RIPE76 - Rebuilding a network data pipeline

Louis Poinsignon

Who am I

Louis Poinsignon

Network Engineer @ Cloudflare.

Building tools for data analysis and traffic engineering.

What is Cloudflare?

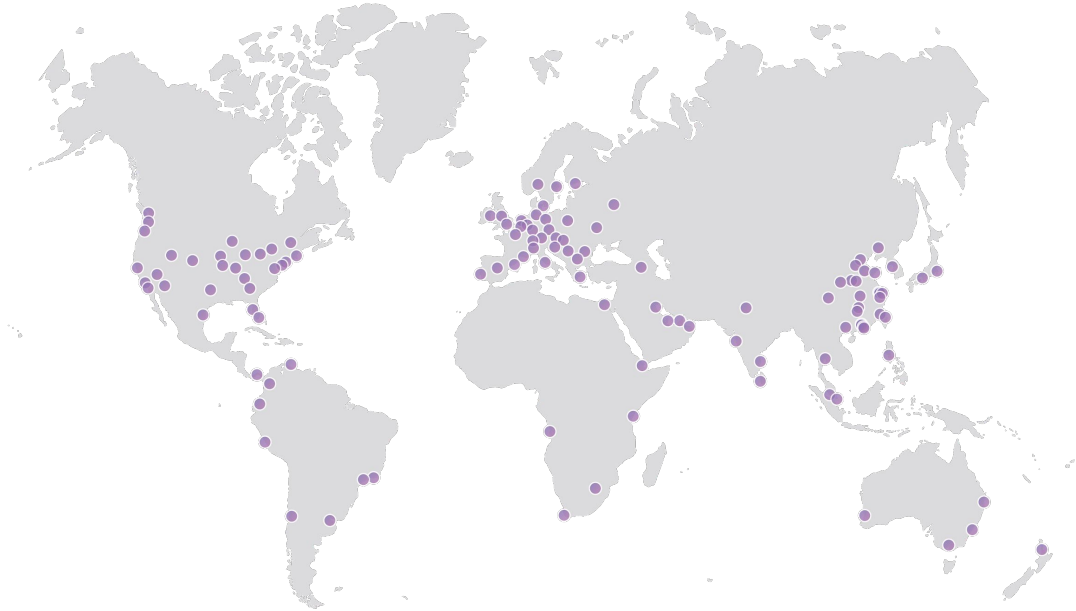
Content delivery network.

We are a DNS resolver.

We received Terabit/s attacks.

140+ PoP globally

170+ IXP presence



CLOUDFLARE

We monitor our network

- We are a CDN
    - We want to know an anomaly before the user notices
        - Alert and fix
    - We want to reduce transits costs
        - Way of serving the same bit for cheaper
    - We want to optimize our network
        - What are the main ISPs in a country
        - Better routes

CLOUDFLARE

Flows

Network samples

A **flow sample** contains **metadata information**:
source/destination IPs, interfaces, size of the payload, timestamp, ports, VLANs.

High cardinality ⇒ storage

Can aggregate to reduce size but losing information

High frequency ⇒ scalability

Depends on your sampling rate and total bandwidth

Building services ⇒ reliability

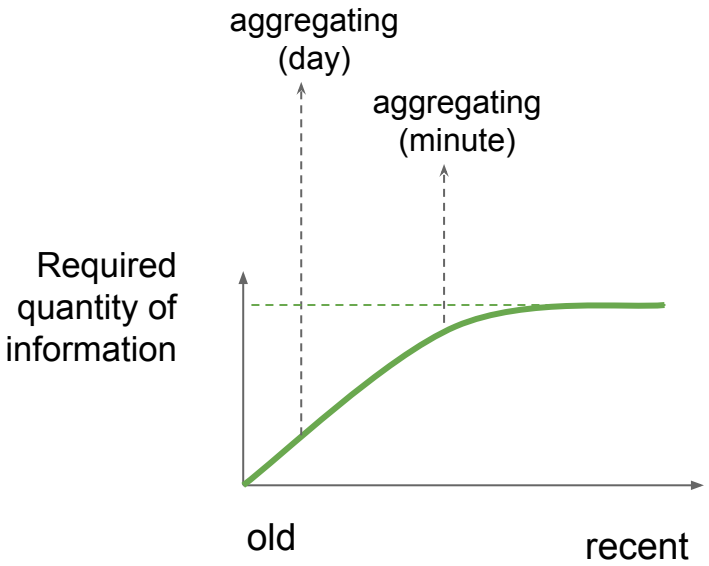
The need

Our existing pipeline was **monolithic**.

Good enough for **one-off lookups**.

Want to deploy new **services**
(automatic rerouting, periodic statistics)

Have a **compression** of data
(keep the maximum relevant)



CLOUDFLARE

Current limitations

No monitoring, storage, only aggregations corrected during bug

nfdump:
Stores singles flows. Aggregation on query.
→ Machine with dump files.

nfacctd:
Configured aggregations. Connects to BGP. Plenty of outputs.
→ Only aggregation. Restart necessary.
→ Performances issues.

Why custom tools and pipeline? (1/2)

We want to use our internal **cloud**:

- Containers
- Load-balanced IPs
- Storage (clusters)
- Message brokers

No more single point of failure (the unique machine that ran nfdump storing locally)

Increases **reliability**, **accessibility** and ease of **maintenance**

CLOUDFLARE

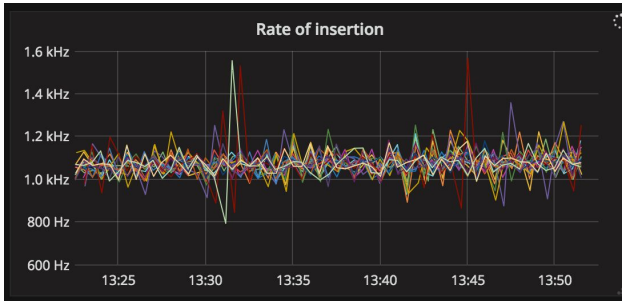
Why custom tools and pipeline? (2/2)

Reliability for data analytics:

- Traffic engineering based on live data
- Parallel tasks
- Monitoring of the flows delivery and processing
  - We were losing flows due to CPU issue

Other teams may want to **access** the data:

- Common file format
- Using common databases and tools



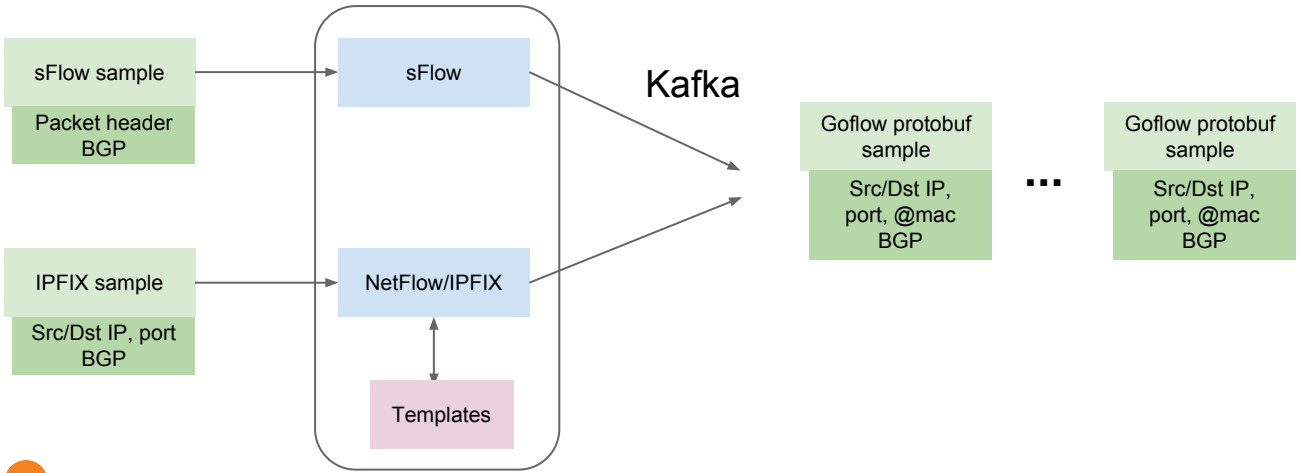
Insertion in database is distributed, rate is 1100 samples per second per container

CLOUDFLARE

GoFlow

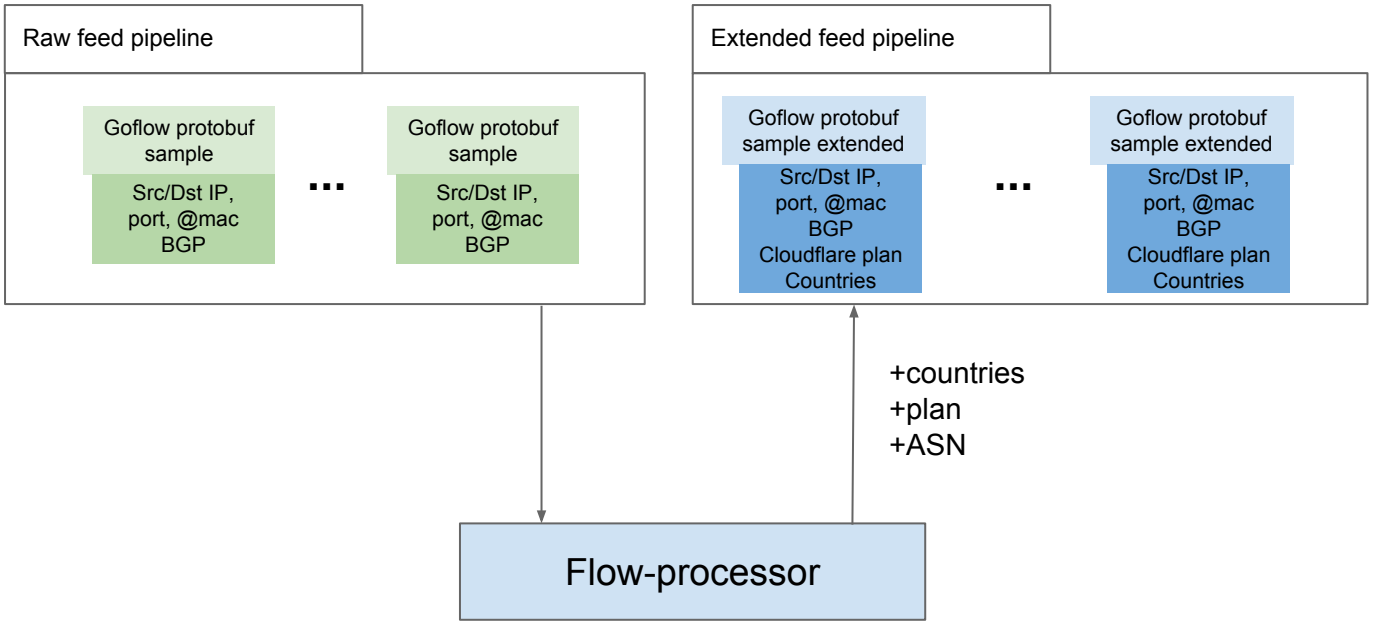
`https://github.com/cloudflare/goflow`

A NetFlow v9, IPFIX and sFlow decoder for network samples that pushes to Kafka and living in containers



CLOUDFLARE

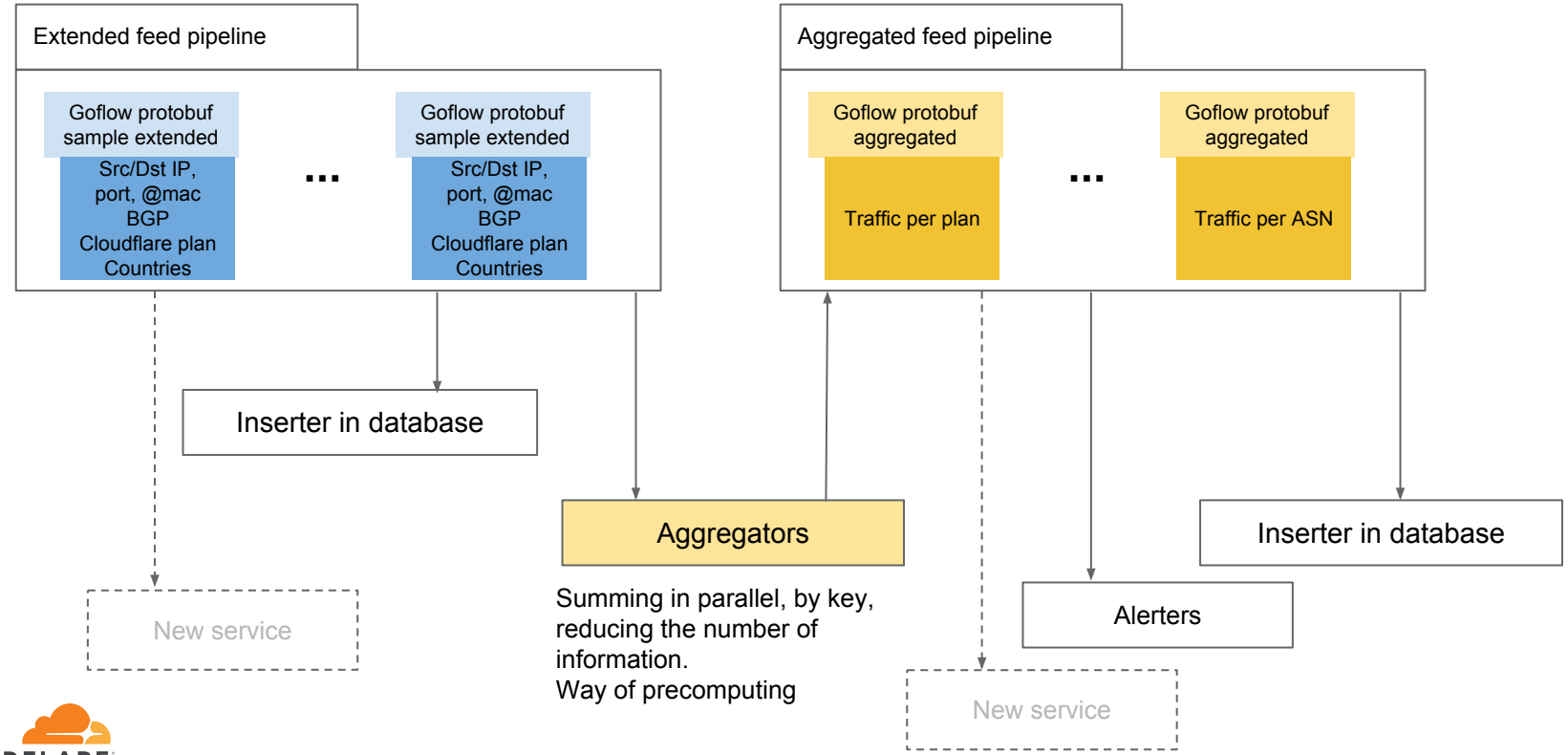
The pipeline



Add countries based on source/destination IPs
Add/correct BGP information using external source
Insert Cloudflare plan (free, pro, biz, enterprise…)

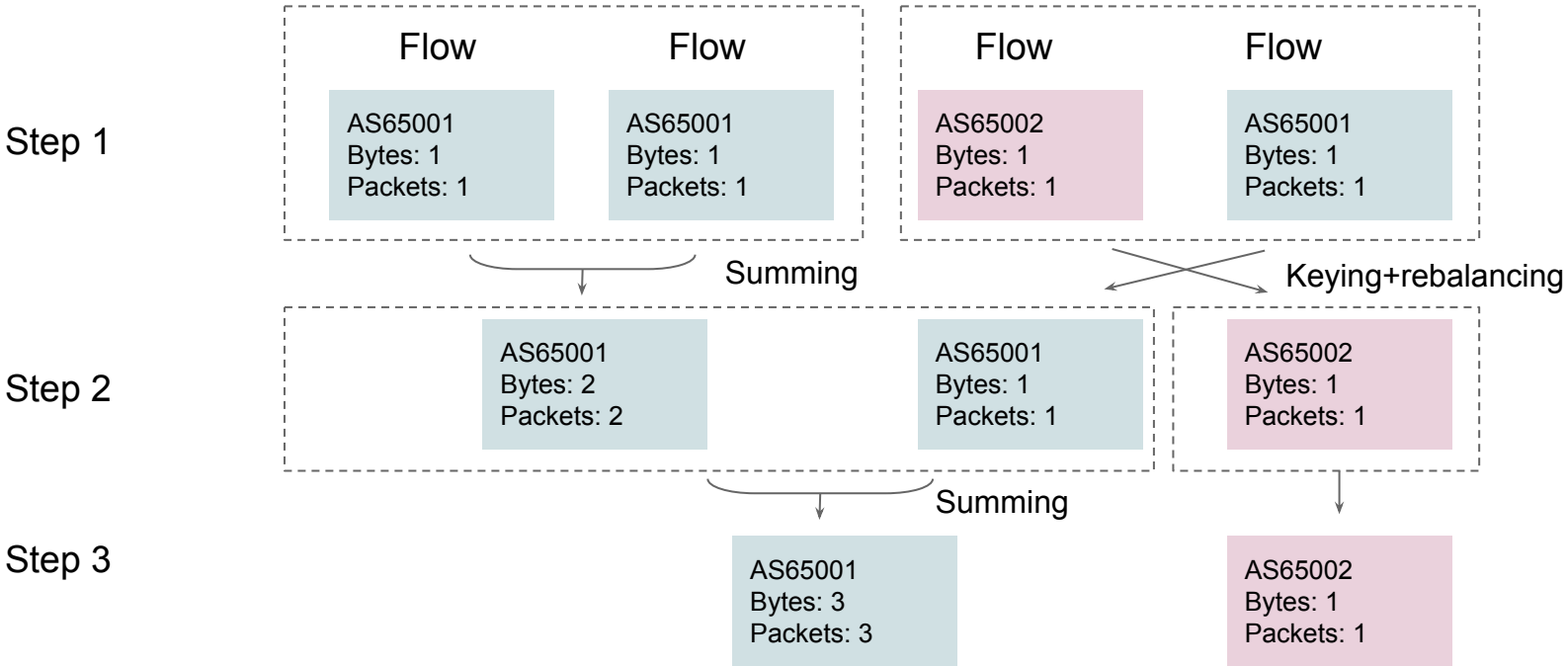
CLOUDFLARE

The pipeline



CLOUDFLARE

Aggregation - MapReduce



CLOUDFLARE

GoFlow - Who is it for?

If you want **flexibility** and integrate the network feed in a **data pipeline**.

You have to develop:

- **Flow processors**: 1:1 mappings of the flows (add country information, etc.)

- **Database inserters**: have a data warehouse (Clickhouse, Amazon RedShift, Google BigQuery)
  - We visualize it in Grafana
  - Specific teams to maintain access to Clickhouse

- **Aggregators**: pre-compute (reduce size by summing on keys) and allows to have a live feed
  - We use Flink

CLOUDFLARE

Flow tools - Comparison

All-in-one software solutions:

[n|s]fdump:

Decode flow samples (sFlow, IPFIX, NetFlow) and store them into a file on the disk.

Can be replayed.

Aggregation done on the fly. Files can be splitted by router/time.

[n|s]facctd:

Aggregates on specific fields, add data (ASN, countries), can forward the result to Kafka, a static file.

Flow tools - Comparison

Performances of GoFlow: on 2 CPU cores, around 20 000 flows packets per second.

Horizontal scalability possible. Only 30 microseconds for decoding.

Monitor using **Prometheus**.

Modulable:

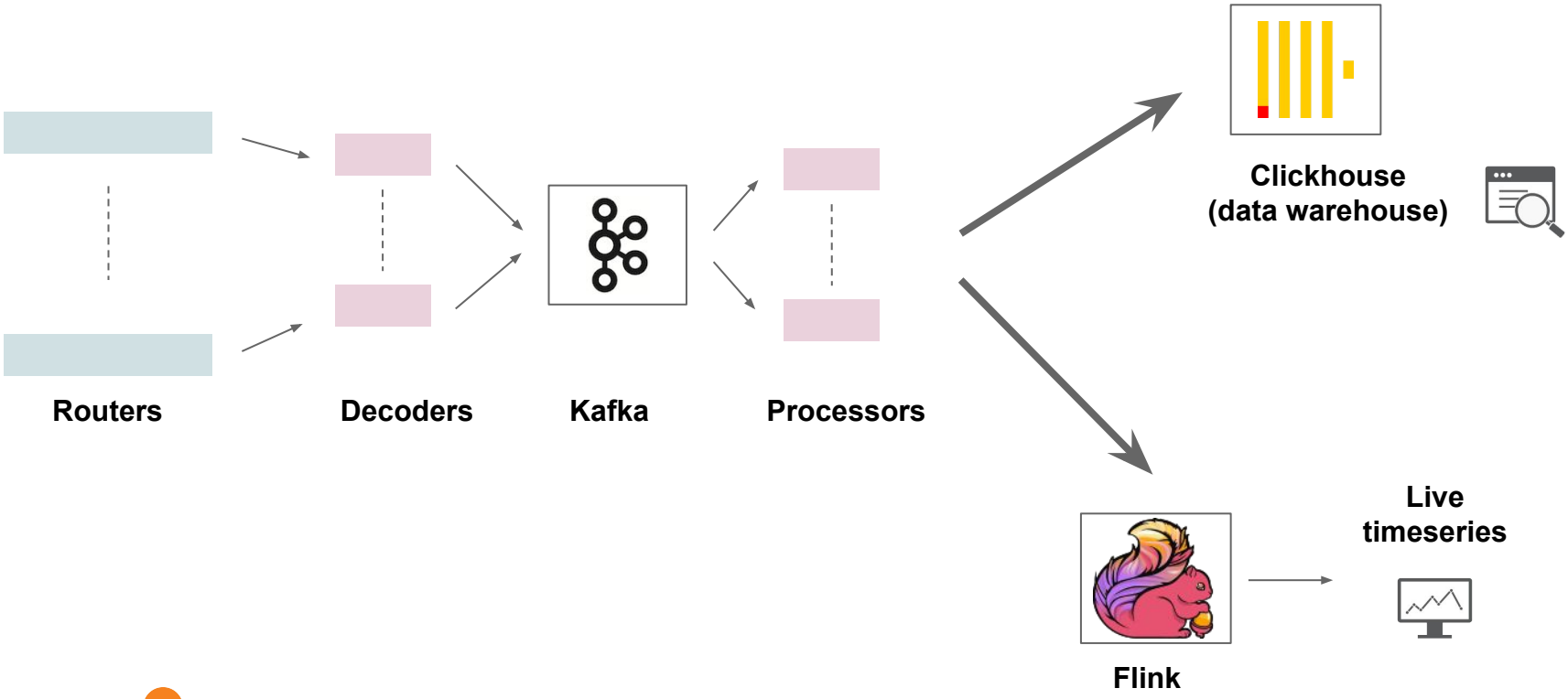
Eg: Create your own producers to send to RabbitMQ or use other NetFlow fields.

```
50
51  func (s KafkaState) SendKafkaFlowMessage(flowMessage flowmessage.FlowMessage) {
52          b, _ := proto.Marshal(&flowMessage)
53          s.producer.Input() <- &sarama.ProducerMessage{
54                  Topic: *KafkaTopic,
55                  Value: sarama.ByteEncoder(b),
56          }
57  }
```

```
203                 case netflow.NFV9_FIELD_IPV4_SRC_ADDR:
204                         flowMessage.IPversion = flowmessage.FlowMessage_IPv4
205                         flowMessage.SrcIP = v
206                 case netflow.NFV9_FIELD_IPV4_DST_ADDR:
207                         flowMessage.IPversion = flowmessage.FlowMessage_IPv4
208                         flowMessage.DstIP = v
209
```

CLOUDFLARE

What we built

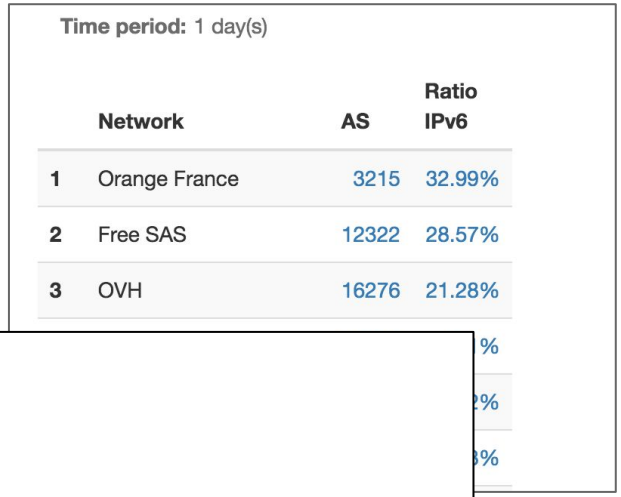


CLOUDFLARE

Results

API with statistics

Everything SQL query:



Time period: 1 day(s)

	Network	AS	Ratio IPv6
1	Orange France	3215	32.99%
2	Free SAS	12322	28.57%
3	OVH	16276	21.28%

```
$ flowquery -s dstport/bytes -f 'dstip = "1.1.1.1" '
SELECT dstPort AS dstport,
       count(*) AS numFlows,sum(packets*samplingRate) AS sumPkts,
       sum(bytes*samplingRate*8) AS sumbits,
       round(sum(packets*samplingRate/(86400*1000)),1) AS rateKpps,
       round(sum(bytes*samplingRate*8/(86400*1000000)),2) AS rateMbps
FROM netflows
WHERE date <= toDate('2018-03-24 00:00:00')
       AND timeFlow <= toDateTime('2018-03-24 23:59:59')
       AND date >= toDate(018-03-24 23:59:59)
       AND timeFlow >= toDateTime('2018-03-24 00:00:00)
       AND (if(dstIpv4 != 0, IPv4NumToString(dstIpv4), IPv6NumToString(dstIpv6)) = '1.1.1.1' )
GROUP BY dstport
ORDER BY sumbits DESC
LIMIT 10

      dstport    numFlows      sumPkts         sumbits    rateKpps    rateMbps
 0        443     2737526  46040068748  28191173342848         533         326
 1         80      422707   7467889690   8388815951552          86          97
 2       8000      433446   7297589502   6969577261264          84          81
 3        514      110813   4863679280   6564511089008          56          76
```

Results

Uses:

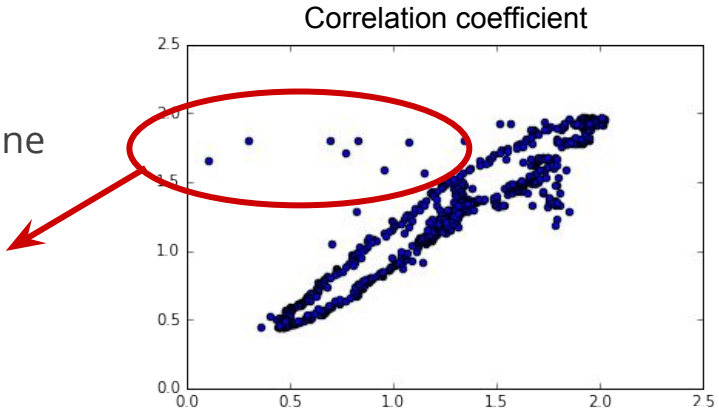
Prediction of network traffic and **anomaly detection**

Finding the best maintenance times per datacenter/timezone

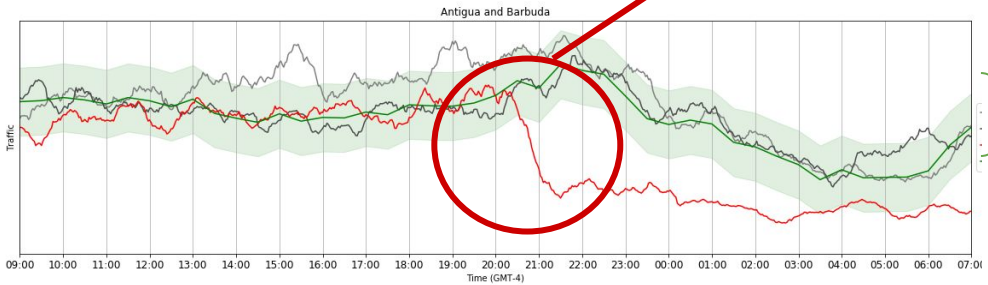
Market share of every ISP per country

Global IPv6 percentages

Transit vs peering reports



Outliers



Local variance following median

CLOUDFLARE

BGP

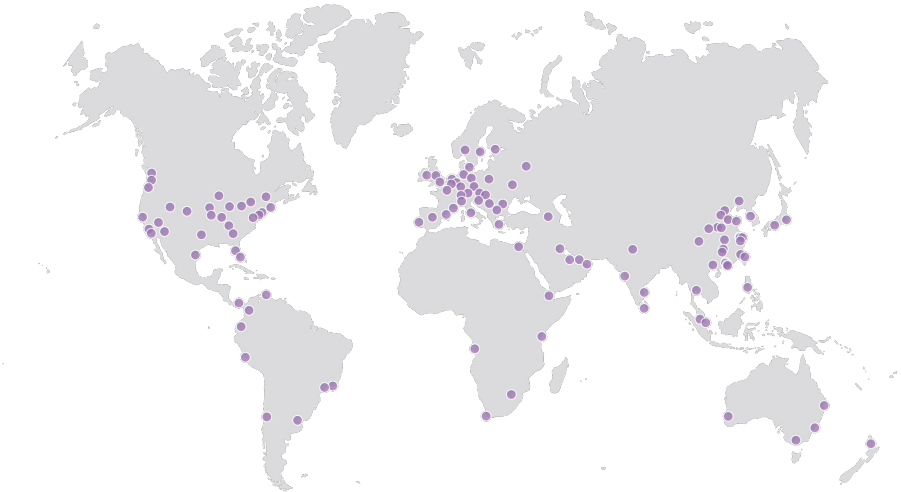
BGP collection

To add more **information** to the flow pipeline (prefix → ASN api).

Built a custom collector for our 140+ datacenters

`https://blog.cloudflare.com/durban-and-port-louis/`

Full tables: 740 000 routes * 140



CLOUDFLARE

BGP collection

Main issue:

- BGP collectors require a static IP, static configuration
  - One fixed machine that would stores the 60+ millions routes

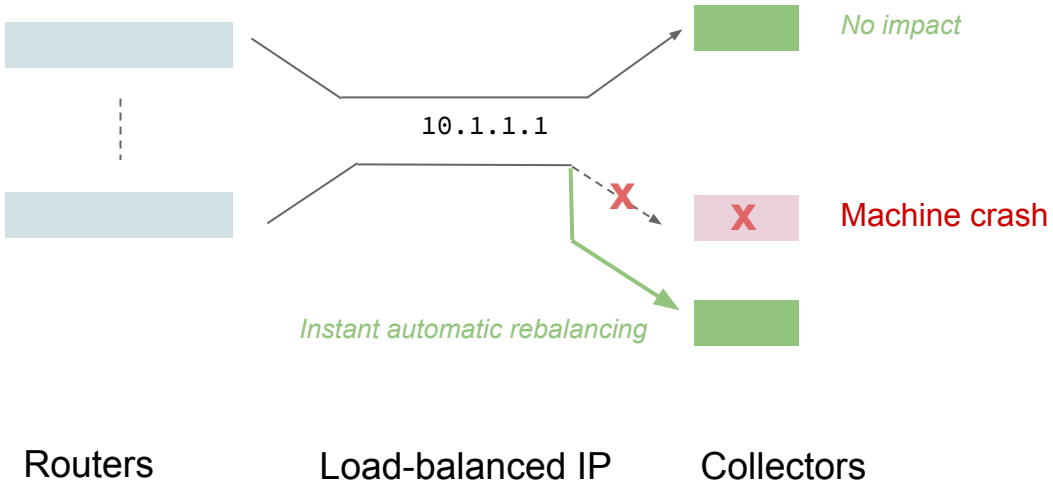
Solved:

- Developed a custom BGP server that only listens and accept connections then forwards updates in Kafka.
  - Removes problem of backpressure
  - Especially when generating full tables dumps

- Dedicated Docker containers for storing the full table and provide an API

BGP pipeline - Failover

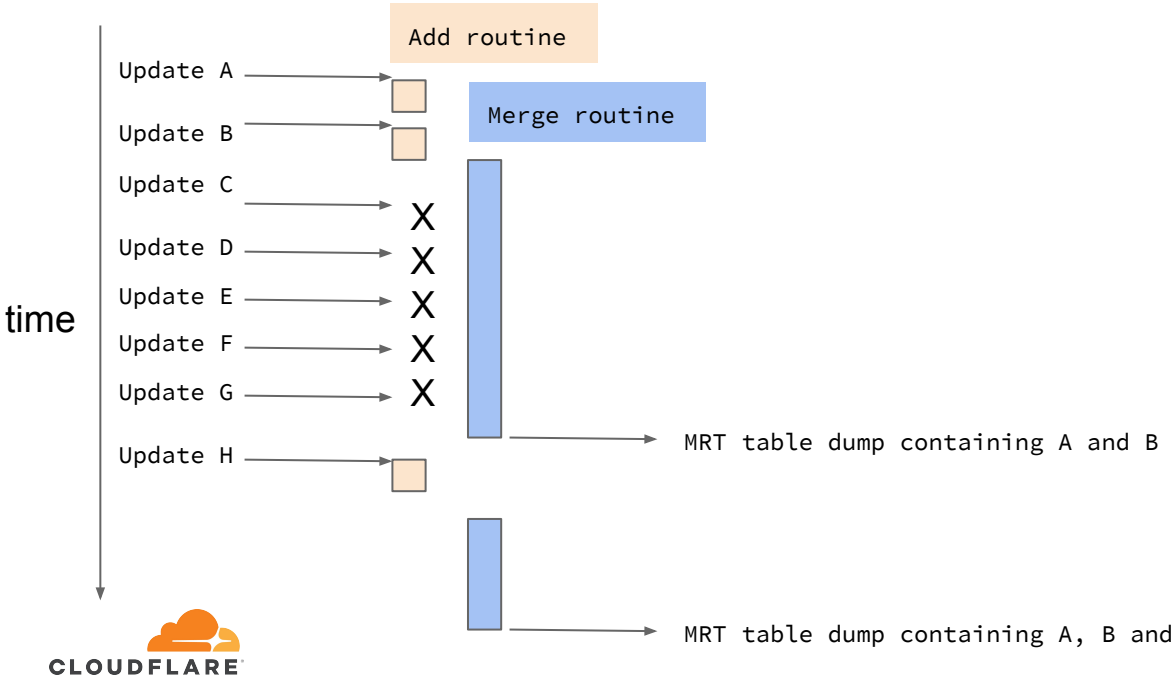
Failure handling (avoid resetting all sessions or losing all routes at once)



CLOUDFLARE

Backpressure, scale problems

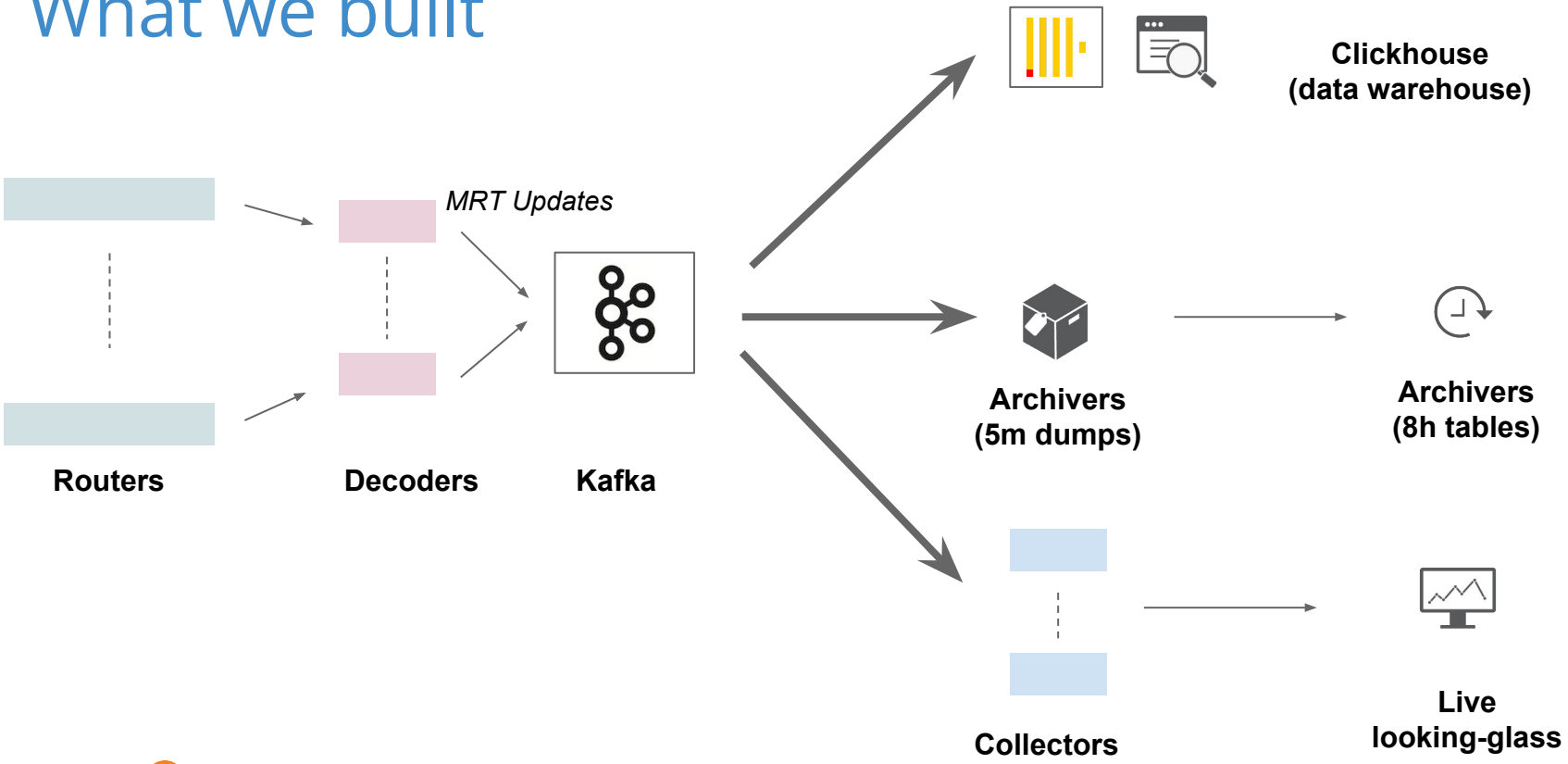
Asynchronous **receive** and **processing**



Overflowing pipe

CLOUDFLARE

What we built



CLOUDFLARE

Results

Live APIs.

Storage on S3-type-cluster (for static table analysis or update-processing)

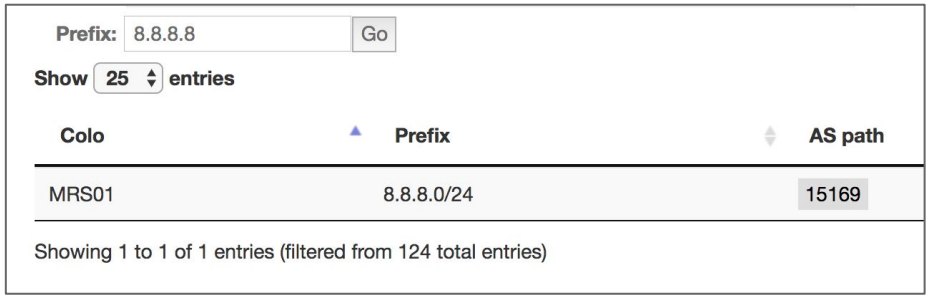
Provide Prefix → ASN information for the flow processing

Ideas for storing **MP-BGP EVPN** routes (mac addresses).

300 MB per full-table (total storage is around 40 GB in RAM over a dozen machines).

Development of a custom level-compressed trie in Go for storage

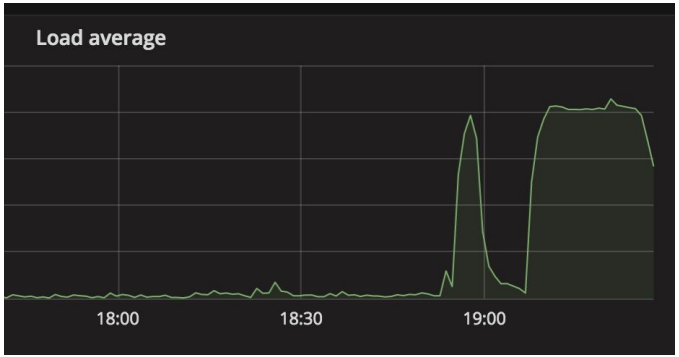
Distributed lookups: 1 millisecond
for a route over 140+ routers
(70 million routes).



CLOUDFLARE

Random fact

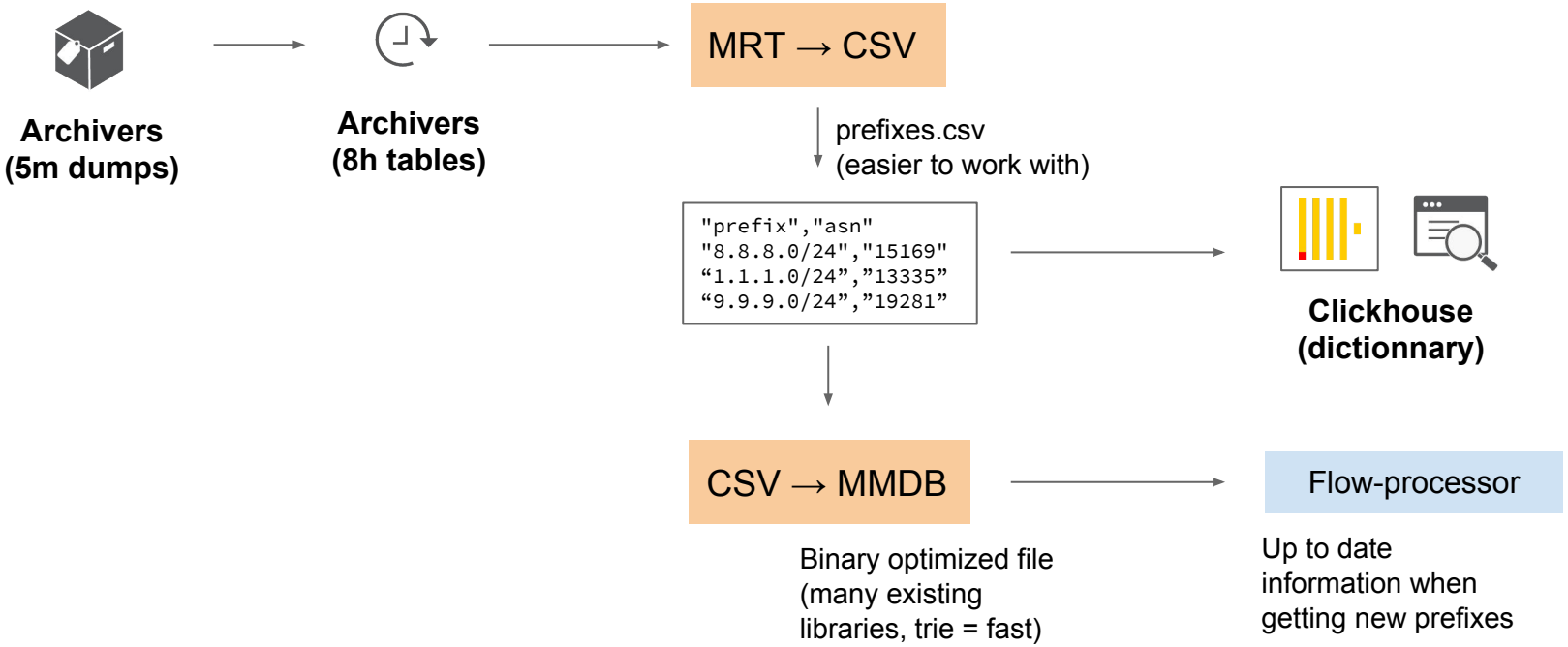
- People sending us IX LAN prefixes
- Receiving smaller than /48 IPv6 and smaller than /24 IPv4
- Longest AS-Path
  - 2402:8100:3980::/42 → 37 ASNs



The converter using 30GB of RAM and 20 CPU for an hour

CLOUDFLARE

Results



CLOUDFLARE

The BGP library

`https://github.com/cloudflare/fgbgp`

Features:

- Open/maintain/accept BGP connection
- Decode/encode BGP messages
- Decode/encode MRT updates or Table Dumps
- Maintain a RIB
- Event-driven API

You implement the behavior.

It is not an automatable client.

```
func NewMrtBGP4MP_StateChange_AS4(peeras uint32, localas u
        return &MrtBGP4MP_StateChange_AS4{
                Timestamp:  time.Now().UTC(),
                PeerAS:     peeras,
                LocalAS:    localas,
                IfaceIndex: iface,
                PeerIP:     peerip,
                LocalIP:    localip,
                OldState:   oldstate,
                NewState:   newstate,
        }
```

CLOUDFLARE

More to come

Coming soon. More code examples, docker-compose, inserters.

One last tool:

`https://github.com/cloudflare/py-mmdb-encoder`

Create your own mmdb files using Python (IP to country, IP to ASN, IP to anything).

Questions?

Thank you

louis@cloudflare.com
@lspgn
traceroute6 cv6.poinsignon.org