443

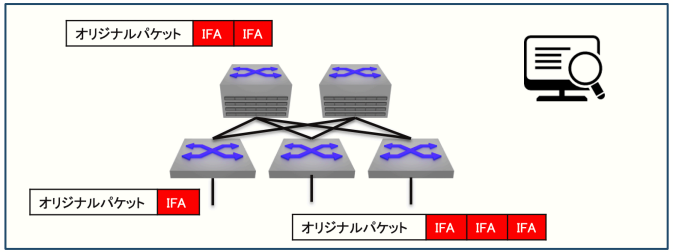
Shishio Tsuchiya
shtsuchi@arista.com

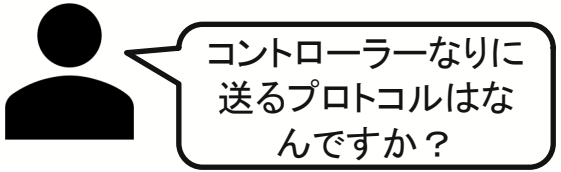
ARISTA

本日の概要

- COVID19の影響もあり、JANOG46では昨今のトラフィックトレンドを見る様なプログラムが多数ありました。
  その中でポート番号443を使うHTTPS/QUICの話題も多かった様に感じます。
  本プログラムではHTTP/2をデフォルトで使うgRPCの使用例およびその拡張であるgNMI/gRIBI/gNOIの概要、さらにHTTP/3やQUICを使ったVPNサービスMASQUEのIETFでのステータスに関してお話したいと思います。

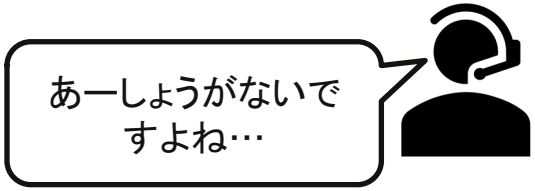
JANOG46にて

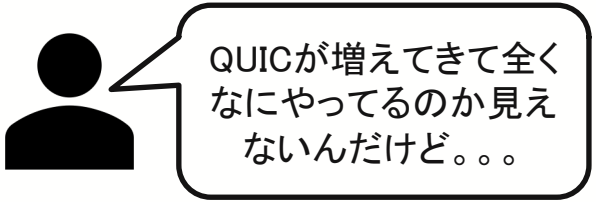




- 普通ってなんだ

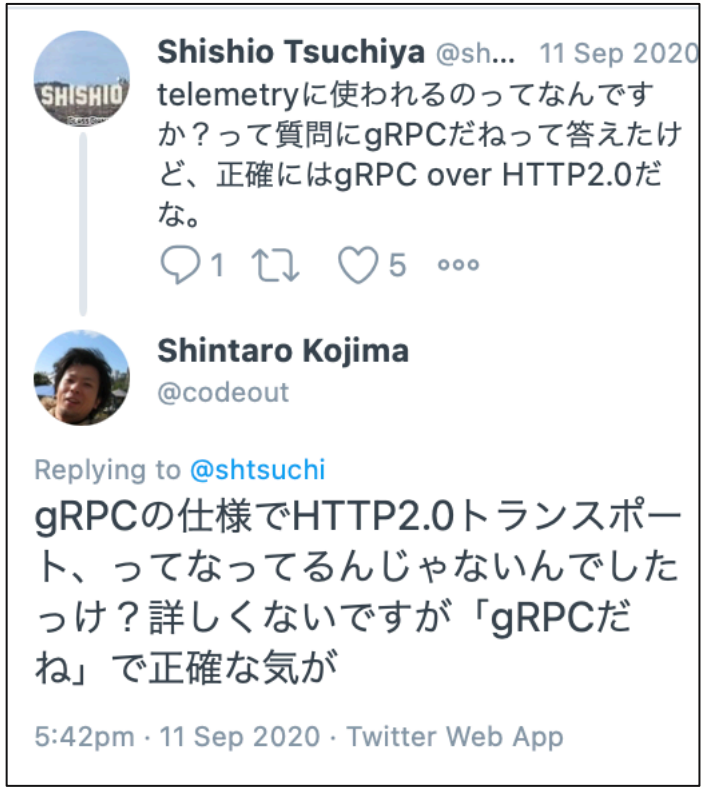
JANOG46にて





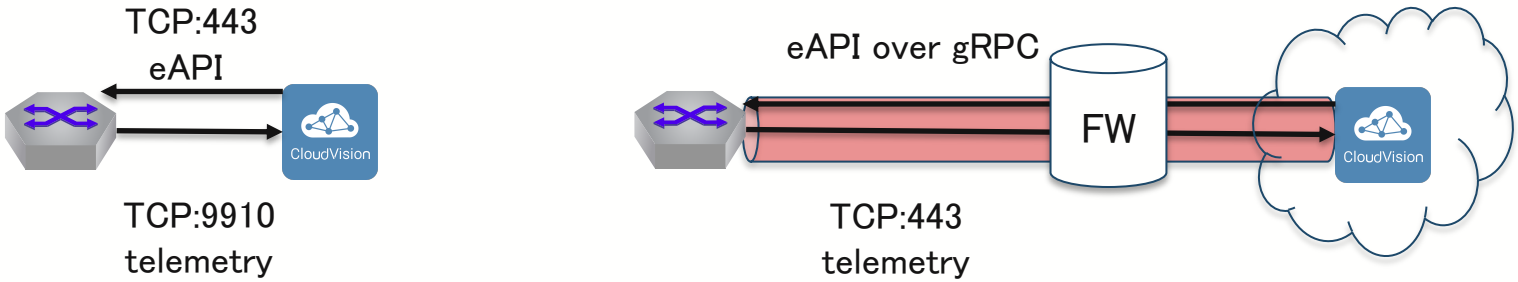
- しょうがないってなんだ

JANOG46にて



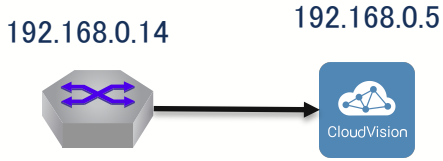
- 確かに…
- 言葉って難しい

背景



- gRPCは当初よりサポートしてたが、設定変更などを行うeAPIとの兼ね合いもあり、9910でポートを分けて設定
- クラウドサービスも始めた為、本来のgRPC(HTTP2)のポート番号である443のみでも運用出来る様にgRPC上でeAPIも実行出来る様に実装を変更

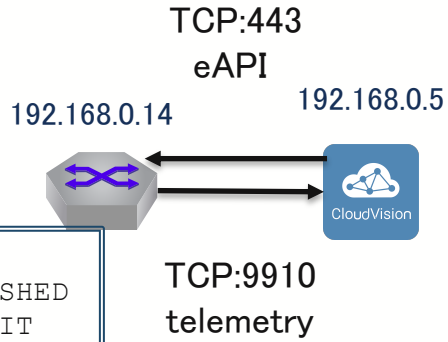
デモンストレーション




```
leaf1#bash netstat -n | grep 192.168.0.5
tcp        0      0 192.168.0.14:44090      192.168.0.5:9910        ESTABLISHED
```

- 通常時はtelemetryの為のポート 9910(実際はgRPC)

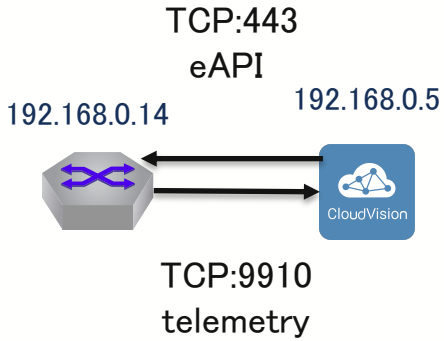
デモンストレーション




```
leaf1#bash netstat -n | grep 192.168.0.5
tcp        0     38 192.168.0.14:44090      192.168.0.5:9910        ESTABLISHED
tcp6       0      0 192.168.0.14:443        192.168.0.5:52224       TIME_WAIT
tcp6       0      0 192.168.0.14:443        192.168.0.5:52232       TIME_WAIT
tcp6       0      0 192.168.0.14:443        192.168.0.5:52236       TIME_WAIT
tcp6       0      0 192.168.0.14:443        192.168.0.5:52240       TIME_WAIT
tcp6       0      0 192.168.0.14:443        192.168.0.5:52244       TIME_WAIT
leaf1#
```

- 稼働中のコンフィグをCVPから見に行く
- HTTPsサーバーとして動作しているleaf1の443番ポートに接続しにいく

デモンストレーション

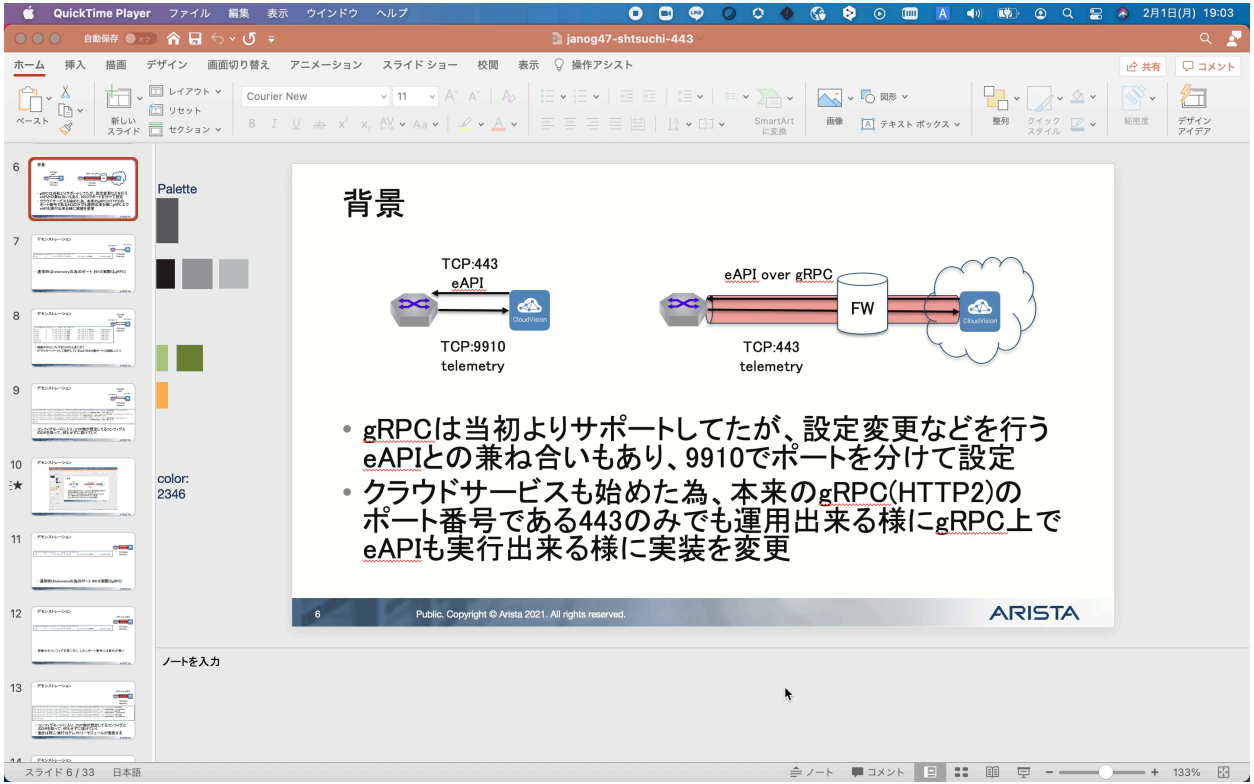



```
Jan 24 03:57:55 leaf1 ConfigAgent: %SYS-5-CONFIG_SESSION_ENTERED: User arista entered configuration
session capiVerify-1975-568435205df811eb846902904dc9ad89 on command-api (192.168.0.5)
Jan 24 03:57:57 leaf1 ConfigAgent: %SYS-5-CONFIG_SESSION_EXITED: User arista exited configuration session
capiVerify-1975-568435205df811eb846902904dc9ad89 on command-api (192.168.0.5)
Jan 24 03:57:57 leaf1 ConfigAgent: %SYS-5-CONFIG_SESSION_ABORTED: User arista aborted configuration
session capiVerify-1975-568435205df811eb846902904dc9ad89 on command-api (192.168.0.5)
leaf1#
```

- コンフィグモードに入り、CVP側が想定してるコンフィグとのDiffを取って、何もせずに抜けていく

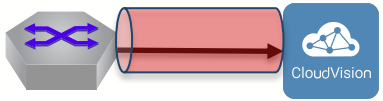
デモンストレーション

https://youtu.be/grEZZQ5pJIA



ARISTA

デモンストレーション

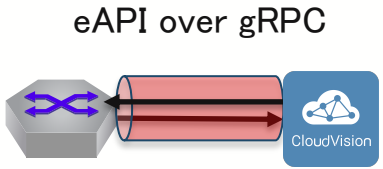


TCP:9910
telemetry

```
cvp-lf-21#bash netstat -n | grep 10.90.165.59
tcp        0     39 10.90.165.21:39420      10.90.165.59:9910       ESTABLISHED
```

- 通常時はtelemetryの為のポート 9910(実際はgRPC)

デモンストレーション

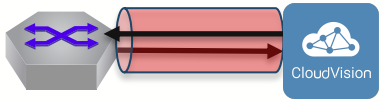



```
cvp-lf-21#bash netstat -n | grep 10.90.165.59
tcp        0     39 10.90.165.21:39420      10.90.165.59:9910       ESTABLISHED
```

- 稼働中のコンフィグを見に行く、しかしポート番号には変化が無い

デモンストレーション

eAPI over gRPC



TCP:9910
telemetry

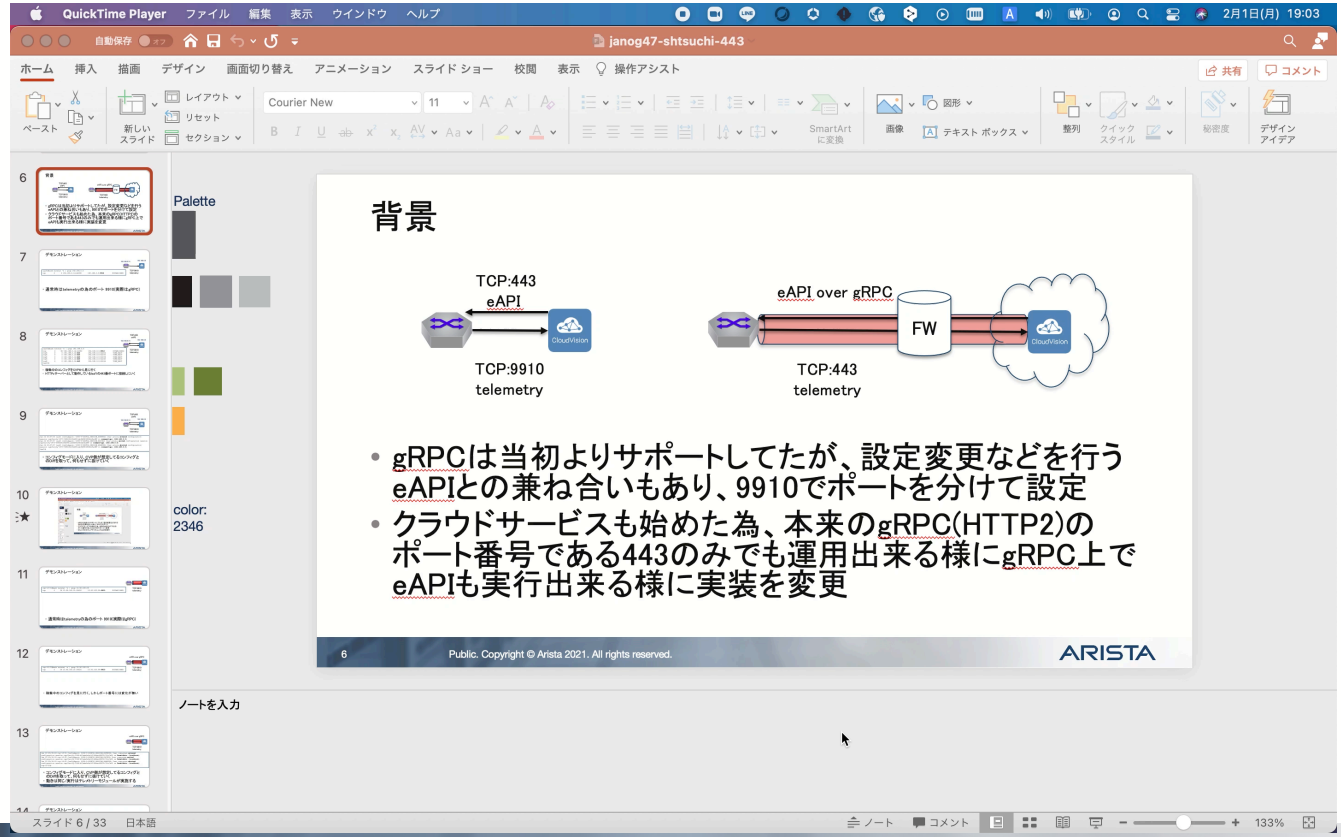
```
Jan 23 23:35:18 cvp-lf-21 ConfigAgent: %SYS-5-CONFIG_SESSION_ENTERED: User cvpsystem entered
configuration session capiVerify-1566-b53eda8e5e1611ebbe46001c731e7b03 on TerminAttr (localhost)
Jan 23 23:35:33 cvp-lf-21 ConfigAgent: %SYS-5-CONFIG_SESSION_EXITED: User cvpsystem exited
configuration session capiVerify-1566-b53eda8e5e1611ebbe46001c731e7b03 on TerminAttr (localhost)
Jan 23 23:35:33 cvp-lf-21 ConfigAgent: %SYS-5-CONFIG_SESSION_ABORTED: User cvpsystem aborted
configuration session capiVerify-1566-b53eda8e5e1611ebbe46001c731e7b03 on TerminAttr (localhost)
cvp-lf-21#
```

- コンフィグモードに入り、CVP側が想定してるコンフィグとのDiffを取って、何もせずに抜けていく
- 動きは同じ/実行はテレメトリーモジュールが実施する

ARISTA

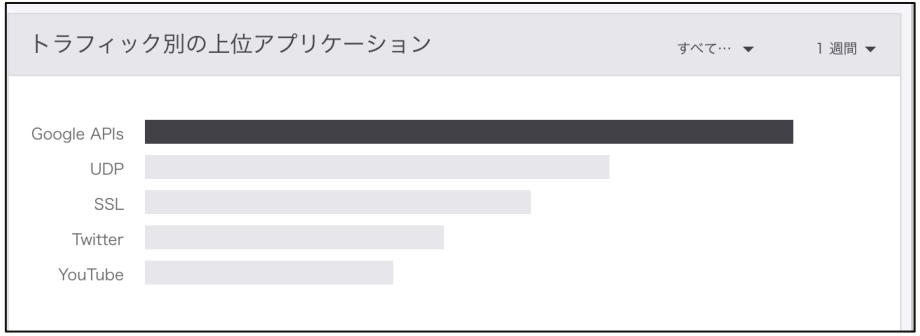
デモンストレーション

https://youtu.be/s1rHXtof-Tg

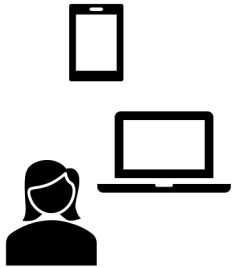


ARISTA

背景



会社員
Zoom/Youtube/Slack/
Twitterなど



大学生
Zoom/Youtube/Twitterなど

高校生スマホ
Youtube/Twitterなど

高校生 Chrome book
UDP:443

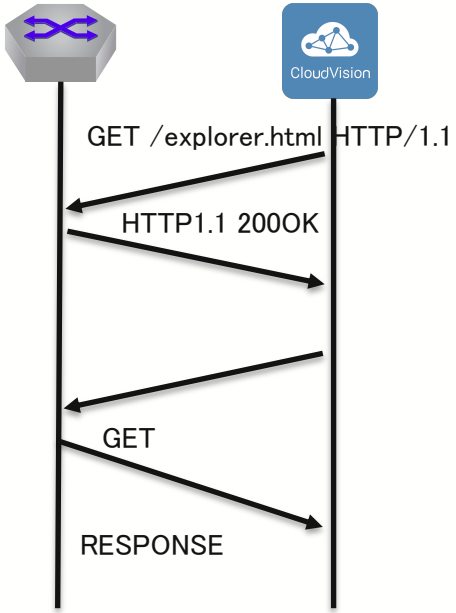
- とある日の在宅勤務
- WIPSでトラフィックを観測
- UDP443 QUICしか吐かない端末が…でもZoomっぽい事もやってる

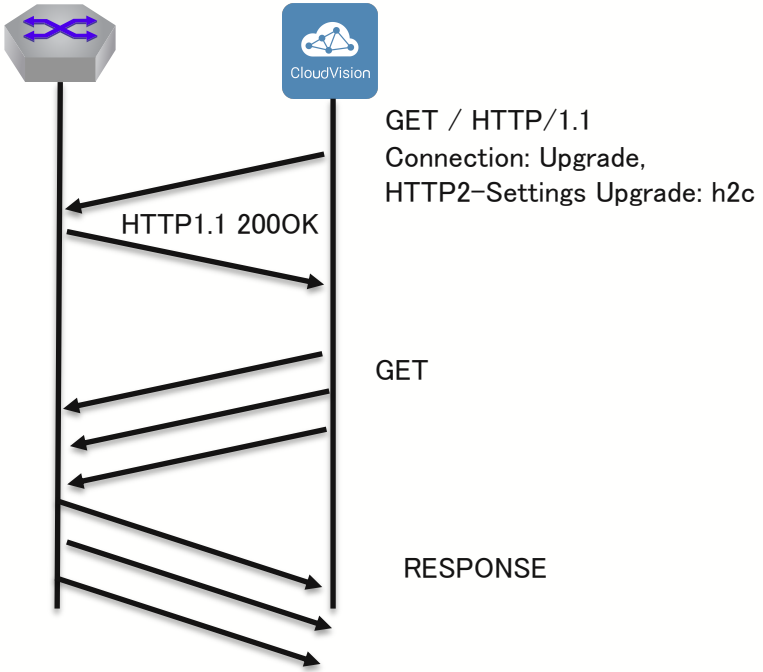
ARISTA

HTTP/2

- GoogleがSPDYというプロトコルを開発
- IETFでRFC7540を2015年に標準化
  - 1.1->2へ16年ぶりの改訂
- メソッドは同じ
- テキストベースからバイナリーベースへ
- ヘッダー圧縮
- ストリームの多重化
- ストリームの優先度
- ヘッダー圧縮
- フロー制御

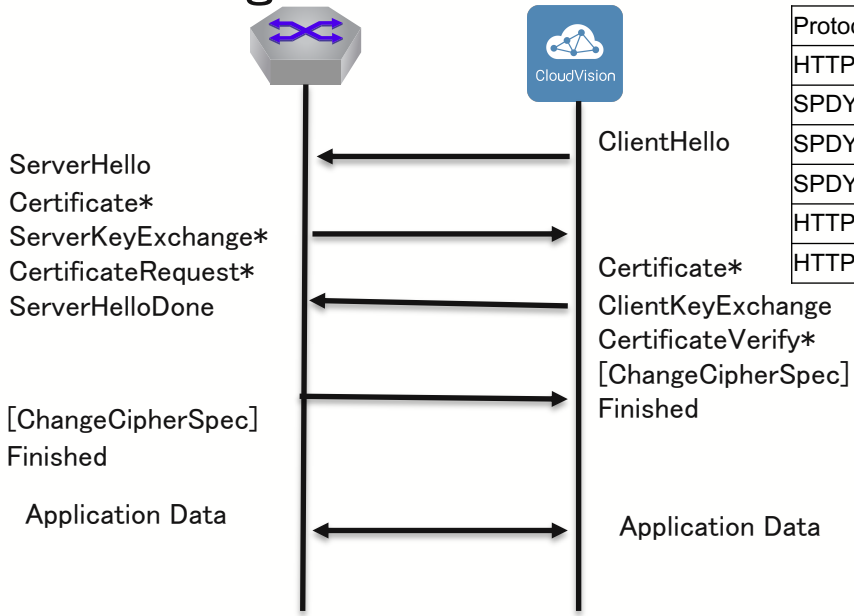
RFC7540 HTTP/2





- 手法はhttp1.1と同等であるためにクライアントでUpgradeリクエストで始まる（RFC7230）
- ストリームの多重化がHTTP/2では可能

RFC7301 Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension



Protocol	Identification Sequence
HTTP/1.1	0x68 0x74 0x74 0x70 0x2f 0x31 0x2e 0x31 ("http/1.1")
SPDY/1	0x73 0x70 0x64 0x79 0x2f 0x31 ("spdy/1")
SPDY/2	0x73 0x70 0x64 0x79 0x2f 0x32 ("spdy/2")
SPDY/3	0x73 0x70 0x64 0x79 0x2f 0x33 ("spdy/3")
HTTP/2 over TLS	0x68 0x32 ("h2")
HTTP/2 over TCP	0x68 0x32 0x63 ("h2c")

- 実際のフレームはTLSで暗号化されている

ARISTA

gRPC

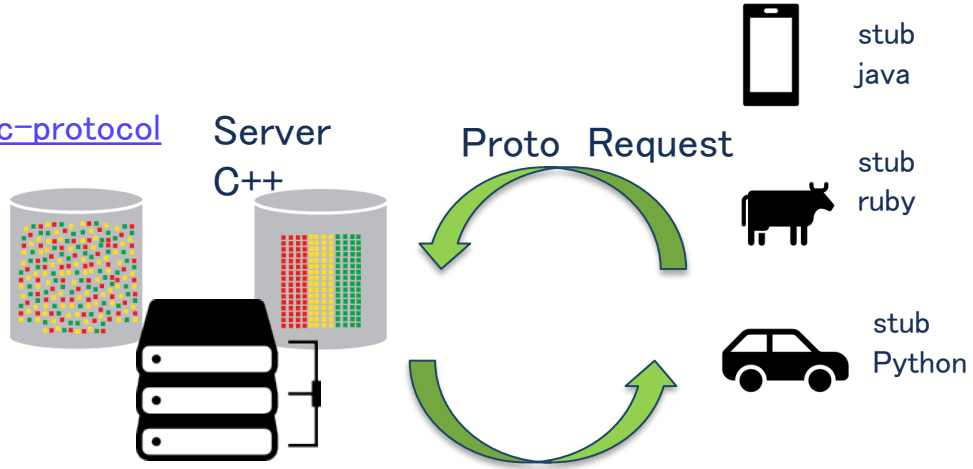
- GoogleがStubbyと呼ばれる汎用RPCインフラを開発し、10年以上運用
- 2015年にオープンソースへgRPCとなる
- 下記をプラグインでサポート
  - 負荷分散
  - トレーサビリティ
  - ヘルスチェック
  - 認証

https://grpc.io/

- Protocol Bufferをインターフェース定義言語およびメッセージ交換フォーマットとして使用可能
- C/Python/Goといった多くのプログラム言語ライブラリをサポート
- HTTP/2トランスポートによる双方向のストリーミング

gRPC

- gRPCでは、クライアントアプリケーションが別のマシン上のサーバーアプリケーション上のメソッドをローカルオブジェクトのように直接呼び出すことができる
- 分散型のアプリケーションやサービスを簡単に作成可能
- デフォルトでProtocol Bufferと動作
  - https://developers.google.com/protocol-buffers
- gRPC Protocol Specification
  - https://tools.ietf.org/html/draft-kumar-rtgwg-grpc-protocol
  - https://github.com/grpc/



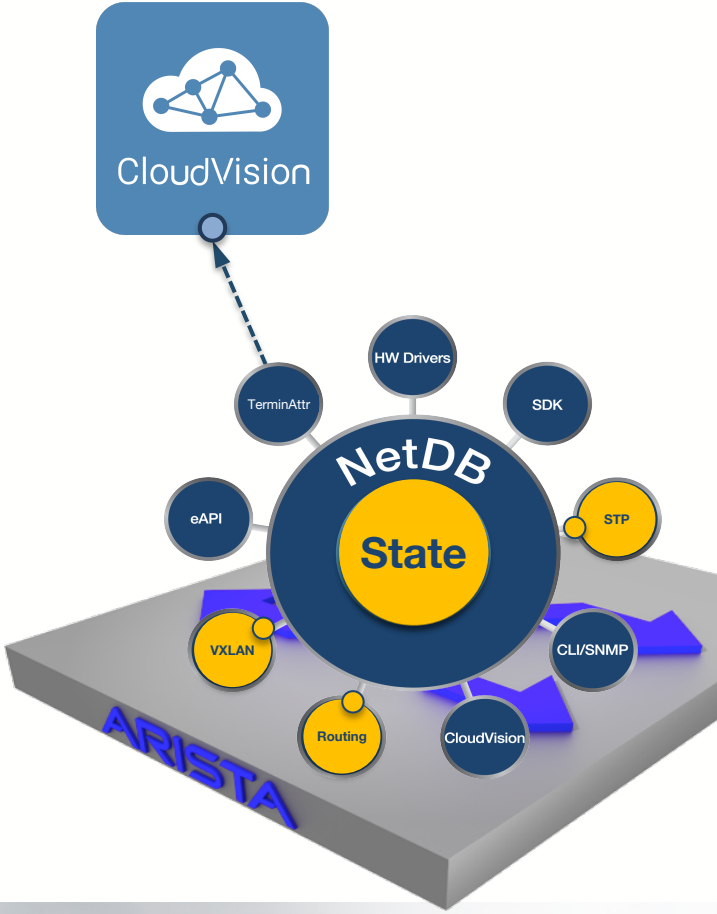
Use cases for gRPC in network management

https://tools.ietf.org/html/draft-talwar-rtgwg-grpc-use-cases

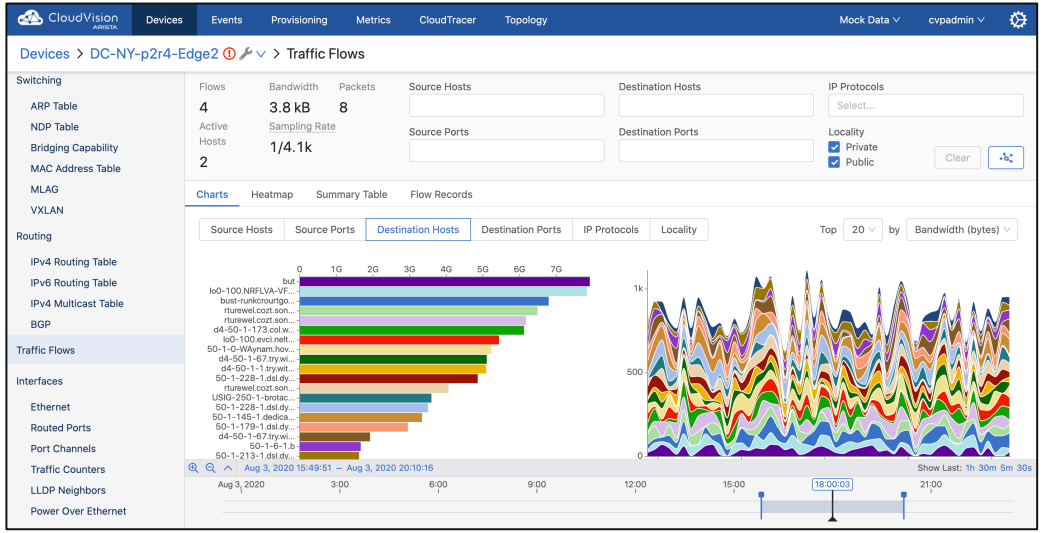
- ネットワークマネージメント
  - ストリーミングテレメトリー
  - ネットワーク設定管理
    - ✓ ベンダー独自プロトコルやNETCONF/TL1などの標準プロトコルに変わる近代的なオープンな手法
    - ✓ 柔軟なデータ構造/多くのプログラム言語サポート/セキュア
  - 既にOpenConfig/YANGベースのデータモデルを使った多くの実装が存在
    - ✓ https://github.com/aristanetworks/goarista/tree/master/cmd/gnmi
    - ✓ https://github.com/CiscoDevNet/grpc-getting-started
    - ✓ https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000632-en.pdf

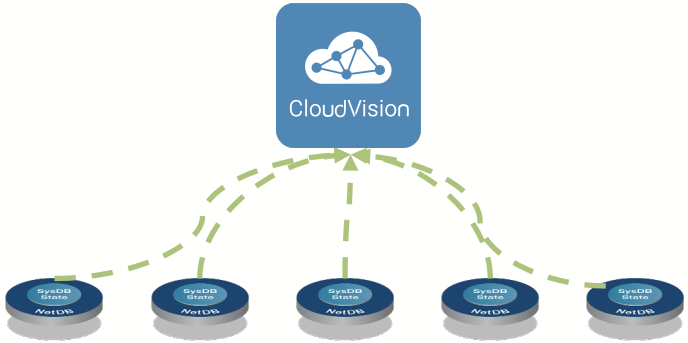
ストリーミングテレメトリエージェント – TerminAttr

- 新しいエージェント
- Goでプログラミング
- EOSの様々なデータソースと接続
- ミリ秒以内でスイッチのそのままの状態をツリー構造またはストリーミングイベントとして提示
- 状態に変化があればストリーミングでアップデート
- CloudVision Telemetry CVP 2017.1とEOS-4.18.1Fにエクステンションとして組み込まれてサポート
- エクステンションをインストールすればこれより前のバージョンでもサポート
  - 全てのSysDBステート(/Sysdb/cell/*を除く)
  - 全てのSMASHテーブル
  - CPUやメモリ使用量などプロセスとKernelデータ
  - システムLogメッセージ



トラフィックフロー over テレメトリー





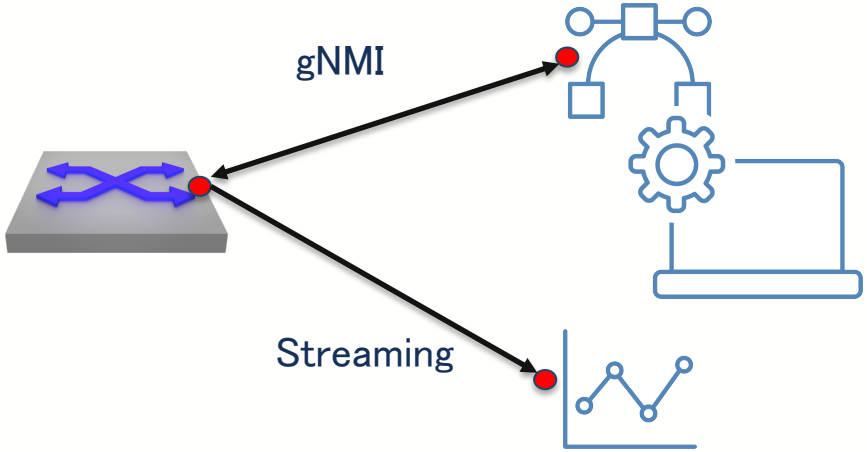

```
sflow destination 127.0.0.1
sflow source-interface Management1
sflow run
```

- sFlow/IPFIXでExportされたトラフィックフローデータをCVPで表示可能
- フローコネクターにCVPがなるわけでなく、フローをローカルに転送、TerminAttrがフロー情報をテレメトリーデータとして変換し、コレクターに送る

gRPC Network Management Interface (gNMI)

https://tools.ietf.org/html/draft-openconfig-rtgwg-gnmi-spec

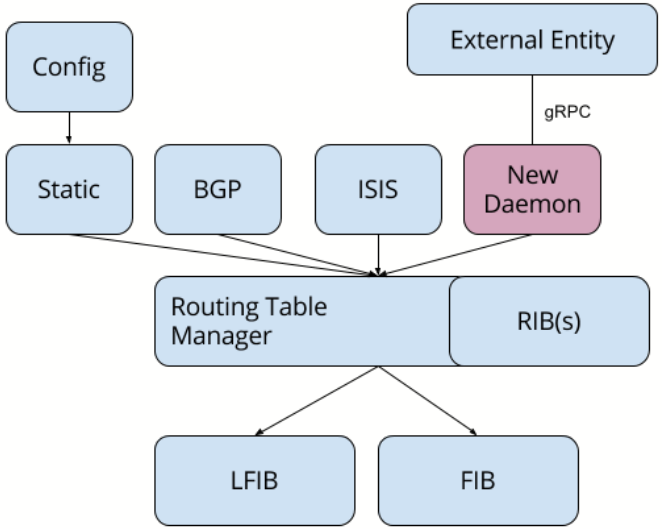
- gRPCを使ったネットワーク管理インターフェース
- テレメトリーと同様な仕組みでサポートする事が可能



gRIBI – gRPC Routing Information Base Interface

https://github.com/openconfig/gribi

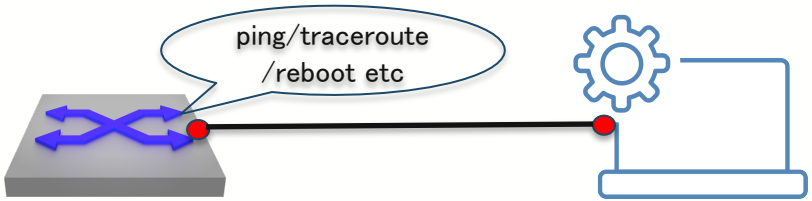
- 現在のネットワークパスをプログラムする手法に主に2つがある
  - OpenConfig/P4 Runtimeの様に直接プログラムする
  - BGP-LUやBGP SR-TEの様にプロトコルの拡張
- gRIBIはgRPCのチャネルを使った新しいアプローチ



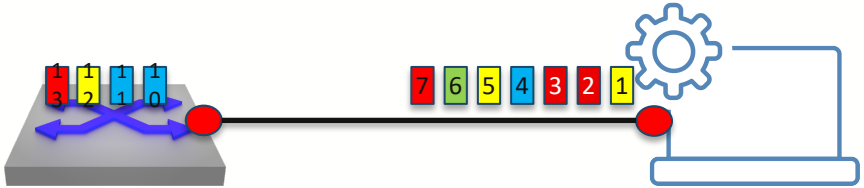
https://github.com/openconfig/gribi/blob/master/doc/img/motivation-fig1.png

gNOI – gRPC Network Operations Interface

- gRPCを使ったマイクロサービスの為のネットワーク運用インターフェース
- BGP, Certificate management, MPLS, interface, layer 2, system (ping, traceroute など)をサポート



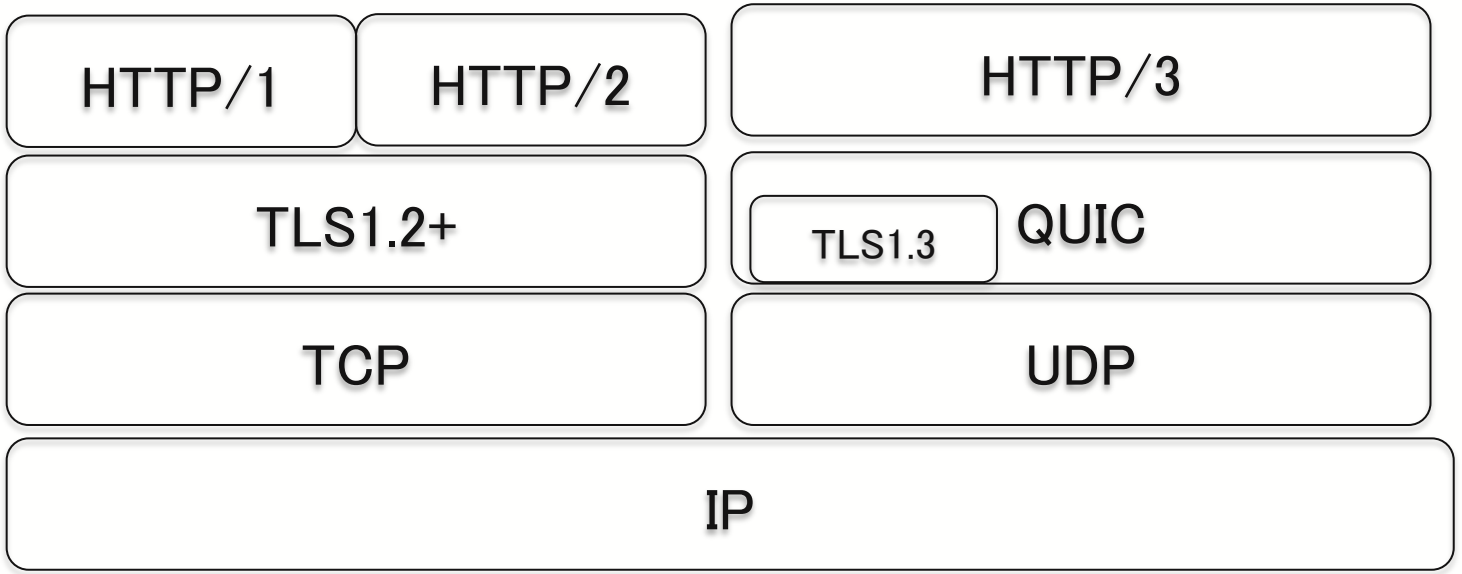
HTTP/2の問題点



- HTTP/2によりhttpのhead on line blockingは無くなったが、TCPである為にパケットが紛失した場合、再送処理がかかり他のストリームにも影響を与えてしまう
- TCP Head On Line Blocking

Hypertext Transfer Protocol Version 3 (HTTP/3)

https://tools.ietf.org/html/draft-ietf-quic-http



- 多重化/ストリーム毎のフローコントロールおよび低遅延のコネクション確立などを行うトランスポートプロトコルとしてQUICを定義
- HTTP/3はQUICトランスポート上でのHTTPの仕組み

QUIC: A UDP-Based Multiplexed and Secure Transport

https://tools.ietf.org/html/draft-ietf-quic-transport

- QUICは下記を実現するための汎用的なトランスポートプロトコル
  - ストリーム多重化
  - ストリーム/コネクションレベルのフローコントロール
  - 低遅延コネクションの確立
  - 接続の移行とNATのリバインディング
  - 認証され暗号化されたヘッダーとペイロード

Multiplexed Application Substrate over QUIC Encryption (masque)

https://datatracker.ietf.org/wg/masque/about/

- 実際のインターネットにはProxyを使用しなければいけない状況がある
- HTTP/3,QUICにおけるProxyのサポートやQUICを使ったVPNのネットワーク
- GoogleではQUICを使ったVPNのネットワーク(QBONE)を既に使用しており、100Gbps+のトラフィックを処理している

まとめ

- TCP443/UDP443を使用したプロトコルやユースケースに関して共有した
- gRPCを使ったネットワーク機器の実装トレンド
- HTTP/2による改善された点
- HTTP/3とQUICで改善される点

まとめ

- テレメトリーデータをコントローラーに送るプロトコル
  - gRPC ：普通に感じました？
- ポート443(HTTP/2またはQUIC)の上で何が行われてるか分からない
  - しょうがなく感じました？