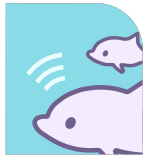
Flow技術まとめ ～基礎から最新動向・応用まで～

2023/11/21 Internet Week

NTT コミュニケーションズ株式会社 イノベーションセンター
三島 航



自己紹介

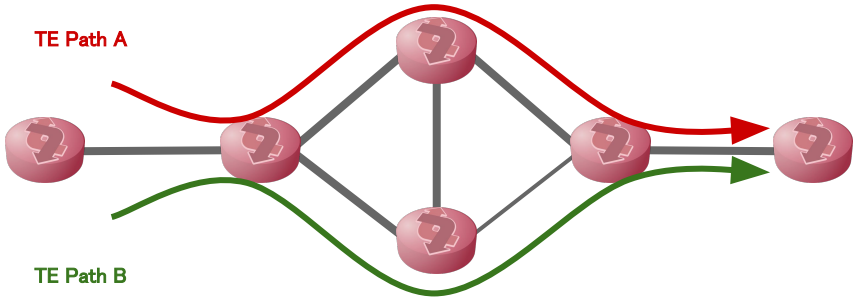
- **三島 航**（watal）
  - NTT コミュニケーションズ イノベーションセンター
  - 業務内容はネットワーク全般の技術検証と Testbed の開発・運用
    - 要素技術: **SR (SRv6/SR-MPLS)** / **xFlow** / **Streaming Telemetry** / MP-BGP / PCEP
  - SR 網の監視のため、IPFIX を選定 & eBPF/XDP ベースの IPFIX exporter を作成して検証中
    - **Fluvia Exporter**: https://github.com/nttcom/fluvia

watal_i27e
https://github.com/watal

背景

- **SRv6/SR-MPLS を用いた応用的な経路制御の技術検証**
  - 遅延が重要となる通信を最適化したい！
    - リアルタイムな遅延メトリックを参照した経路制御や、 SR Policy ごとの遅延の評価を行いたい
    - ライブストリーミングや e-sportsのトラフィックなど、特定用途の通信を制御したい

- **SRv6網においてフローごとの遅延を取得し、経路制御のメトリックにしたい**
  - ある SR Policy の E2E な遅延メトリックを取得するのではなく、hop-by-hop に遅延メトリックを取得したい
    - 例：遅延の原因となるリンクの回避、ルータにおける転送遅延の抑制など



実現に向けて – ネットワーク監視技術の採用

- **SRv6 網において、フロー毎の遅延取得を実現する技術が欲しい！**
  - SRv6 SRH に含まれる Segment List や Segment Left の情報でフローを区別したい
  - outer header の destination や、inner header の 5-tuple なども取得したい

- 上記を満たす技術として **xFlow（IPFIX）を選定**し、技術検証を実現中
  - そもそもフローの選定理由は？

- **以降では、各種 Network Telemetry 技術の利点や自作の IPFIX exporter を紹介！**

監視技術の比較と利点

RFC9232 Network Telemetry Framework

監視技術：Network Telemetry について

- **Network Telemetry**：ネットワークの情報を遠隔で取得する技術の総称
  - Telemetry：tele- + -metry。広義には遠隔測定技術全般を指す言葉
    - 離れた場所からメトリクスを収集する概念
    - F1サーキットのメトリクス収集や、野生動物のタグづけによる監視なども Telemetryと呼ぶ
  - 監視対象や監視手法などの違いから様々な技術が存在
    - xFlow・SNMP・Streaming Telemetry・BMP…
  - 概念を明確に定義し、情報取得のフレームワークを提供するため**RFC 9232** が提案済

RFC 9232 Network Telemetry Framework

- **ネットワーク運用で生じる課題を解決するための Network Telemetry Framework を定義**
  - https://datatracker.ietf.org/doc/rfc9232/
  - Network Telemetry の定義や分類、体系化を行うRFC
    - IPFIX は Network Telemetry に含まれるため、そのメリットを整理
    - 従来の SNMP/CLI/Syslog などが持つ課題が書かれ、あるべき姿を定義

Network Telemetry のユースケース

- **Security**: IDS/IPS。人間の介入を最小化＆通常のトラフィックに影響を与えない保護を目指す
- **Policy and Intent Compliance** Policy や Intent への準拠を監視 & 違反を報告
  - Policy: アクセスの制限・サービスの差別化・特定トラフィックの制御(e.g. SFC)
  - Intent: 運用目標のセット。ネットワークが満たす結果を、手段を指定せずに宣言的に定義するもの
- **SLA Compliance** プロバイダが契約通りのサービスを提供しているか、ユーザやオペレータが確認・評価する
- **Root Cause Analysis** ネットワーク障害の原因を迅速に識別
  - 機械学習などの技術でも分析可能。ネットワークからの入力はactive/passive の双方が選択可能
- **Network Optimization** 短期・長期でのネットワーク最適化を行うためのロードバランシング・TE・計画など
  - 短期：輻輳の回避には、マイクロバーストを検出するためのリアルタイムかつ細かい粒度での取得が必要
  - 長期：ネットワーク容量やトポロジの計画には、長期にわたるNetwork Telemetry データの分析が必要
- **Event Tracking and Prediction** 将来の障害を避けるため、ネットワーク状態の把握と追跡を可能にする
  - 取得対象：トラフィックの経路やパフォーマンス、パケットのドロップなどのイベント

Network Telemetry について

- **一般に下記の特徴を持つ（全てを満たすとは限らない）**
  - **Push and Streaming**: ポーリングではなくデバイスのデータソースから push されたストリーミングデータ（e.g. Pub/Sub）
  - **Volume and Velocity**: 人間ではなく機械による自動化された処理を前提とした、大規模かつリアルタイムなデータ
  - **Normalization and Unification**: 自動化への親和性の高い、異なるデバイス間で正規化されたデータやプロトコル
  - **Model-Based**: モデル化された Telemetry data。アプリケーションがモデルからデータを構成可能にする
  - **Data Fusion**: 複数のデータソースで共通の name/ID を持ち、cross-domain/device/layer に相関を取得可能にする
  - **Dynamic and Interactive**: closed-loop 等による自動化のため、コントローラからの動的かつ対話的なクエリに対応する

- **理想的には下記の特徴を持って良い**
  - **In-Network Customization**: アプリケーションのニーズを満たすためのネットワーク側でのデータの加工
  - **In-Network Data Aggregation and Correlation**: ネットワーク内でのデータの集約（間隔の設定や一部の廃棄など）
  - **In-Network Processing**: データ処理をネットワークにオフロード
  - **Direct Data Plane Export**: リアルタイム処理を目的とした、データ消費者への直接エクスポート（e.g. IOAM DEX）
  - **In-Band Data Collection**: 転送パス全体のフローの直接収集（Passive/Active/Hybrid なデータ収集）

Telemetry Protocol の分類軸: Plane 毎の整理

- **Management Plane Telemetry**
  - ネットワーク管理システムから情報を取得するTelemetry
  - パフォーマンス・ネットワークロギング・警告・統計や状態などの情報を提供を目指す
  - gRPC, NETCONF, RESTCONF, SNMP, Syslog
- **Control Plane Telemetry**
  - L1-L7 の C-Plane Protocol 自体の監視を行うTelemetry
  - ネットワークの最適化や問題の検知、予測を行う。
    - ただし、E2E の KPI を特定レイヤの KPI と結びつけ、具体的な事象の発生理由をあるプロトコルやデバイスに特定するのは困難
  - **xFlow**, BMP, traffic mirroring, gRPC, NETCONF, RESTCONF
    - BMP のように、IGP のモニタリングをするプロトコルは存在していない
- **Data Plane Telemetry**
  - データプレーンでの転送に関する情報の監視を行うTelemetry
  - **xFlow**, traffic mirroring, gRPC, NETCONF

参考：Network Telemetry についての考慮点

- **RFC 9232 では下記の考慮点が書かれている。これらは D-Plane では特に厳しくなる。**
  - D-Plane のメインであるパケットの処理と転送に影響を与えないように努める
  - 配信する Telemetry データによって帯域を使い果たしてはならない
  - 最小遅延でのタイムリーなデータの提供
    - 処理・転送・ディスク書き込み・分析等にかかる時間を最小に抑える
  - アプリケーションが処理しやすい構造化とラベル付け。またプログラマビリティ
  - 非対応機器への機能追加（拡張性）

Technique Taxonomy (1/2)

- **Active, Passive, and Hybrid**（c.f. RFC 7799）
  - Active: OWAMP/TWAMP/STAMP, ping など。カバレッジの高い計測が可能だが、帯域を消費する
  - Passive: xFlow, tcpdump, traffic mirroring など。フローの品質を直接測れるが、流れている区間しか計測できない
  - Hybrid: IOAM, AM (Alternate-Marking) など。柔軟な計測が可能だが、実装が複雑

- **In-Band and Out-of-Band**
  - In-Band: collector への export 前に、ユーザパケットにTelemetry データを付与するアプローチ
    - 一般に言われる In-band Telemetry はこの整理とは異なる。IETF 的には On-path Telemetry と呼ばれる
  - Out-of-Band: ユーザパケットを変更せず、collector に Telemetry パケットを直接 export するアプローチ

Technique Taxonomy (1/2)

- **End-to-End and In-Network**
  - End-to-End: ネットワークの endpoint から endpoint への計測を行うアプローチ（ping による計測のイメージ）
  - In-Network: ネットワーク上のあるポイントで計測する手法（Flow はこれ）

- **Data Subject**
  - flow based: IOAM など、あるフロー全体を包括するデータ
  - path based: traceroute のように、ある経路単位でのデータ
  - node based: IPFIX のように、あるポイントで取得されるデータ

（参考）SRv6 における遅延計測＆広告手法の比較

- **OWAMP/TWAMP/STAMP + IGP**
  - Active Measurement
  - flow 毎の遅延計測性能がない
- **IPFIX On-Path Telemetry**
  - Passive Measurement & In-Band/Out-of-Band Hybrid
  - On-Path Telemetry Postcard 型アプローチ & IOAM 対応プロトコル限定（IPv6/SR-MPLS/GRE/Geneve/NSH等）
- **SRv6 Path Tracing**
  - Passive Measurement & In-Band
  - On-Path Telemetry Passport 型アプローチ & SRv6 限定

→ **Postcard 型アプローチのスケーラビリティと IOAM の対応プロトコルの豊富さを考慮し、IPFIX On-Path Telemetry を選択！**

Evolution of Network Telemetry Applications

- 自動操作に向けた進化の段階
  - **Stage 0 - Static Telemetry**: データソースとタイプが設計時に決定され、オペレータは限定的な情報を元にNWを操作
  - **Stage 1 - Dynamic Telemetry**: データソースやタイプをNW操作を中断することなく動的に構成可能
  - **Stage 2 - Interactive Telemetry**: リアルタイムなフィードバックをもとに、柔軟なネットワーク操作が可能
  - **Stage 3 - Closed-Loop Telemetry**: 人間のオペレータがNW操作に干渉しない。Telemetry データが自動的に分析され、ネットワークが操作・更新される

- 現在の情報取得技術は Stage 0 and 1
- 個々のアプリケーションでは Stage 2 or 3 の準備ができているものもある
  - Network Telemetry Framework で定義されている技術は、このためのfirst step になる

ここまでのまとめ：監視技術の比較と利点について

- **監視技術の整理のため、Network Telemetry Framework について紹介！**
  - Security・ネットワーク最適化・TE・トレースなど様々なユースケースに対応
  - 満たすべき特徴：IPFIX は基本的な特徴を満たしている
- **Network Telemetry の定義と分類・体系化**
  - Plane 毎の分類：Management Plane・Control Plane・Data Plane
  - その他の分類：監視の手法・On-Path Telemetry の分類・データの取得区間など
- **Network Telemetry の目指すべき世界**
  - Closed-loop による自動化された運用に向け、Step 0 から Step 3までの段階が定義

→ 次ページからは、xFlow 技術の1つである IPFIX について紹介！

IP Flow Information eXport（IPFIX）

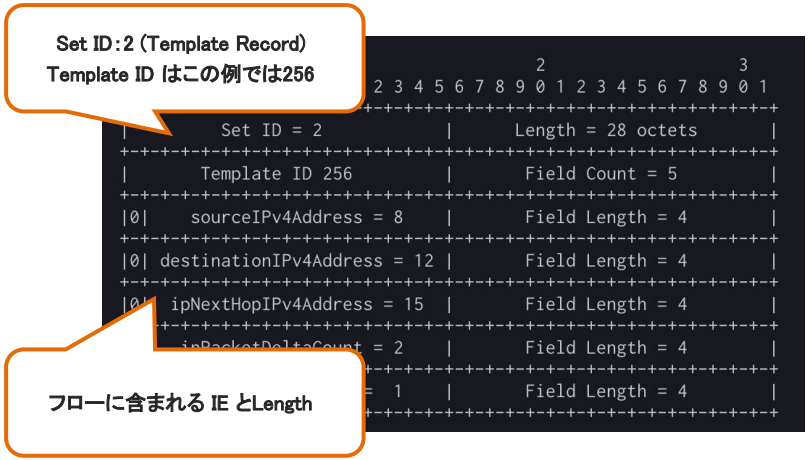
IP Flow Information eXport（IPFIX）

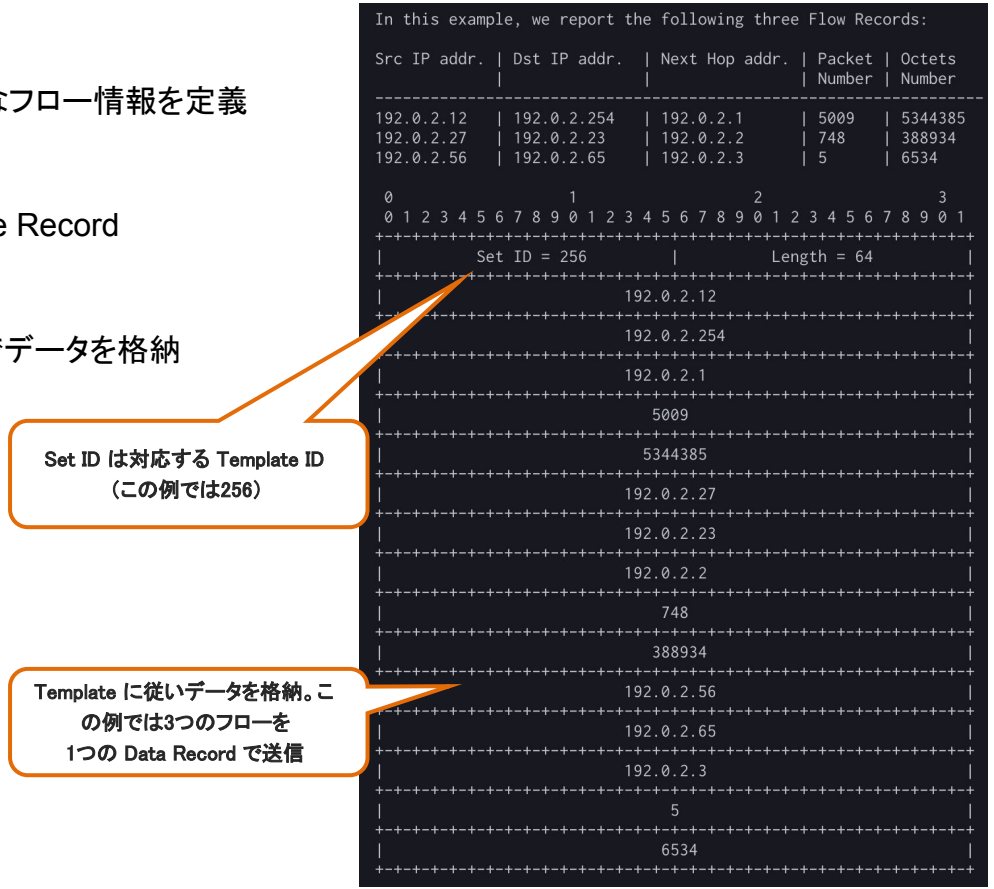
- **ネットワーク上の IP フローをエクスポートするプロトコル**
  - トラフィックの監視・分析のために利用
  - 情報送信側を IPFIX exporter、受信側を IPFIX collector と呼称

- フローデータは **Record** と呼ばれる単位で構造化
  - フローの各要素は、**Information Element（IE）**と呼ばれる単位で扱われる
    - IE一覧：https://www.iana.org/assignments/ipfix/ipfix.xhtml
  - Record には **Template Record・Options Template Record・Data Record** の3種が存在
    - Template は定期的、あるいは Data と共にエクスポートされる
  - Transport は UDP・TCP・SCTP が選択可能（ポート番号:4739）

IPFIX – Template Record と Data Record

- **Template Record**
  - フローのフォーマットを定義するための Record。基本的なフロー情報を定義
- **Options Template Record**
  - そのフローに付随する追加の情報を扱うための Template Record
- **Data Record**
  - 実際のフロー情報。Template で定義されたフォーマットでデータを格納
  - 異なるフローを1つの Data Record に格納可能





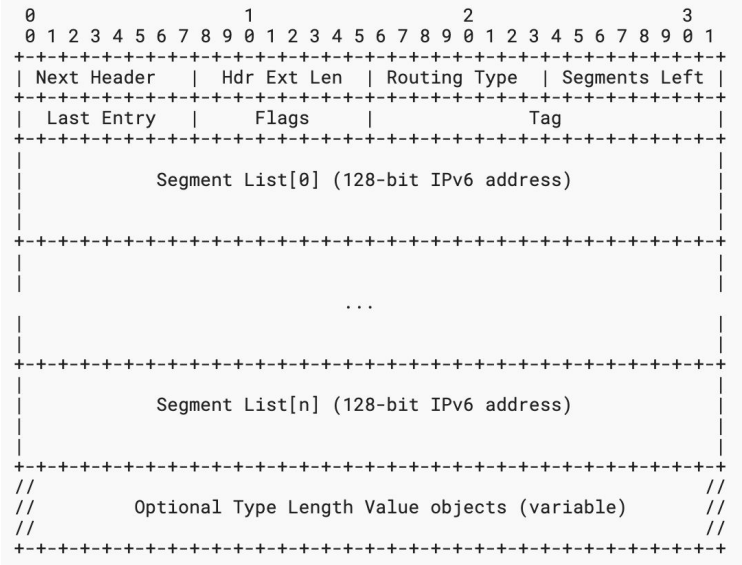
IPFIX を用いた SRv6 SRH と On-path Delay の取得

- **下記2種の RFC/I-D で提案済**
  - SRv6 SRH の取得：Export of Segment Routing over IPv6 Information in IPFIX
    - https://datatracker.ietf.org/doc/rfc9487/
  - On-path Delay の取得：Export of On-Path Delay in IPFIX
    - https://datatracker.ietf.org/doc/draft-ietf-opsawg-ipfix-on-path-telemetry/

Export of Segment Routing over IPv6 Information in IPFIX

- **SRv6 のフローを扱うため、下記11種の IE を定義**
  - **srhFlagsIPv6** : SRH の Flags フィールド
  - **srhTagIPv6** : SRH の Tag フィールド
  - **srhSegmentIPv6** : SID（ipv6address）
  - **srhActiveSegmentIPv6** : Active Segment
  - **srhSegmentIPv6BasicList** : Segment List（basicList）
  - **srhSegmentIPv6ListSection** : Segment List（octetArray）
  - **srhSegmentsIPv6Left** : SRH の Segment Left
  - **srhIPv6Section** : SRH 全体（octetArray）
  - **srhIPv6ActiveSegmentType** : Active Segment の学習方式
  - **srhSegmentIPv6LocatorLength** : Locator 長
  - **srhSegmentIPv6EndpointBehavior** : SID が表す behavior の番号

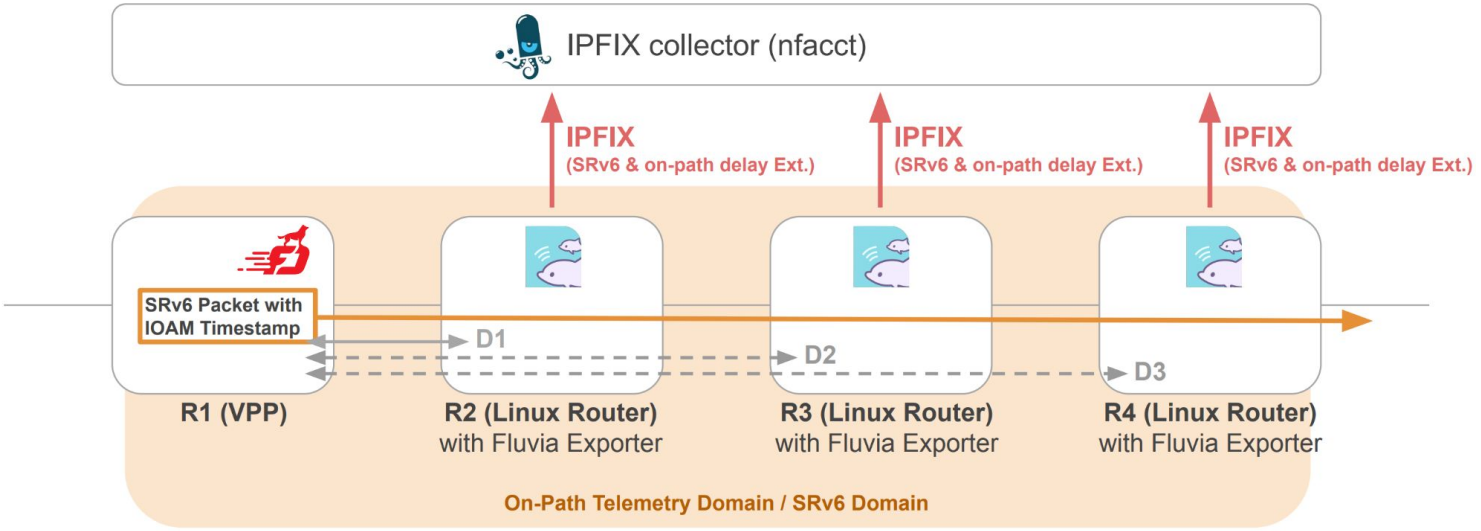
- **参考：IPFIX のデータ型について**
  - basicList：同じデータ型の IE を複数まとめるためのデータ型。
    データは構造化されている
  - octetArray：バイト列を格納するためのデータ型。
    データはバイト列がそのまま格納される



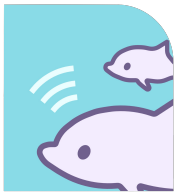
参考：SRH format（RFC 8754）

Export of On-Path Delay in IPFIX

- **IPFIX で遅延(On-Path delay)を取得するための I-D**
  - In-situ OAM(IOAM)で取得した遅延情報と、機器のtimestamp の差から、送出からの累積遅延を計測
  - その他の IPFIX の情報を元に、一連のフローを特定、それぞれの遅延の差から、区間毎の遅延を計算
    - SRv6 の場合は、先述の RFC 9487 の IE と組み合わせ、SRv6 Policy ごとの遅延を取得することも可能



自作 IPFIX exporter: Fluvia Exporter



なぜ自作の IPFIX exporter が必要？

- **新標準やアイデアへの迅速な対応**
  - NTT Com の R&D 部門として、研究開発を進めるためのツールとして利用
  - 開発に着手した当初はどちらの技術もI-D であるため、商用ルータでの既存実装が存在しない→ 自作しよう
    - https://datatracker.ietf.org/doc/draft-ietf-opsawg-ipfix-srv6-srh/06/
    - https://datatracker.ietf.org/doc/draft-ietf-opsawg-ipfix-on-path-telemetry/00/

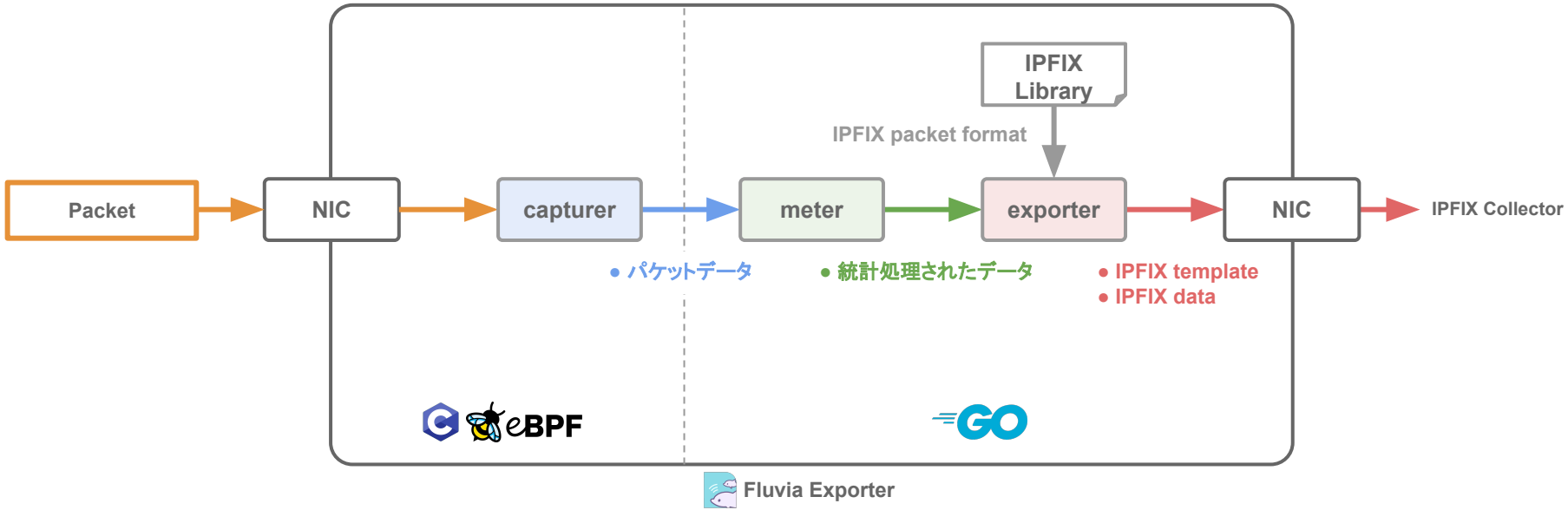
- **PoC を通じて価値を確かめ、商用機器にも実装されることを期待**
  - 自作 IPFIX exporter 自体は社内網での永続的な利用や商用網での利用は想定せず、あくまでPoC として利用
    - 性能よりも実装・拡張のしやすさを重視
  - OSS として公開することにより、コミュニティ手動で開発されることを期待
    - 必要最低限の IE のみを実装

Fluvia Exporter

- Linux など eBPF/XDP 環境で動作する IPFIX exporter + Go の IPFIX Library
  - OSS 公開済み（MIT）
    - https://github.com/nttcom/fluvia
- 下記2種の RFC/I-D に対応。SRv6 と on-path delay の取得が可能
  - Export of Segment Routing over IPv6 IPFIX (https://datatracker.ietf.org/doc/rfc9487/)
  - Export of On-Path Delay in IPFIX (https://datatracker.ietf.org/doc/draft-ietf-opsawg-ipfix-on-path-telemetry/)

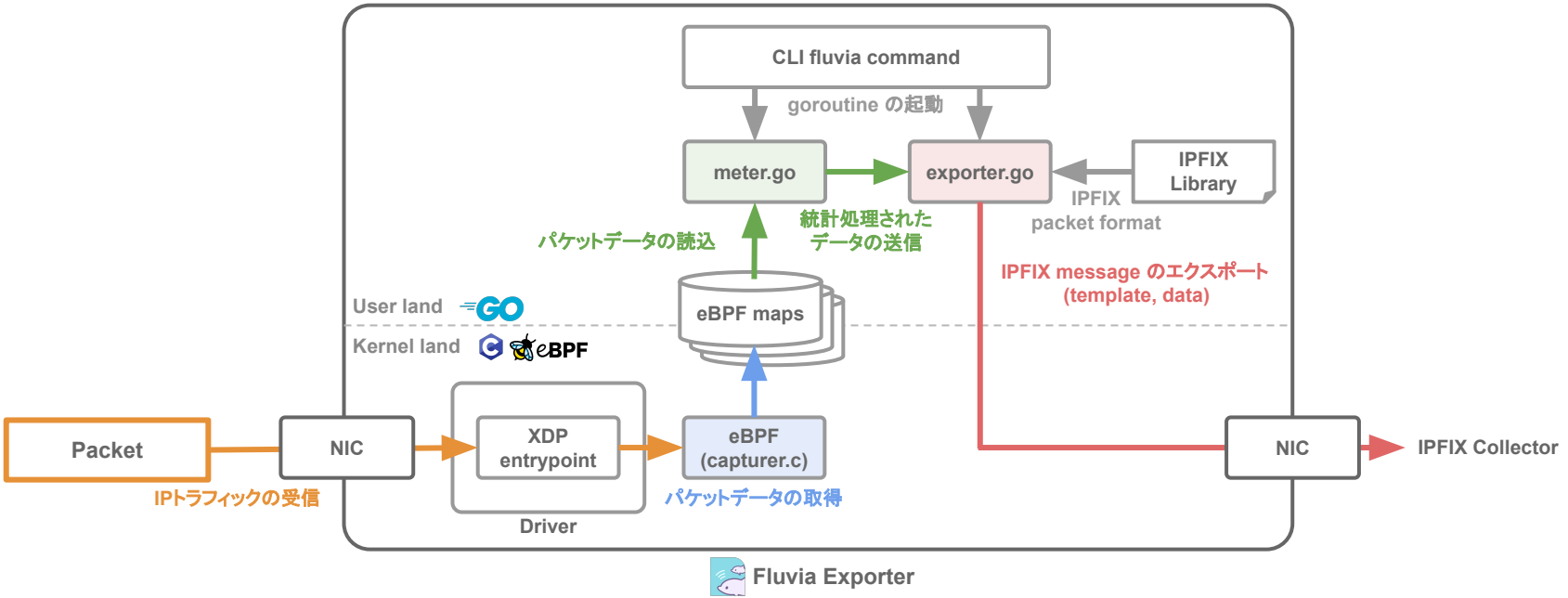
アーキテクチャ（概要）

- capturer・meter・exporter の3つのコンポーネントによるシンプルなデザイン
  - 取得したパケットを統計処理し、IPFIX 形式で export するパイプライン



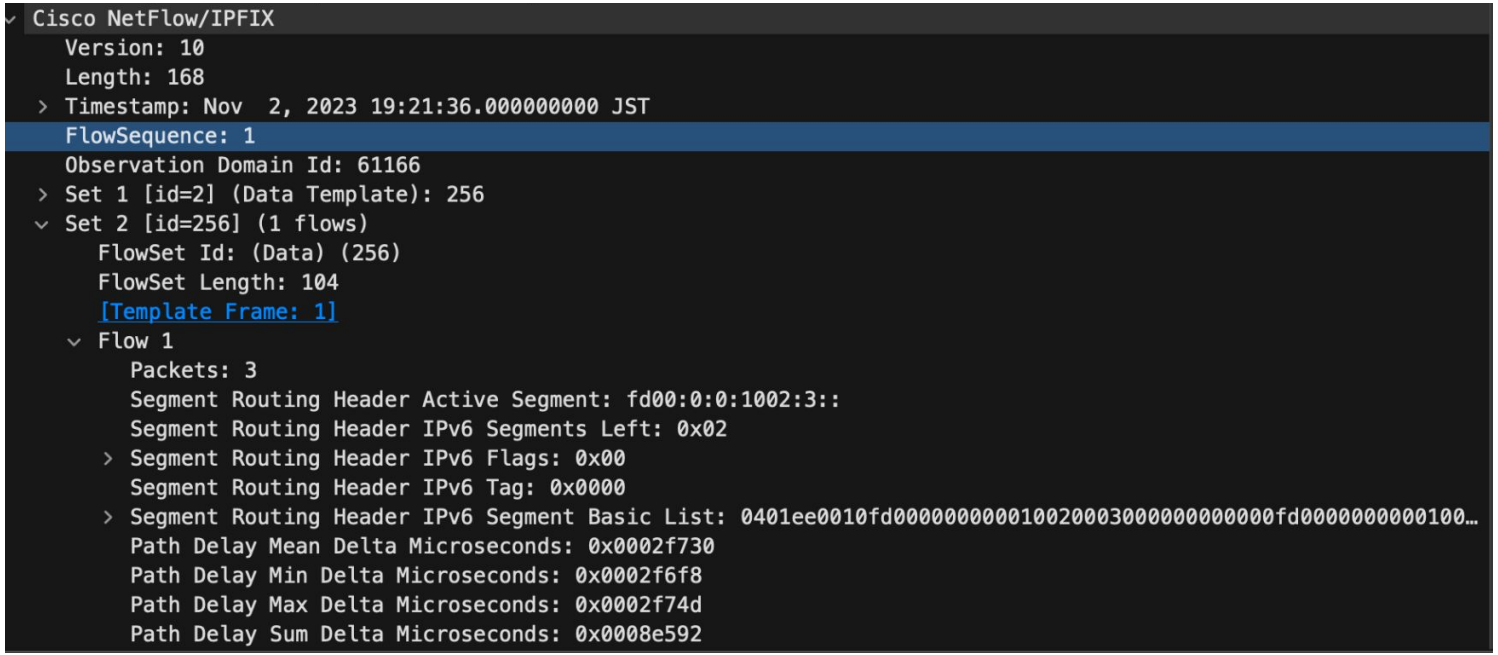
アーキテクチャ（詳細）

- **eBPF/XDP を利用したパケット取得と、Goによる統計処理＆パケット送出**
  - capturer となる eBPF プログラムは C で実装。取得したパケットをeBPF map に書き込み
  - meter, exporter を含む User land 部は Go で実装。goroutine によるマルチスレッド化とchannel でのデータ共有



参考：WireShark Dissector

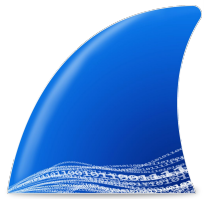
- 新技術の検証やトラブルシューティングの際には、パケットを確認することが効果的
  - Dissector と呼ばれるパケットの構造の定義を記載し、GUI から確認可能に
    - https://gitlab.com/wireshark/wireshark/-/merge_requests/11138/diffs



参考：WireShark Dissector の書き方

- Gitlab にある公式リポジトリを fork し、epan/dissectors/ 内に追加実装
  - 参考：https://gitlab.com/udon-yuya/wireshark/-/compare/master...feature%2Fipfix-on-path-delay-dissector?from_project_id=7898047

- ソースコードからビルドし利用
  - $ cd wireshark
  - $ mkdir build && cd build
  - $ cmake -DQt5_DIR=$(brew --prefix qt5)/lib/cmake/Qt5 ..
  - $ make
  - $ mv run/Wireshark.app /Applications/



開発について

- IETF 116/117/118 の Hackathon にて開発
  - 最終発表資料:https://github.com/IETF-Hackathon/ietf118-project-presentations/blob/main/ietf118-hackathon-on-path-telemetry.pdf
  - (参考)IETF Hackathon Wiki:https://wiki.ietf.org/en/meeting/118/hackathon
  - I-D の著者と直接連絡し、仮決めの番号などを統一し相互接続検証を実施
    - 実装対象の I-D は IANA 払出しが未完了であったため、自分たちで値を決める必要あり

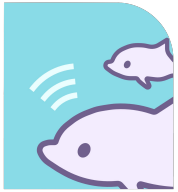
- **Fluvia Exporter:**
  - **COMPLETED: Decode the IOAM timestamp with eBPF/XDP**
    - RFC 9197
  - **COMPLETED: Export the on-path delay & SRv6 SRH information with IPFIX**
    - draft-ietf-opsawg-ipfix-on-path-telemetry-04
    - Interoperability test was succeeded with **nfacct and following Wireshark dissector**
- **Wireshark:**
  - **COMPLETED: IPFIX IEs for on-path telemetry**
    - draft-ietf-opsawg-ipfix-on-path-telemetry-04
  - **TODO: IOAM timestamp format**
    - RFC 9197, draft-ahuang-ippm-dex-timestamp-ext

自作 IPFIX Exporter のまとめ

- **迅速なアイデアの実現を目指し、Fluvia Exporter を作成**
  - 性能よりも実装・拡張のしやすさを重視

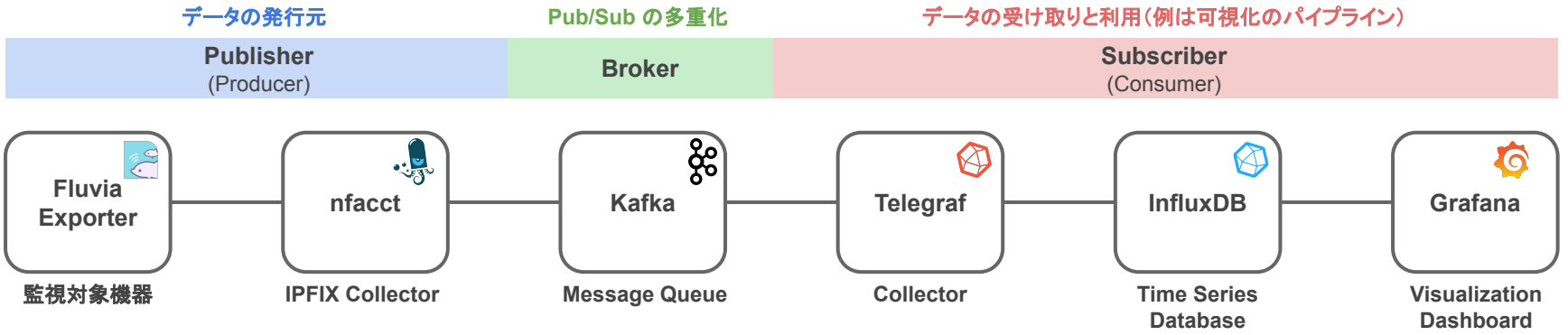
- **今回の目的である SRv6 網のデータ収集・遅延メトリックの活用のため、2つの I-D に対応**
  - Export of Segment Routing over IPv6 IPFIX
  - Export of On-Path Delay in IPFIX
  - IETF Hackathon で、I-D の著者と連携をとりつつ実装

- **NTT Com 公式リポジトリより OSS 公開中**
  - https://github.com/nttcom/fluvia



参考：データ処理のパイプライン

- 監視対象機器から Export したデータは、目的毎に適切にパイプライン処理を行う必要がある
- 下記は Pub/Sub アーキテクチャでデータの可視化を行う例
  - export された IPFIX の受け取り：nfacct
  - Brokerによるスケーラビリティの向上：Kafka
  - 時系列 DB とダッシュボードによる可視化：TIG stack（Telegraf/InfluxDB/Grafana）



今後の課題

- **遅延メトリックを用いた経路計算の実現**
  - TED 内にインターフェースが持つ情報として埋め込み（GoBGP の API を利用した実現を検討）
  - 経路計算までの処理時間の計測

- **パケットの処理性能の調査**
  - Fluvia Exporter は PoC であり実利用するものではないが、性能を計測しておきたい
    - 今回の実装では eBPF の処理後に Linux Kernel の network stack にパケットを渡しているため、そこで律速する

- **サンプリング間隔の調査**
  - 一般に、取得間隔は機器の性能に依存
  - Fluvia Exporter では、eBPF 自体の処理性能や eBPF map への書き込みが重要となると予想

まとめ

まとめ

- **各種 Network Telemetry 技術の利点や立ち位置を整理**
  - Network Telemetry のユースケースと満たすべき要件
  - Plane 毎の整理や技術の体系化、自動化に向けた進め方の整理
- **NTT Com の活動を通じ、IPFIX の利用例を紹介**
  - SRv6 網で達成したいユースケース
  - 自作 IPFIX Exporter である Fluvia exporter について
  - データ活用のためのパイプラインの紹介
- **今後はデータ処理部分に取り組み、SRv6網の機能向上を目指す**
  - TE 機能との連携によるフローのQoS 向上
  - Network Telemetry Step 3 の実現