーマに無いプロトコルへの対処

参照スキーマに SRH の専用フィールドが無いことを 2 段階のスキーマ問い合わせで確認し、`custom.bytes` 拡張で代替する判断を自律的に下した

② バイナリレイアウトの自力組み立て

RFC 8754 の SRH フォーマット (HdrExtLen 計算、Segment List の逆順格納、Last Entry / Segments Left の関係) を正しく適用

③ hex エンコード仕様への適応

1 回目は人間可読のスペース区切り → `snappi` が拒否 → スペースを除去して再投入、というエラー → 修正サイクルを 1 ターンで完了

④ キャプチャポートの誤りからの自己修正

TX 側 (p1) で取った `pcap` が空であることを `get_capture` の結果から検出し、RX 側 (p2) に切替えて再実行

6. デモ② : SRH バイト列詳細

- custom.bytesで16進数表記で投入した値を記載する

■ エージェントが set_config に投入した値 (56 byte)

```
04 06 04 02 02 00 00 00
20 01 0d b8 00 03 00 00 00 00 00 00 00 01 00 : Address[0] 2001:db8:3::100
20 01 0d b8 00 02 00 00 00 00 00 00 00 01 : Address[1] 2001:db8:2::1
20 01 0d b8 00 01 00 00 00 00 00 00 00 01 : Address[2] 2001:db8:1::1
```

■ バイト位置の意味

| バイト位置 | 値 | 意味 |
|---------|-----------|-----------------------------|
| [0] | 0x04 | Next Header: IPIP (4) |
| [1] | 0x06 | Length: 6 [Length: 56bytes] |
| [2] | 0x04 | Type: Segment Routing (4) |
| [3] | 0x02 | Segments Left: 2 |
| [4] | 0x02 | Last Entry: 2 |
| [5] | 0x00 | Flags: 0x00 |
| [6..7] | 0x0000 | Tag: 0000 |
| [8..55] | 16×3 byte | Address[0]~[2] |

6. デモ② : config.json

- 以下が確定保存された config.json で、このまま別の OTG API 対応トラフィックジェネレータで利用可能

■ (続き) packet: ipv4 / udp

```
{
  "choice": "ipv4",
  "ipv4": {
    "src": {"choice": "value", "value": "192.168.1.1"},
    "dst": {"choice": "value", "value": "192.168.2.1"},
    "protocol": {"choice": "value", "value": 17},
    "time_to_live": {"choice": "value", "value": 64}
  }
},
{
  "choice": "udp",
  "udp": {
    "src_port": {"choice": "value", "value": 1234},
    "dst_port": {"choice": "value", "value": 5678}
  }
}
],
```

■ (続き) size / rate / duration / metrics

```
"size": {"choice": "fixed", "fixed": 128},
"rate": {"choice": "pps", "pps": 10},
"duration": {"choice": "fixed_packets", "fixed_packets": {"packets": 100}},
"metrics": {"enable": true}
}
]
}
```

付録: 実入出力例 (1/3) 探索系 ①

- 以降に実際にツールを利用した際の入出力を一部抜粋して記載する
- 接続先と状態確認用の2ツール（get_available_targets/health）を記載する

A.1 get_available_targets

定義された全ターゲットへ実際に接続し、ポート定義・疎通可否・apiVersion を返す。
入力: {} (引数なし)

```
{
  "localhost:8443": {
    "ports": {"p1": {"location": "eth1", "name": "p1"},
             "p2": {"location": "eth2", "name": "p2"}},
    "available": true,
    "apiVersion": "1.43.0"
  }
}
```

A.2 health

各ターゲットの死活と版数。入力: {} (引数なし)

```
{
  "status": "healthy",
  "targets": {
    "localhost:8443": {"name": "localhost:8443", "healthy": true,
                      "version_info": {"api_spec_version": "1.*",
                                       "sdk_version": "1.43.0",
                                       "app_version": "1.43.0-7"},
                      "error": null}
  }
}
```

付録: 実入出力例 (1/3) 探索系 ②

- スキーマ探索系の 2 ツール (list_schemas_for_target/get_schemas_for_target) を記載する

A.3 list_schemas_for_target

対象が解釈できるスキーマ名の列挙

```
{
  "target_name": "localhost:8443"
}
```

```
["Config", "Config.Options", "Port", "Port.Options", "Lag", ...,
 "Flow", "Flow.TxRx", "Flow.Port", "Flow.Header",
 "Flow.Custom", "Flow.Custom.MetricTag",
 "Flow.Ethernet", "Flow.Vlan", "Flow.Vxlan",
 "Flow.Ipv4", "Flow.Ipv4.Priority", "Flow.Ipv4.Dscp",
 "Flow.Ipv6", "Flow.Ipv6.Auto",
 "Flow.Tcp", "Flow.Udp", "Flow.Gre", "Flow.Gtpv1", "Flow.Gtpv2",
 "Flow.Arp", "Flow.Icmp", "Flow.Icmpv6", "Flow.Mpls", "Flow.Snmpv2c", ...]
```

A.4 get_schemas_for_target

指定スキーマの JSON Schema 定義を取得

```
{
  "target_name": "localhost:8443",
  "schema_names": ["Flow.Custom", "Flow.Ipv6", "Flow.Ipv4", "Flow.Udp"]
}
```

```
{
  "Flow.Custom": {
    "type": "object",
    "description": "Custom packet header",
    "required": ["bytes"],
    "properties": {
      "bytes": {
        "description": "A custom packet header defined as a string of hex bytes. The string MUST contain sequence of valid hex bytes. Spaces or colons can be part of the bytes but will be discarded. ...",
        "type": "string",
        "pattern": "^[A-Fa-f0-9: ]+$"
      },
      ...
    }
  },
  "Flow.Ipv6": { "description": "IPv6 packet header", "properties": { "next_header": ..., "src": ..., "dst": ..., ... } },
  "Flow.Ipv4": { ... },
  "Flow.Udp": { ... }
}
```

付録: 実入出力例 (2/3) 設定系 ①

- configの投入用のツール (set_config) の入出力例を記載する
- 出力は入力で省略していたフィールドが既定値で補完されたものが返ってくる

■ A.5 set_config — 成功例 (入力 抜粋)

```
{
  "target": "localhost:8443",
  "config": {
    "ports": [{ "name": "p1", "location": "eth1",
                { "name": "p2", "location": "eth2"}],
    "captures": [{ "name": "cap_p2", "port_names": ["p2"], "format": "pcap"}],
    "flows": [{
      "name": "udp_flow",
      "tx_rx": { "choice": "port", "port": { "tx_name": "p1", "rx_names": ["p2"] } },
      "packet": [ ...ethernet / ipv4 / udp (デモ①のconfig.json と同内容) ... ],
      "size": { "choice": "fixed", "fixed": 128 },
      "rate": { "choice": "pps", "pps": 100 },
      "duration": { "choice": "fixed_packets", "fixed_packets": { "packets": 10 } }
    } ]
  }
}
```

補完された既定値の例:

```
ether_type=auto (65535) / header_checksum=generated "good" /
duration.gap=12 / payload=pattern "00" repeat
```

■ 同一 応答 (正規化済み config 抜粋)

```
{
  "status": "success",
  "config": {
    "ports": [...],
    "captures": [{ "port_names": ["p2"], "overwrite": true, "format": "pcap", "name":
"cap_p2"}],
    "flows": [{
      "name": "udp_flow",
      "packet": [
        { "choice": "ethernet", "ethernet": { ...,
          "ether_type": { "choice": "auto", "auto": 65535 } },
          { "choice": "ipv4", "ipv4": { ...,
            "header_checksum": { "choice": "generated", "generated": "good", ... } },
            { "choice": "udp", "udp": { ...,
              "checksum": { "choice": "generated", "generated": "good" } }
          } ],
      "duration": { "choice": "fixed_packets", "fixed_packets": { "packets": 10, "gap": 12 } },
      "payload": { "choice": "fixed", "fixed": { "pattern": "00", "repeat": true }, ...
    } ]
  }
}
```

付録: 実入出力例 (2/3) 設定系 ②

- 設定投入ツール (set_config) の失敗例と設定取得ツール (get_config) の入出力例を記載する

A.5 set_config — 失敗例 (入力 抜粋)

```
{
  "choice": "custom",
  "custom": {
    "bytes": "04 06 04 02 02 00 00 00 20 01 0d b8 00 03 00 00 00 00 00 01
00 20 01 0d b8 00 02 ..."
  }
}
```

上記のconfigに対するエラー応答

```
{
  "status": "error",
  "config": {
    "error": "code: 500\nerrors:\n- 'Error occurred while setting Traffic config (Layer1
only) for user common:Error\n setting frame for flow ¥"srv6_l3vpn¥":Could not add
Custom: invalid custom bytes:\n encoding/hex: invalid byte: U+0020 ' ' ¥\nkind:
internal\n"
  }
}
```

U+0020 = スペース。スペース無し hex (112 文字) への修正で成功

A.6 get_config

デモでは未使用のため、別途取得

```
{
  "target": "localhost:8443"
}
```

```
{
  "status": "success",
  "config": {
    "ports": [{"location": "eth1", "name": "p1"}, {"location": "eth2", "name": "p2"}],
    "captures": [{"port_names": ["p2"], "overwrite": true, "format": "pcap", "name":
"cap_p2"}],
    "flows": [{"name": "udp_flow",
      "packet": [ ..ethernet/ipv4/udp (既定値補完済) ... ],
      "size": {"choice": "fixed", "fixed": 128},
      "rate": {"choice": "pps", "pps": "100"},
      "duration": {"choice": "fixed_packets", "fixed_packets": {"packets": 10,
"gap": 12}}}],
    "options": {"port_options": {"location_preemption": false, "data_integrity": false}}
  }
}
```

付録: 実入出力例 (3/3) 実行・観測系 ①

- 実行系 4 ツール (start_traffic/stop_traffic/start_capture/stop_capture)を記載する

A.7 start_traffic

"Start traffic generation."

```
{
  "target": "localhost:8443"
}
```

```
{"status": "success", "action": "traffic_generation", "verified": null,
 "timestamp": "2026-05-08T11:57:18.210784", "result": null}
```

A.8 stop_traffic

"Stop traffic generation."

```
{
  "target": "localhost:8443"
}
```

```
{"status": "success", "action": "traffic_generation", "verified": null,
 "timestamp": "2026-05-08T11:57:21.240466", "result": {"verified": true}}
```

A.9 start_capture

"Start packet capture on a port."

```
{
  "target": "localhost:8443",
  "port_name": "p2"
}
```

```
{"status": "success", "port": "p2", "data": null, "capture_id": null, "file_path": null}
```

A.10 stop_capture

"Stop packet capture on a port."

```
{
  "target": "localhost:8443",
  "port_name": "p2"
}
```

```
{"status": "success", "port": "p2",
 "data": {"status": "stopped", "warnings": []},
 "capture_id": null, "file_path": null}
```

- キャプチャ取得ツール (get_capture) とメトリクス取得ツール (get_metrics) の入出力例を記載する

A.11 get_capture

"Get packet capture from a port and save it to a file. The capture data is saved as a .pcap file that can be opened with tools like Wireshark."

```
{
  "target": "localhost:8443",
  "port_name": "p2",
  "output_dir": "/path/to/output_dir"
}
```

```
{
  "status": "success",
  "port": "p2",
  "data": {"status": "captured",
    "file_path": "/path/to/output_dir/capture_p2_59993541.pcap"},
  "capture_id": "capture_p2_59993541.pcap",
  "file_path": "/path/to/output_dir/capture_p2_59993541.pcap"
}
```

A.12 get_metrics (フロー)

flow_names → フローのメトリクスを取得 / **port_names** → ポートのメトリクスを取得。フローのメトリクスを取得する場合は set_config 時に flows[].metrics.enable: true が必要 (デモ④では未設定だったため 400 エラー → ポートのメトリクスを取得へ切替)

```
{
  "target": "localhost:8443",
  "flow_names": ["srv6_l3vpn"]
}
```

```
{
  "status": "success",
  "metrics": {
    "choice": "flow_metrics",
    "flow_metrics": [{
      "name": "srv6_l3vpn", "transmit": "stopped",
      "frames_tx": "100", "frames_rx": "100",
      "bytes_tx": "16600", "bytes_rx": "16600",
      "frames_tx_rate": 0, "frames_rx_rate": 0,
      "tx_rate_bps": 0, "rx_rate_bps": 0
    }]
  }
}
```

- メトリクス取得ツール (get_metrics) の入出力例を記載する

■ A.12 get_metrics (ポート)

```
{  
  "target": "localhost:8443",  
  "port_names": ["p1", "p2"]  
}
```

```
{  
  "status": "success",  
  "metrics": {  
    "choice": "port_metrics",  
    "port_metrics": [  
      {  
        "name": "p1", "location": "172.17.0.4:5555;1", "link": "up",  
        "capture": "stopped", "frames_tx": "10", "frames_rx": "1",  
        "bytes_tx": "1280", "bytes_rx": "74", "transmit": "stopped",  
        "frames_tx_rate": 0, "frames_rx_rate": 0, "bytes_tx_rate": 0, "bytes_rx_rate": 0},  
      {  
        "name": "p2", "location": "172.17.0.3:5555;1", "link": "up",  
        "capture": "stopped", "frames_tx": "0", "frames_rx": "11",  
        "bytes_tx": "0", "bytes_rx": "1354", "transmit": "stopped",  
        "frames_tx_rate": 0, "frames_rx_rate": 0, "bytes_tx_rate": 0, "bytes_rx_rate": 0}  
    ]  
  }  
}
```