Registerを使ったトラフィック解析の例：NetFlow

APRESIA Systems株式会社　桑田　斉

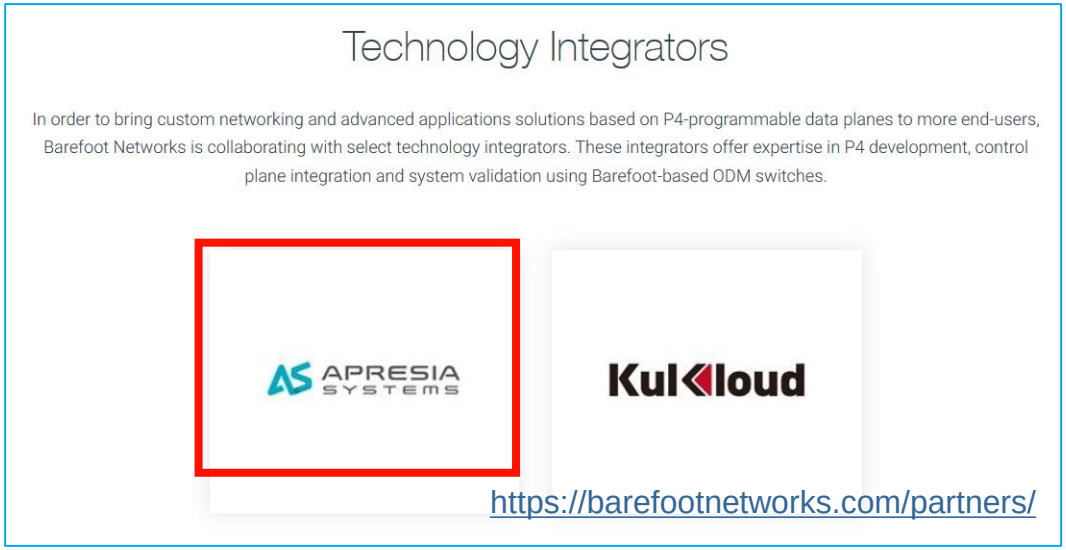
hitoshi.kuwata.gt@apresiasystems.co.jp

2020/10/22

APRESIAのP4の取り組み

- Edgecore社のPremium Distributorとしてホワイトボックススイッチを販売
- P4プログラマブルスイッチであるWedge100BF32(Barefoot Tofinoチップ搭載)も販売
  - Barefoot NetworksからTechnology Integratorsとして認定

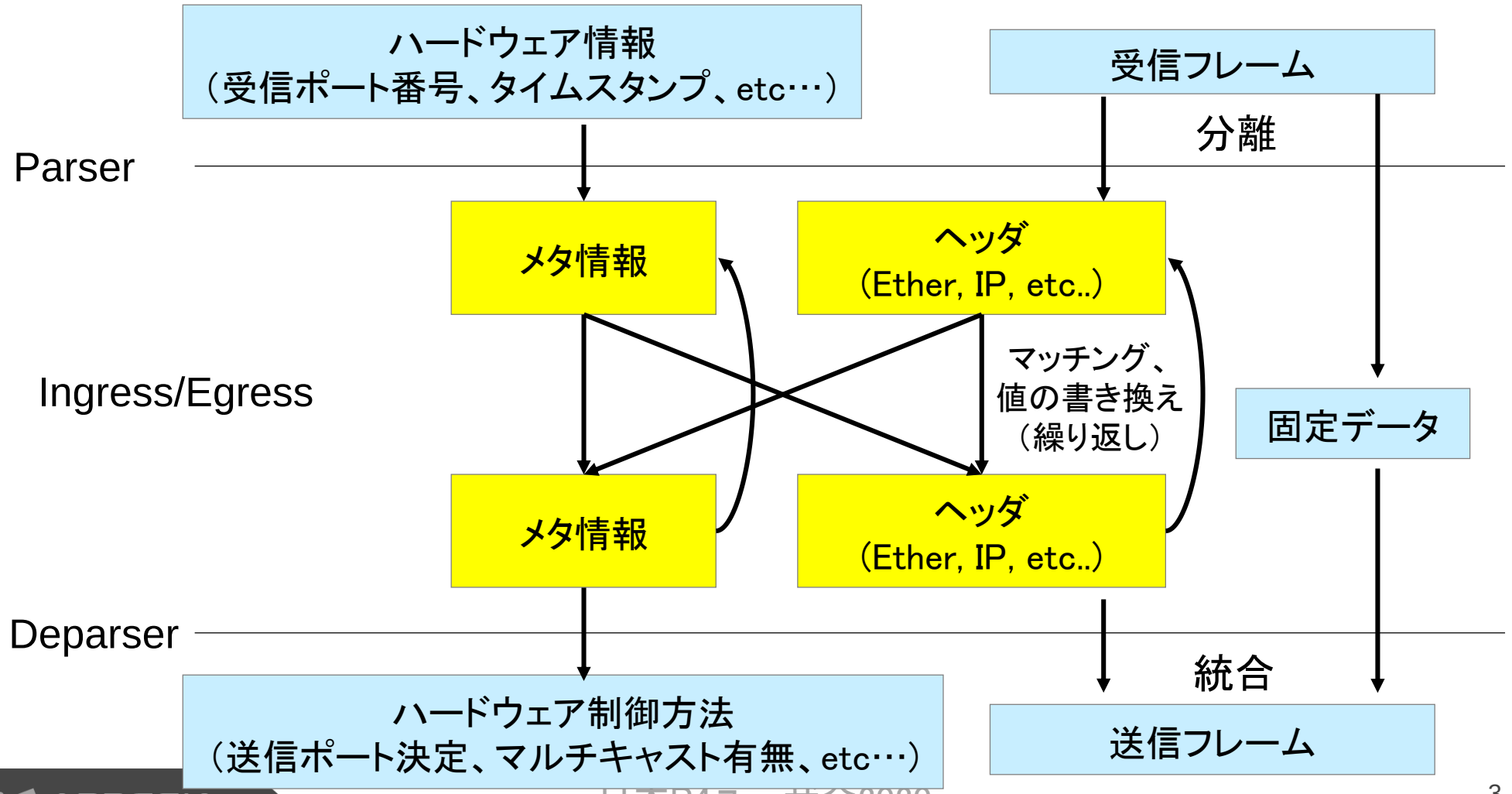
Wedge100BF32
Barefoot Tofinoチップ搭載
100G x 32ポート



- 最近のP4の活動 （JANOG46）
  - 次世代SDNスイッチStratumとONOSコントローラーを使ったパケット制御
    - https://www.janog.gr.jp/meeting/janog46/sdn/
  - SONiC + P4によるマルチテナントSRv6サービスチェイニングの実現
    - https://www.janog.gr.jp/meeting/janog46/sonic/

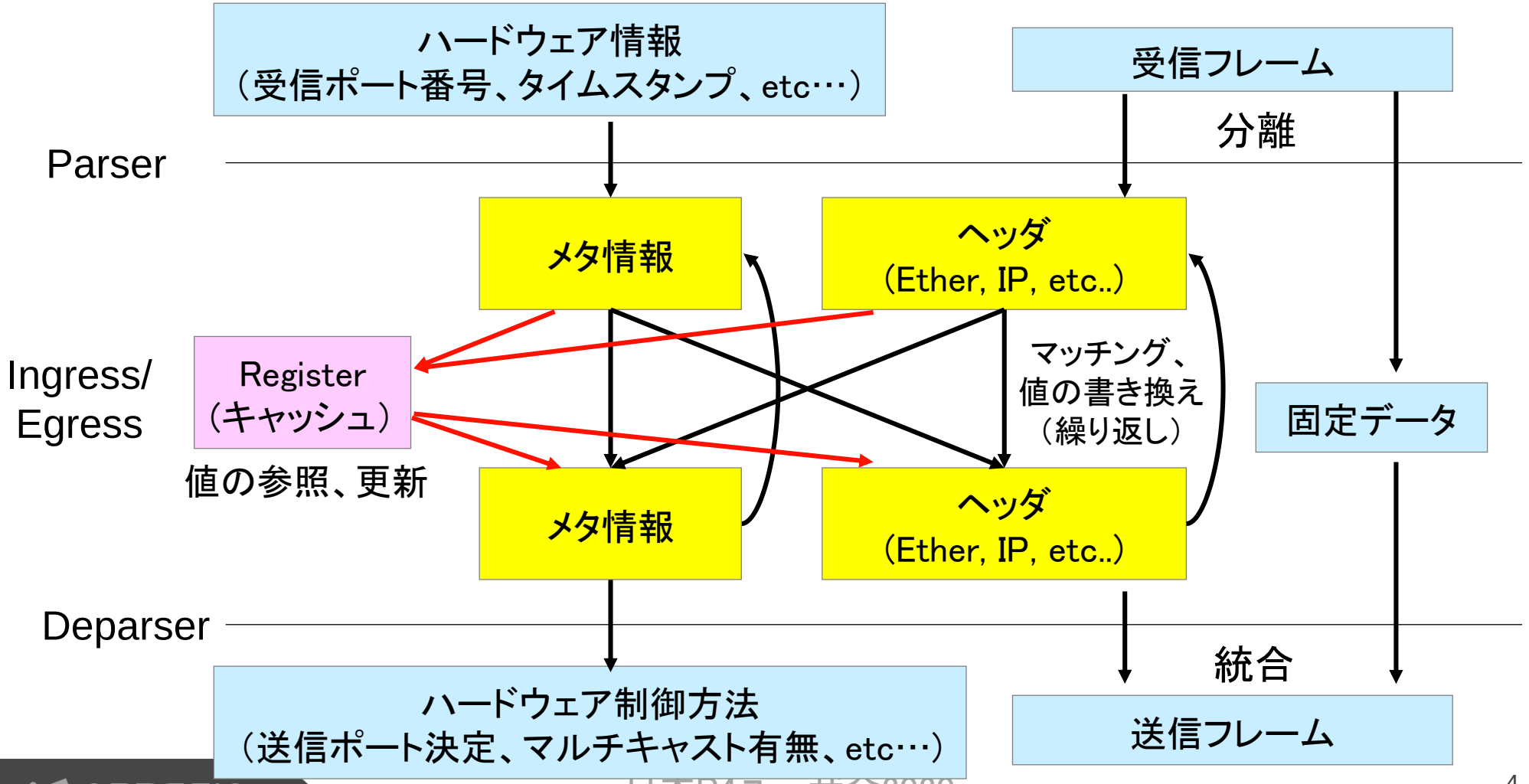
おさらい P4_16のプログラミングのイメージ

- Parseで抽出したヘッダとハードウェアが付与するメタ情報を組み合わせてプログラミング
- 更新されたヘッダ情報にて送信フレームを再生成



P4におけるRegisterプログラミング

- メタ情報やヘッダ情報から計算した値をRegisterに記録
- Registerに保持した情報を使ってメタ情報、ヘッダ情報を更新



Registerの活用アイディア

【Registerの特徴】
過去に受信したフレームの情報を保持・参照することが可能

【活用方法①】
過去に受信したフレームをもとにフレーム書き換え、制御方法を変更

- シーケンス番号の抜け・重複チェック
- ARP/ICMP要求をハードウェアでキャッシュした結果から応答

【活用方法②】
過去に受信したフレームの情報をキャッシュに保存して、トラフィック分析に使用

- 要求・応答時間の差分（レスポンスタイム）
- フロー情報の保存（NetFlow）

今回はこちらを試した結果を共有します

P4 Registerキャッシュ → NetFlowへ変換

APRESIA

弊社事務所

トラフィック

Mirroring

外部NW

Wedge100BF32

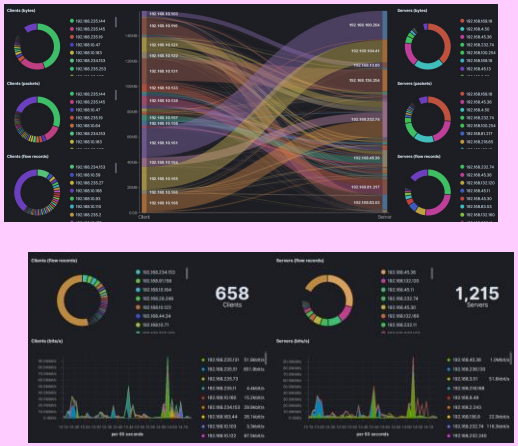
Etherヘッダ | IP ヘッダ | Dst IPアドレス | Src IPアドレス | IP Protocol | TOS | TCP/UDPヘッダ | Dst Port | Src Port

データプレーン

Dst IP アドレス	Src IP アドレス	IP Protocol	TOS	Dst Port	Src Port	パケットカウンタ	バイトカウンタ
X.X.X.X	X.X.X.X	X	X	X	X	xxxx	xxxx
Y.Y.Y.Y	Y.Y.Y.Y	Y	Y	Y	Y	yyyy	yyyy
Z.Z.Z.Z	Z.Z.Z.Z	Z	Z	Z	Z	zzzz	zzzz

Register

【可視化】 ElastiFlow



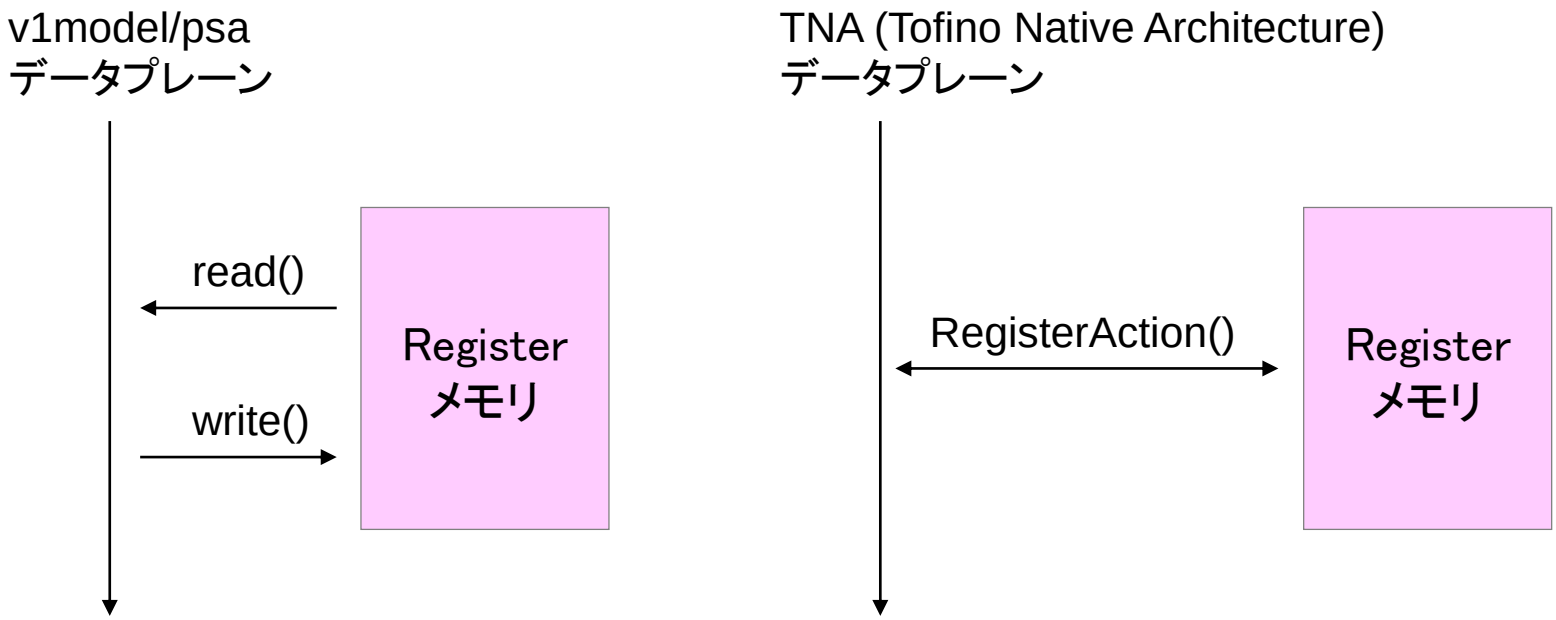
NetFlow v5

【コントロールプレーン処理】
Register読み出し
NetFlow v5へ変換

gRPC

Barefoot TNAモデルのRegisterAction()関数

- v1model/psaモデルのRegisterはread()/write()関数が用意されている
  - 例えばカウンタをインクリメントする場合、read()した値に加算した値をwrite()する必要があり、Registerへのアクセスが二回発生
- TNA (Tofino Native Architecture)にはRegisterの値の読み書きを同時に実行可能なRegisterAction()関数が用意されている
  - Registerメモリへのアクセスを一回にまとめることは、ハードウェアのデータプレーンの実現に重要



Barefoot TofinoのRegisterActionのサンプルP4コード

■Registerのエントリ（index番号で指定）が使用されているかのフラグの読み出しとセットの例

```
Register<bit<8>, bit<FLOW_REG_SIZE_BIT>>(FLOW_REG_SIZE) flow_entry_status;

RegisterAction<bit<8>, bit<FLOW_REG_SIZE_BIT>, bit<8>>(flow_entry_status) flow_entry_update = {
    void apply(inout bit<8> flow_status, out bit<8> result) {
        result = flow_status;
        flow_status = FLOW_REG_STATUS_BIT_USED | FLOW_REG_STATUS_BIT_HIT;
    }
};
```

RegisterActionの出力であるresultにRegisterの値(flow_status、初期値0)を保存してから、flow_statusにフラグをセット

■Bytesカウンタ用のレジスタの例

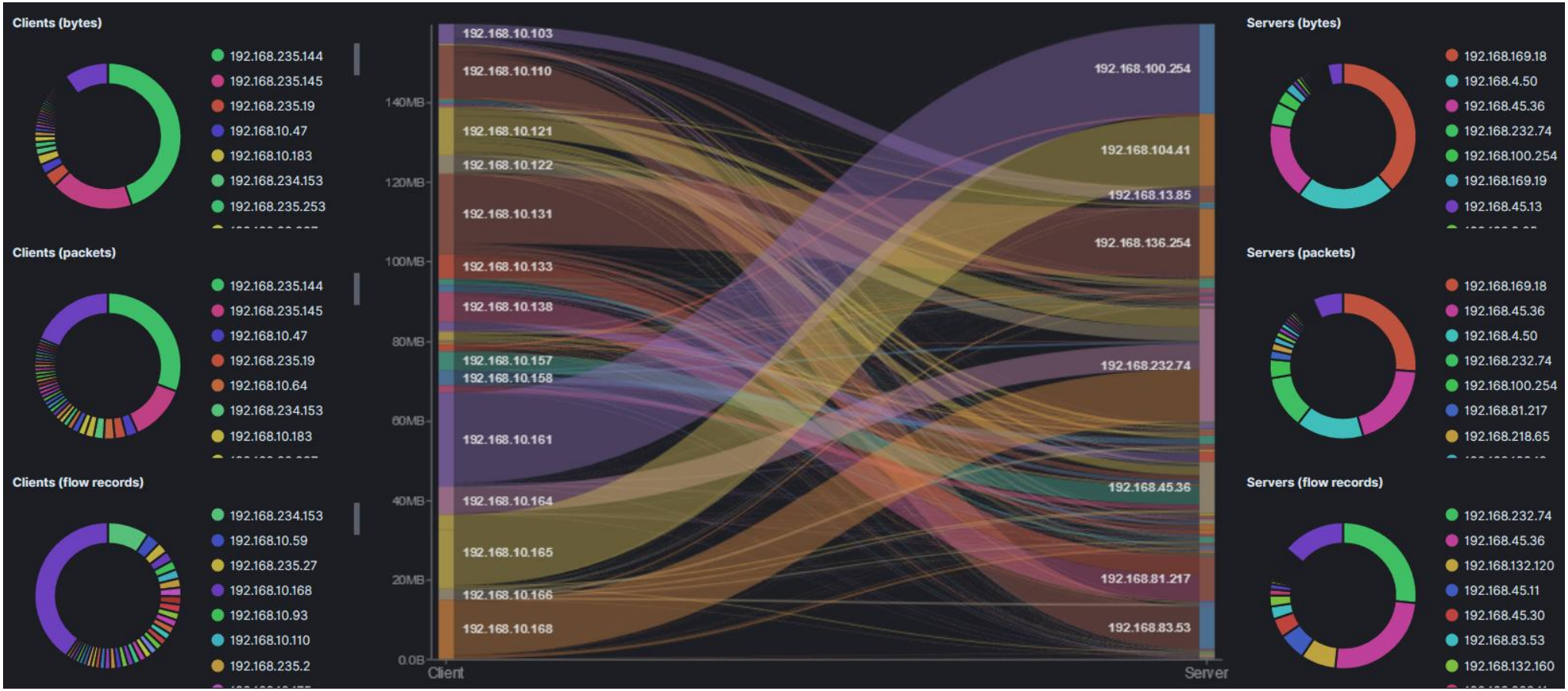
```
Register<bit<32>, bit<FLOW_REG_SIZE_BIT>>(FLOW_REG_SIZE) flow_v4_reg_byte_count;

RegisterAction<bit<32>, bit<FLOW_REG_SIZE_BIT>, void>(flow_v4_reg_byte_count) flow_byte_count_init  = {
    void apply(inout bit<32> count) {
        count = (bit<32>)eg_md.pkt_len;
    }
};

RegisterAction<bit<32>, bit<FLOW_REG_SIZE_BIT>, void>(flow_v4_reg_byte_count) flow_byte_count_update  = {
    void apply(inout bit<32> count) {
        count = count + (bit<32>)eg_md.pkt_len;
    }
};
```

Registerの値(count)にフレーム長を加算した値でRegister(count)を書き換え

可視化の結果例（フロー図）



※IPアドレスの上位16bitは’192.168’に上書きしている

まとめ

- 実現できたこと
  - P4のRegisterにフローデータをキャッシュし、キャッシュしたデータをNetFlowに変換して可視化
- 気づき
  - Register（キャッシュ）を使用することで、ハードウェアでのARP応答など、ネットワークのパフォーマンスを改善するような制御も実現できる可能性あり
  - P4によって、Registerにネットワークの特徴量を保存する処理をプログラミングすることで、新しいトラフィック分析が実現できる可能性あり
  - 使用可能なRegisterのメモリ量はハードウェアに依存
    - 将来的に、Wedge100BF32の後継機種やFPGA NICなど、大容量のメモリをRegisterとして使用できるようになることに期待

Thank You!

是非こちらの情報もご参照ください！

https://www.apresia.jp/

http://www.apresiatac.jp/blog/