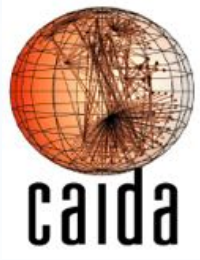
BGPSTREAM

A framework for BGP data analysis



Alberto Dainotti, Alistair King, **Chiara Orsini**, Vasco Asturiano

chiara@caida.org

BGPSTREAM

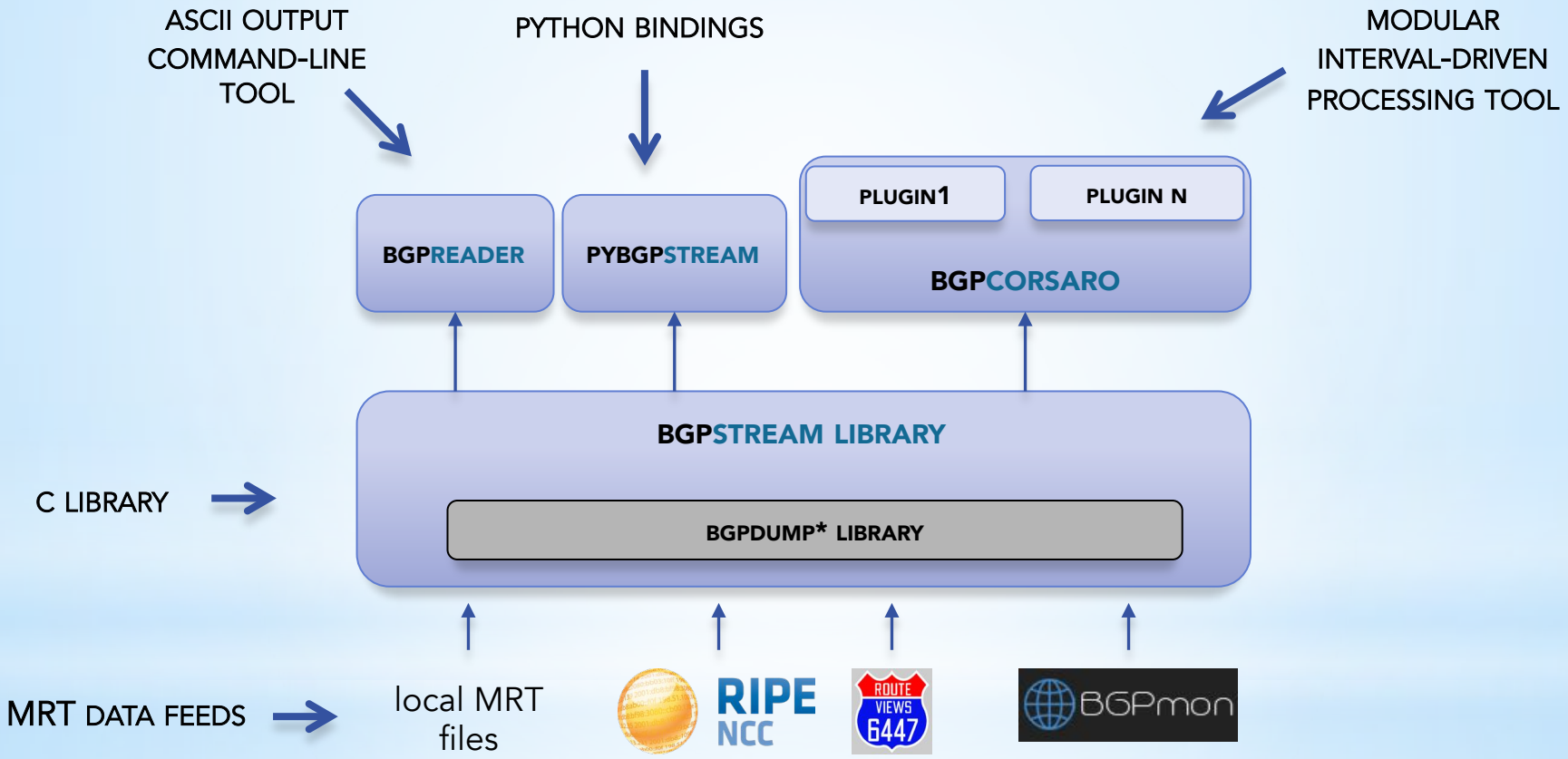
A software framework for the historical analysis and real-time monitoring BGP data

Goals →

* generate a sorted stream to support maintaining a BGP "state" over time
* abstract from underlying data sources
* filters BGP data based on user needs
* tag unreliable BGP data
* support real-time

* work in progress, soon to be released as open-source
* *v1 release expected for this summer*

BGPSTREAM framework



Data Feeds

Transparent access to several *annotated* MRT data sources:

* Previously-downloaded **local files**
* **Real-time stream** from:
  * Colorado State's BGPmon (all Route
    [work-in-progress for release v1]
  * RIPE RIS *[discussion in-progress]*

BGPDOWNLOADER

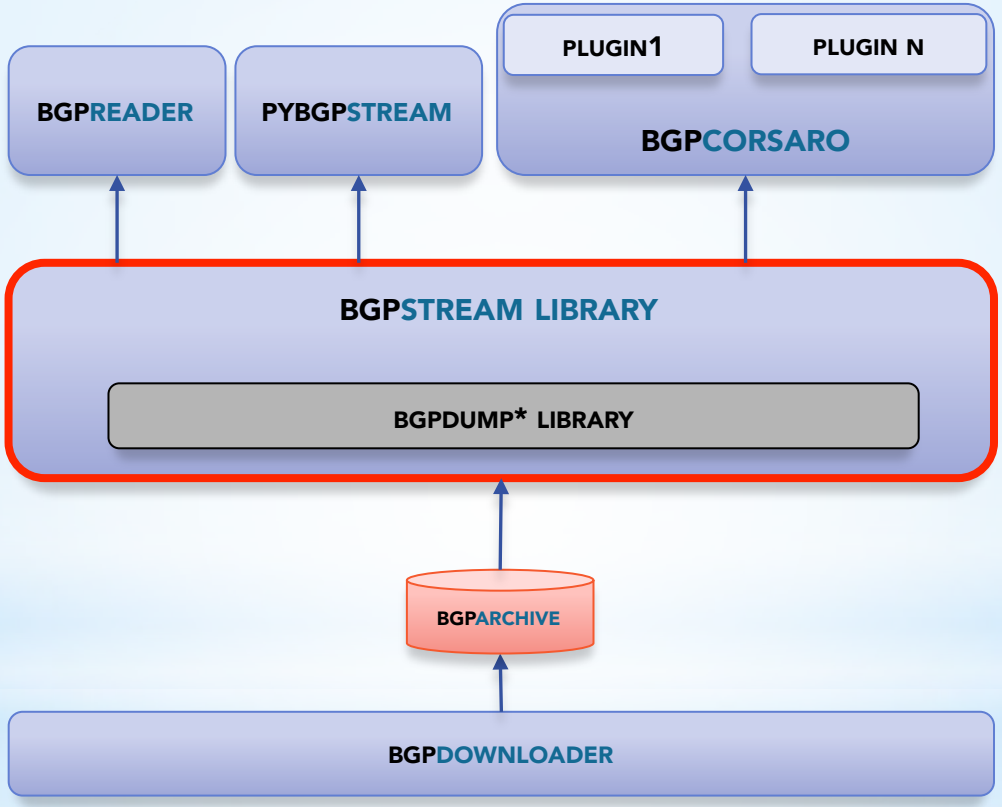
* Perl program
* ~20 mins average delay
* meta data into a **BGPARCHIVE** (mySQL DB)
* MRT files stored on hard disk

* **Historical and continuous download** from RIPE RIS and RouteViews projects

* 13 active collectors
* RIBS every 8 hours
* Updates every 5 minutes

* 17 active collectors
* RIBS every 2 hours
* Updates every 15 minutes

BGPSTREAM framework



BGPSTREAM library

① access the MySQL **BGPARCHIVE** and select files based on
   * project
   * type
   * collector
   * time

② use a modified version of **BGPDUMP** [1] to open group of dump files in parallel

③ extract **BGPRECORDS** from these files, i.e. wrappers around the **BGPDUMP ENTRY** format

④ marshal the **BGPRECORDS** according to their timestamp

⑤ optionally unwrap **BGPRECORDS** and extract atomic BGP information called **BGPELEMS**

[1]: https://bitbucket.org/ripencc/bgpdump/wiki/Home

BGPRECORD

* PROJECT
* BGP TYPE
* COLLECTOR
* DUMP TIME
* DUMP POSITION
* RECORD TIME
* RECORD STATUS
* **BGPDUMP ENTRY**

BGPARCHIVE metadata (common to entire dump)

position of entry in dump
- START
- MIDDLE
- END

time associated with the **BGPDUMP ENTRY**

status of **BGPRECORD**
- VALID
- CORRUPTED RECORD
- EMPTY SOURCE
- CORRUPTED SOURCE

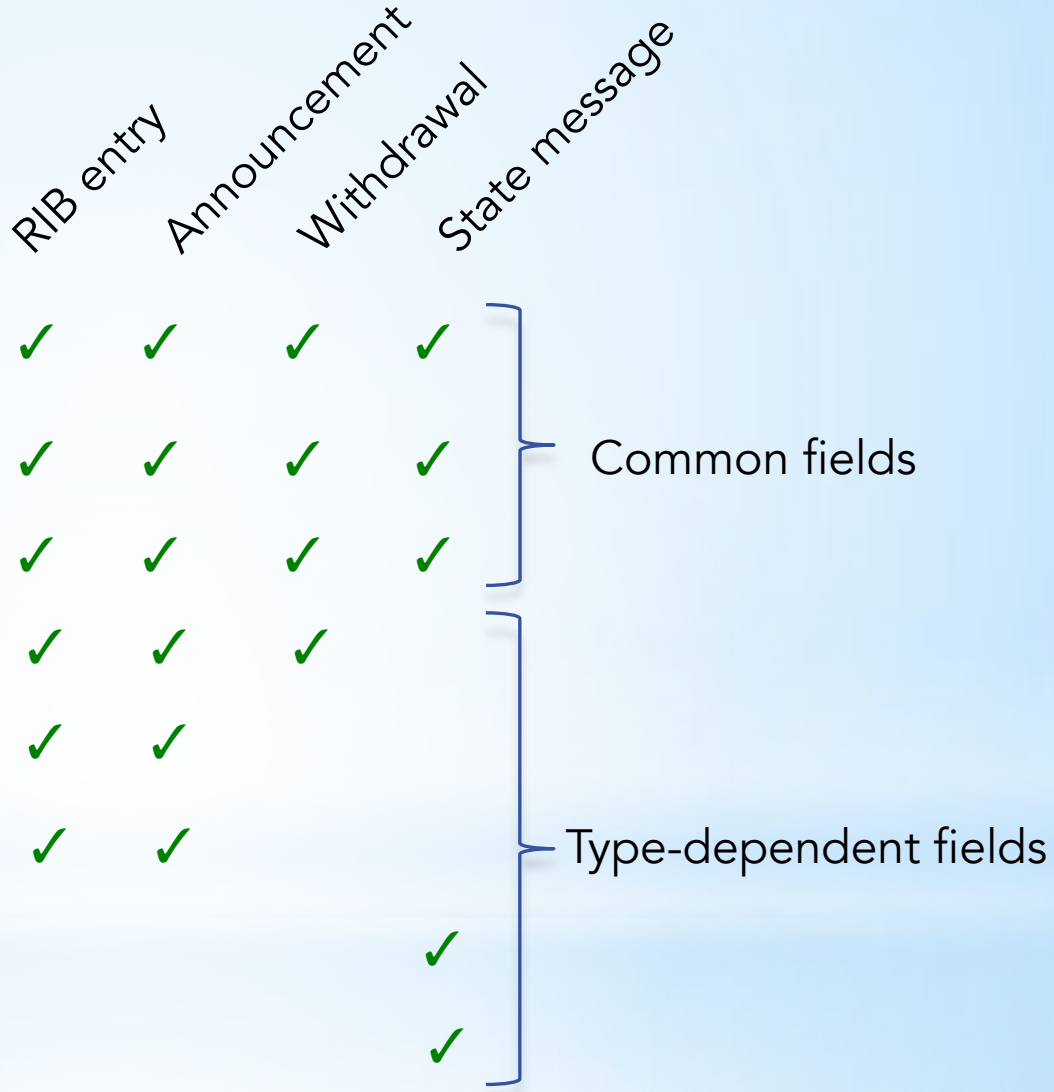
set of MRT formatted entries

BGPRECORD → BGPELEM

* PROJECT
* BGP TYPE
* COLLECTOR
* DUMP TIME
* DUMP POSITION
* RECORD TIME
* RECORD STATUS
* **BGPDUMP ENTRY**

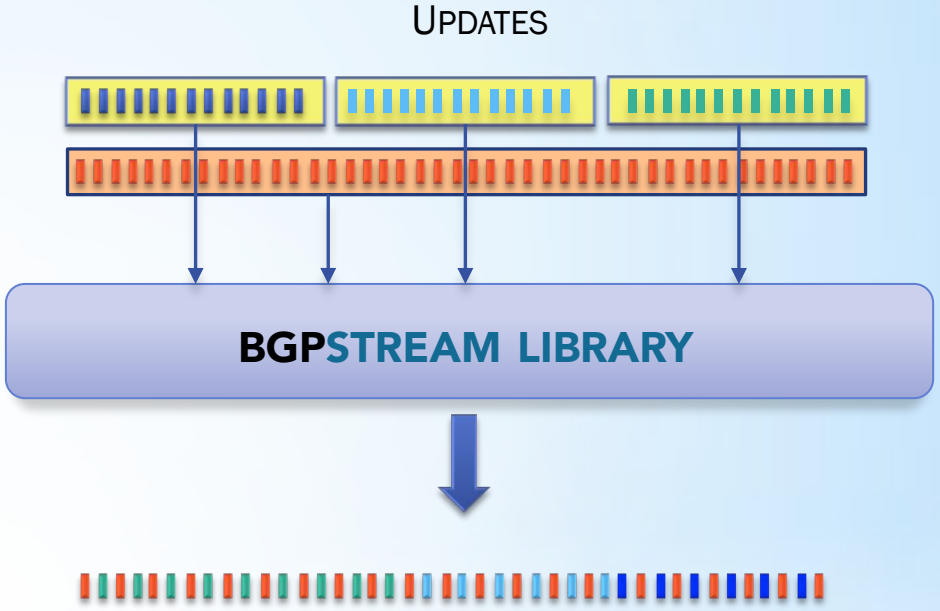
BGPELEM

- * TYPE
- * TIMESTAMP
- * PEER IP ADDRESS
- * PEER AS NUMBER
- * IP PREFIX
- * NEXT HOP
- * AS PATH
- * OLD STATE
- * NEW STATE



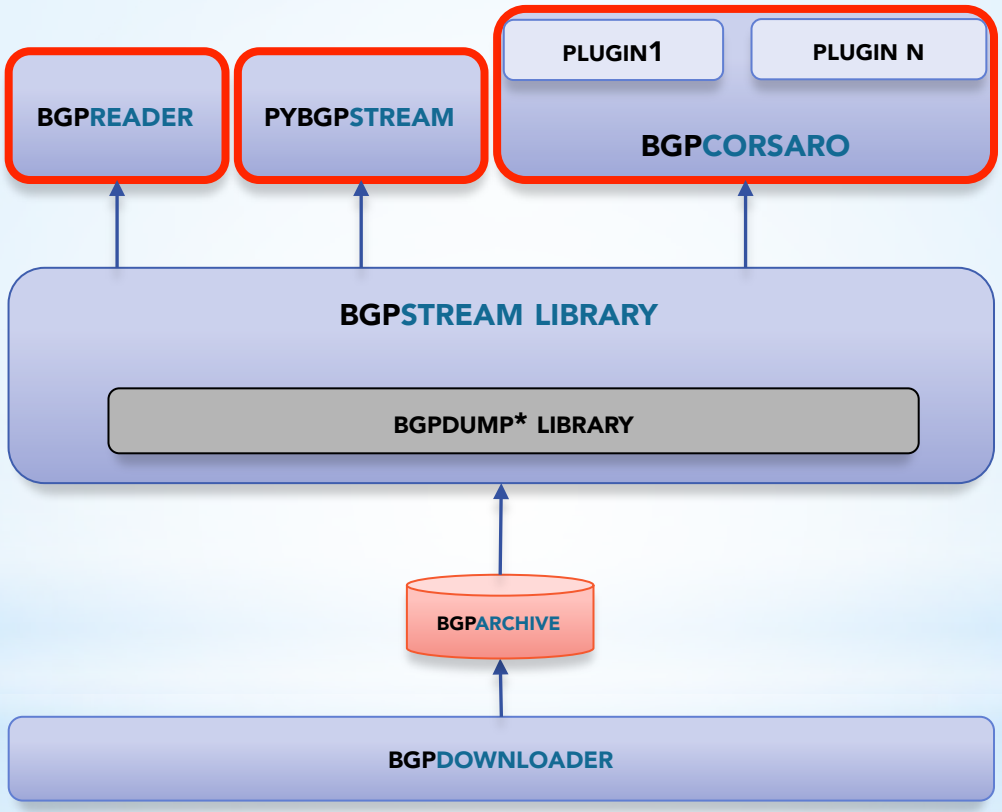
BGPSTREAM library

```c
#include "bgpstream_lib.h"

int main(int argc, char *argv[])
{
  bgpstream_t * bs = bgpstream_create();
  bgpstream_record_t *rec = \
              bgpstream_create_record();

  bgpstream_add_filter(bs, BS_COLLECTOR, "rrc00");
  bgpstream_add_filter(bs, BS_COLLECTOR, "route-views2");
  bgpstream_add_filter(bs, BS_BGP_TYPE, "updates");

  bgpstream_add_interval_filter(bs, BS_TIME_INTERVAL,
                                    "1410285600",
                                    "1412886500");

  int init_res = bgpstream_init(bs);
  while(bgpstream_get_next_record(bs, rec) > 0)
  {
     // [[ USE BGPRECORD HERE ]]
  }

  bgpstream_close(bs);
  bgpstream_destroy_record(rec);
  bgpstream_destroy(bs);
  return 0;
}
```

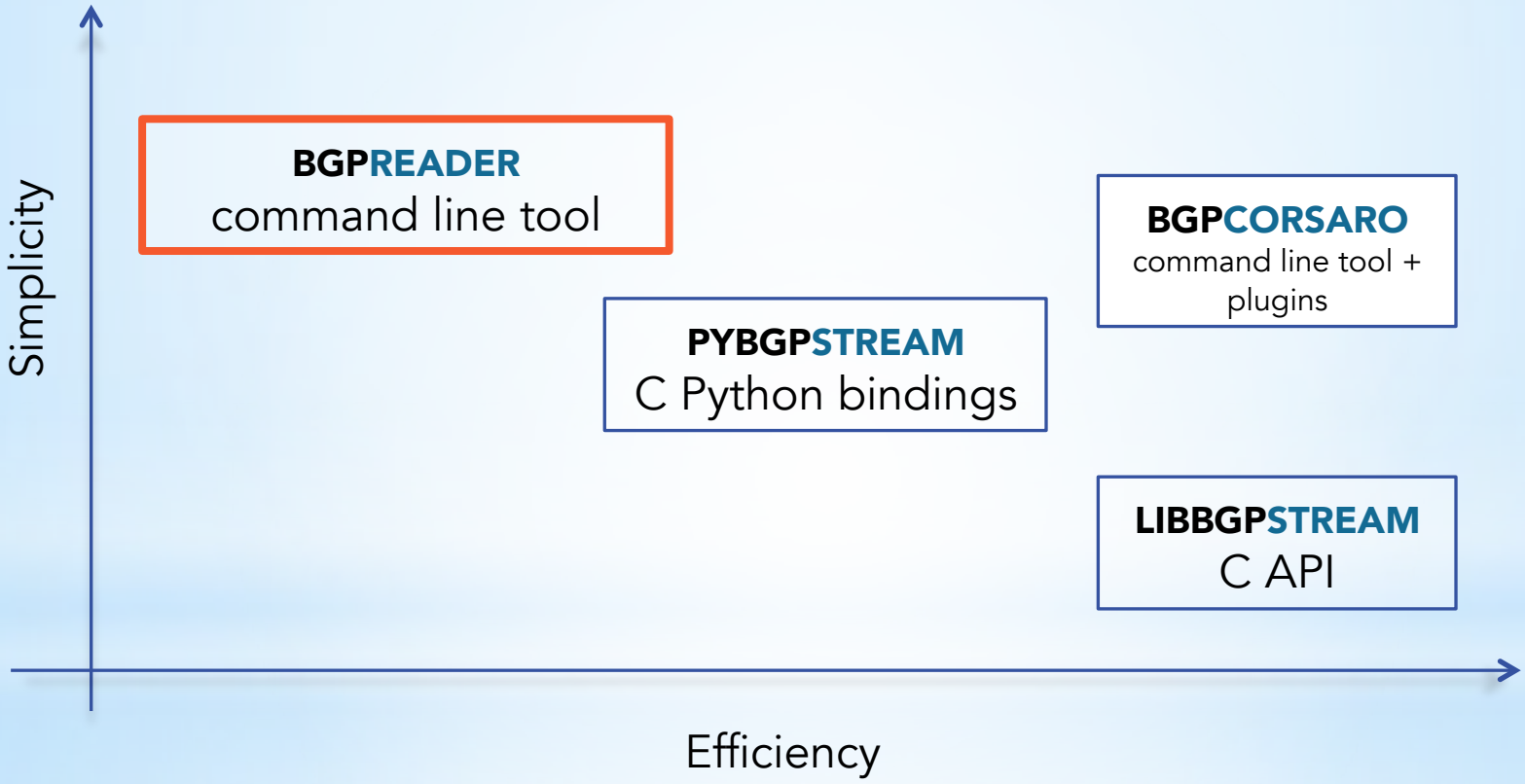


* rely on metadata to decide how many dumps to open in parallel
* sort based on **BGPRECORD** time

BGPSTREAM framework



BGPSTREAM just a C library?

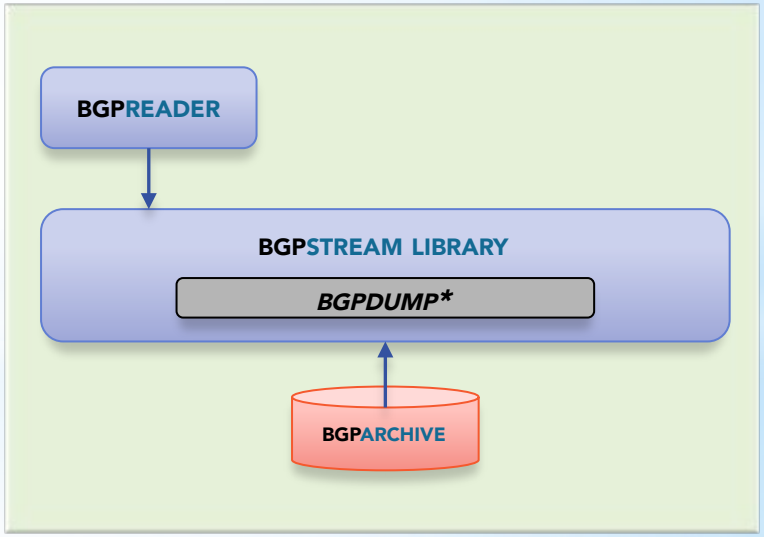


BGPREADER

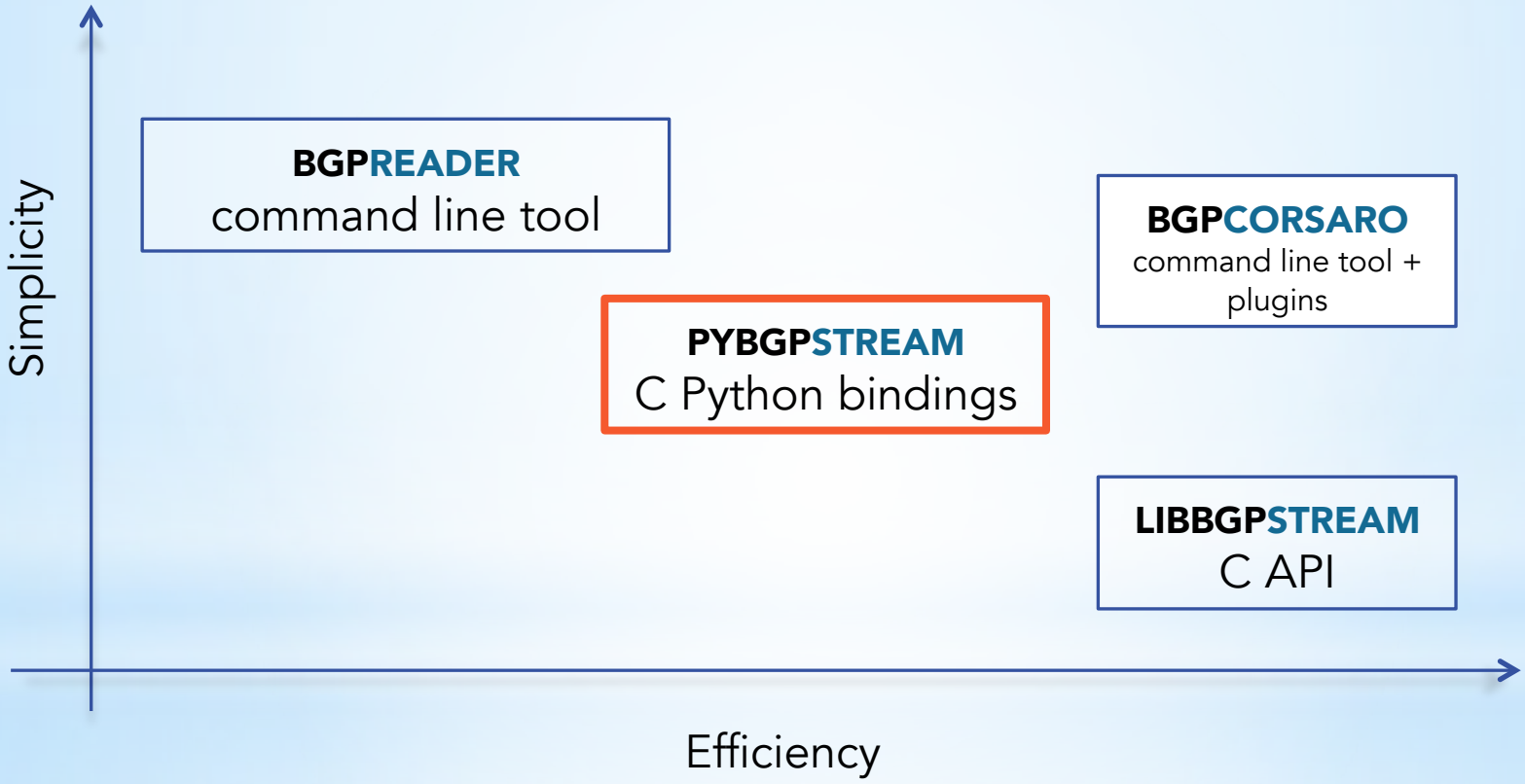
Metadata filters

```
$ bgpreader -C rrc00 -C rrc03 -W1407808260,1407808440 -T updates -m
...
1407808270|195.69.145.167|6453|A|202.70.88.0/21|195.69.145.167|6453 3549 9304 23752|23752||
1407808270|218.189.6.2|9304|A|202.70.88.0/21|218.189.6.2|9304 6453 23752|23752||
1407808270|12.0.1.63|7018|A|202.70.88.0/21|12.0.1.63|7018 6453 23752|23752||
1407808270|195.69.145.167|6453|A|202.70.64.0/21|195.69.145.167|6453 23752|23752||
1407808270|193.0.0.56|3333|A|202.70.88.0/21|193.0.0.56|3333 1257 6453 23752|23752||
1407808270|195.69.144.200|12859|A|202.70.88.0/21|...
1407808270|213.200.87.254|3257|A|190.55.32.0/20|...
1407808270|213.200.87.254|3257|A|186.23.96.0/20|...
1407808270|213.200.87.254|3257|A|190.55.48.0/20|...
1407808270|213.200.87.254|3257|A|186.23.240.0/20|...
1407808270|213.200.87.254|3257|A|186.23.160.0/20|...
1407808270|213.200.87.254|3257|A|186.23.208.0/20|..
...
```

* **BGPDUMP** compatible output
* **BGPREADER** output

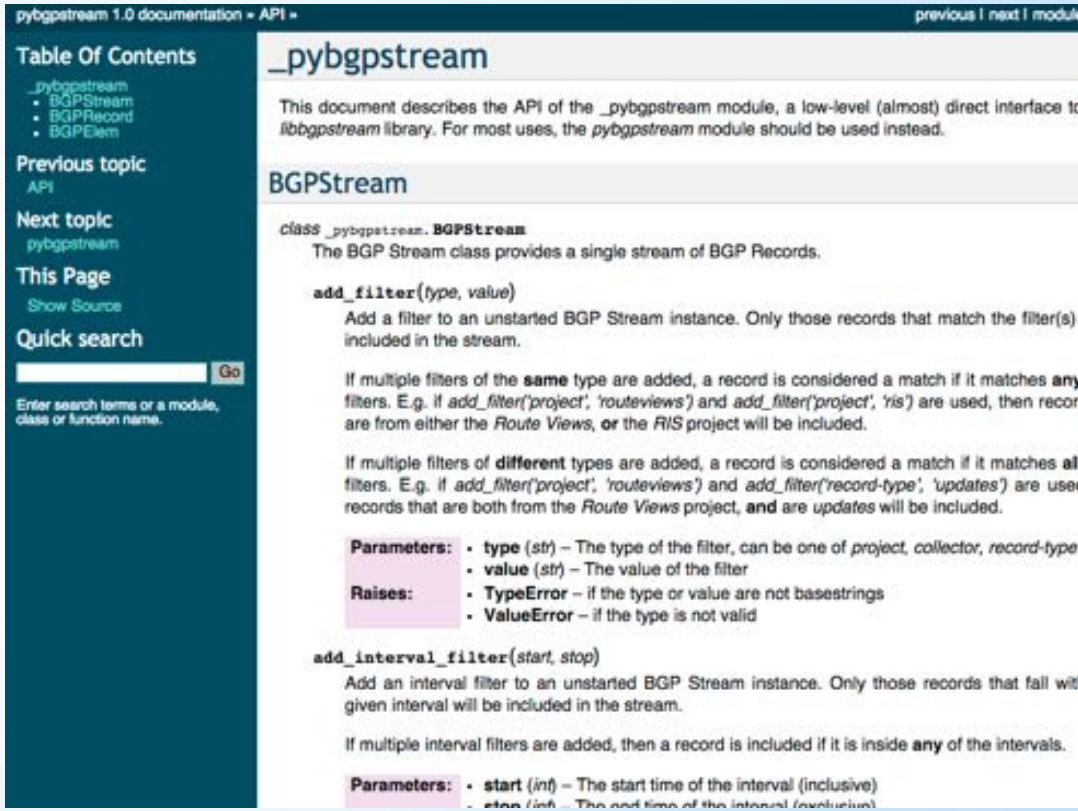


BGPSTREAM just a C library?



PYBGPSTREAM

* Python bindings
* same API exported in C
* no functionalities are lost



PYBGPSTREAM

What's the AS topology seen by collector Y?

- collector rrc00 on Thu, 30 Apr
  - 1 RIB file
  - 8,205,994 RIB entries
  - 108,197 unique AS adjacencies

2m:09s

- all RIS collectors on Thu, 30 Apr
  - 13 RIB files
  - 57,690,921RIB entries
  - 164,739 unique AS adjacencies

15m:18s

```
#stream.add_filter('collector','rrc00')
```

```
#!/usr/bin/env python

from _pybgpstream import BGPStream, BGPRecord, BGPElem

stream = BGPStream()
rec = BGPRecord()

as_topology = set()
rib_entries = 0

# Select datasource
stream.set_data_interface('mysql')

# Apply filters
stream.add_filter('project', 'ris')
stream.add_filter('collector', 'rrc00')
stream.add_filter('record-type', 'ribs')
# Wed, 29 Apr 2015 23:50:00 GMT -> Thu, 30 Apr 2015 00:10:00 GMT
stream.add_interval_filter(1430351400, 1430352600)

stream.start()

# Process data
while(stream.get_next_record(rec)):
    elem = rec.get_next_elem()
    while(elem):
        rib_entries += 1
        path = elem.fields['as-path']
        ases = path.split(" ")
        for i in range(0,len(ases)-1):
            if(ases[i] != ases[i+1]):
                as_topology.add(tuple(sorted([ases[i],ases[i+1]])))
        elem = rec.get_next_elem()

        # Output results
print "Processed ", rib_entries, " rib entries"
print "Found ", len(as_topology), " AS adjacencies"
```

PYBGPSTREAM

What is the number of MOAS (multi origin AS) prefix events observed in a 3 hours period?

- 1 collector: rrc00
  - 1 RIB file + 36 update files
  - 3,824 MOAS events

4m:57s

- all RIS collectors (13)
  - 13 RIB files + 468 update files
  - 6671 MOAS events

53m:16s

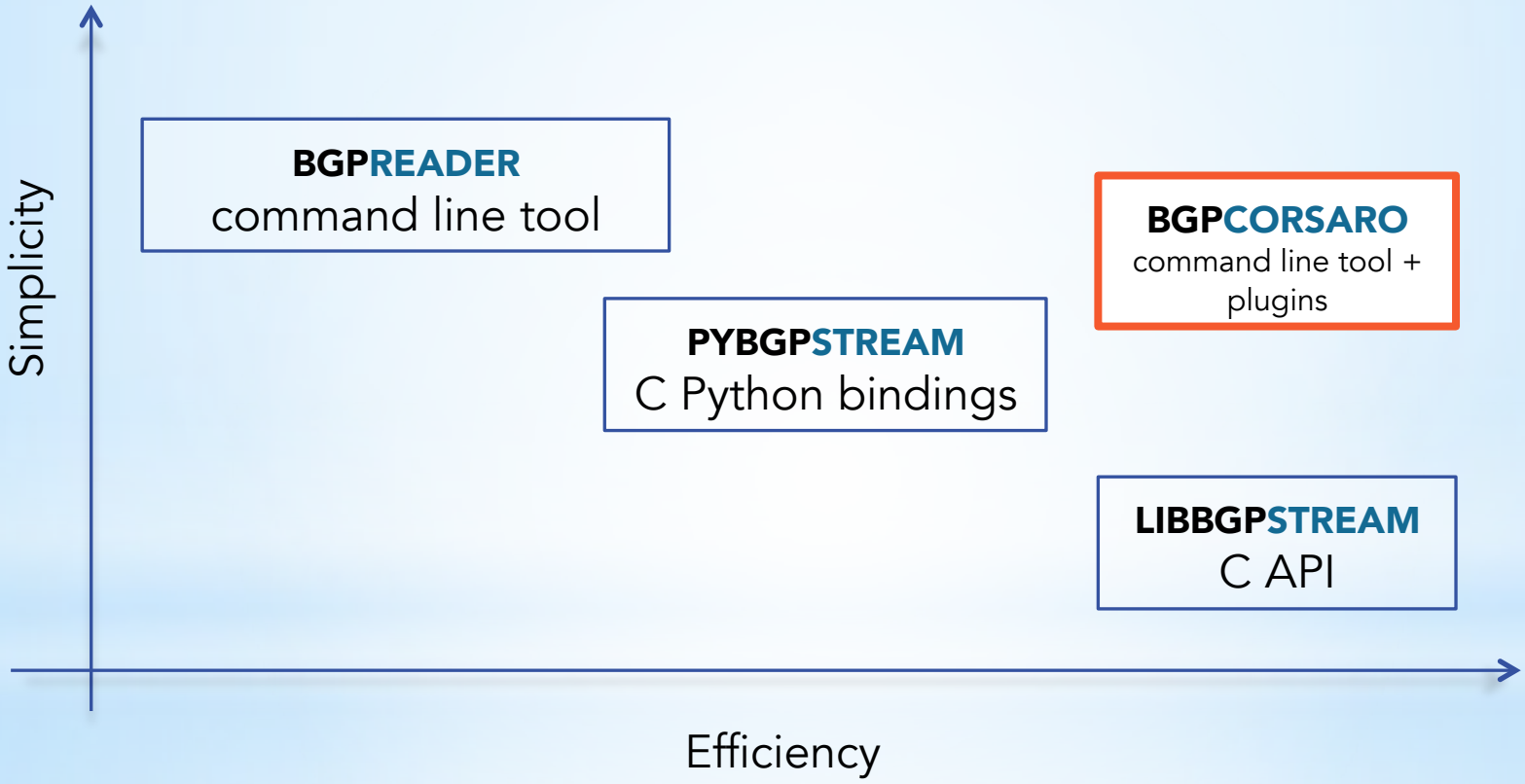
What if I want to do it in real time?

- end time in future
- just add one more line of configuration!

```
stream.set_blocking()
```

```python
#!/usr/bin/env python

from _pybgpstream import BGPStream, BGPRecord, BGPElem

# pfx -> list of peers observing the prefix
pfx_peers_asn = {}
# signature {collector}{ip}{asn} -> id
info_id = {}
# last id assigned
last_id = -1

# get id associated with current peer
def get_peer_id(col, ip, asn):
    global info_id
    global last_id
    if col not in info_id:
        info_id[col] = {}
    if ip not in info_id[col]:
        info_id[col][ip] = {}
    if asn not in info_id[col][ip]:
        last_id = last_id + 1
        info_id[col][ip][asn] = last_id
    peer_id = info_id[col][ip][asn]
    return peer_id

# add prefix peer origin
def add_prefix_peer_origin(ts, pfx, peer, asn):
    global pfx_peers_asn
    global peer_info
    moas = 0
    already_there = 0
    if pfx not in pfx_peers_asn:
        pfx_peers_asn[pfx] = {}
    for p in pfx_peers_asn[pfx]:
        if p != peer and pfx_peers_asn[pfx][p] != asn:
            moas = 1
        if pfx_peers_asn[pfx][p] == asn:
            already_there = 1
    if int(peer) not in pfx_peers_asn[pfx]:
        pfx_peers_asn[pfx][int(peer)] = asn
    else:
        pfx_peers_asn[pfx][int(peer)] = asn
    if already_there ==0 and moas == 1:
        print ts, "MOAS", pfx, pfx_peers_asn[pfx]

# remove prefix peer (i.e. this prefix is not seen by this peer anymore)
def remove_prefix_peer(pfx, peer):
    global pfx_peers_asn
    global peer_info
    if pfx not in pfx_peers_asn:
        return
    if peer not in pfx_peers_asn[pfx]:
        return
    del pfx_peers_asn[pfx][int(peer)]

# remove peer (i.e. this peer is not active anymore)
def remove_peer(peer):
    global pfx_peers_asn
    for pfx in pfx_peers_asn:
        if int(peer) in pfx_peers_asn[pfx]:
            del pfx_peers_asn[pfx][int(peer)]

stream = BGPStream()
rec = BGPRecord()
stream.set_data_interface('mysql')
stream.set_data_interface_option('mysql', 'db-host', 'loki-ge')
stream.set_data_interface_option('mysql', 'db-port', '3306')
stream.set_data_interface_option('mysql', 'db-user', 'bgpstream')
stream.add_filter('project', 'ris')
stream.add_filter('collector', 'rrc00')
stream.add_interval_filter(1430351400, 1430352600)
stream.start()

while(stream.get_next_record(rec)):
    elem = rec.get_next_elem()
    while(elem):
        # get peer information
        peer_id = get_peer_id(rec.collector, elem.peer_address, elem.peer_asn)
        # apply bgp message
        if(elem.type == 'R' or elem.type == 'A'):
            path = elem.fields['as-path']
            ases = path.split(" ")
            add_prefix_peer_origin(rec.time, elem.fields['prefix'], peer_id, ases[-1])
        if(elem.type == 'W'):
            remove_prefix_peer(elem.fields['prefix'], peer_id)
        if(elem.type == 'S' and elem.fields['new-state'] != "established"):
            remove_peer(peer_id)
        # get next element
        elem = rec.get_next_elem()
```

BGPSTREAM just a C library?

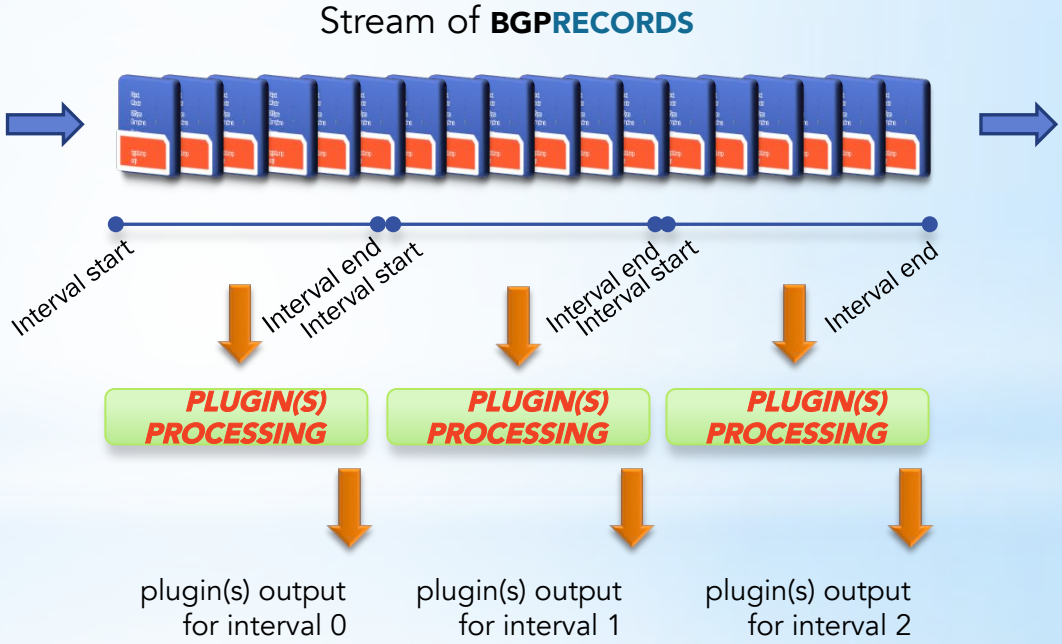


BGPCORSARO

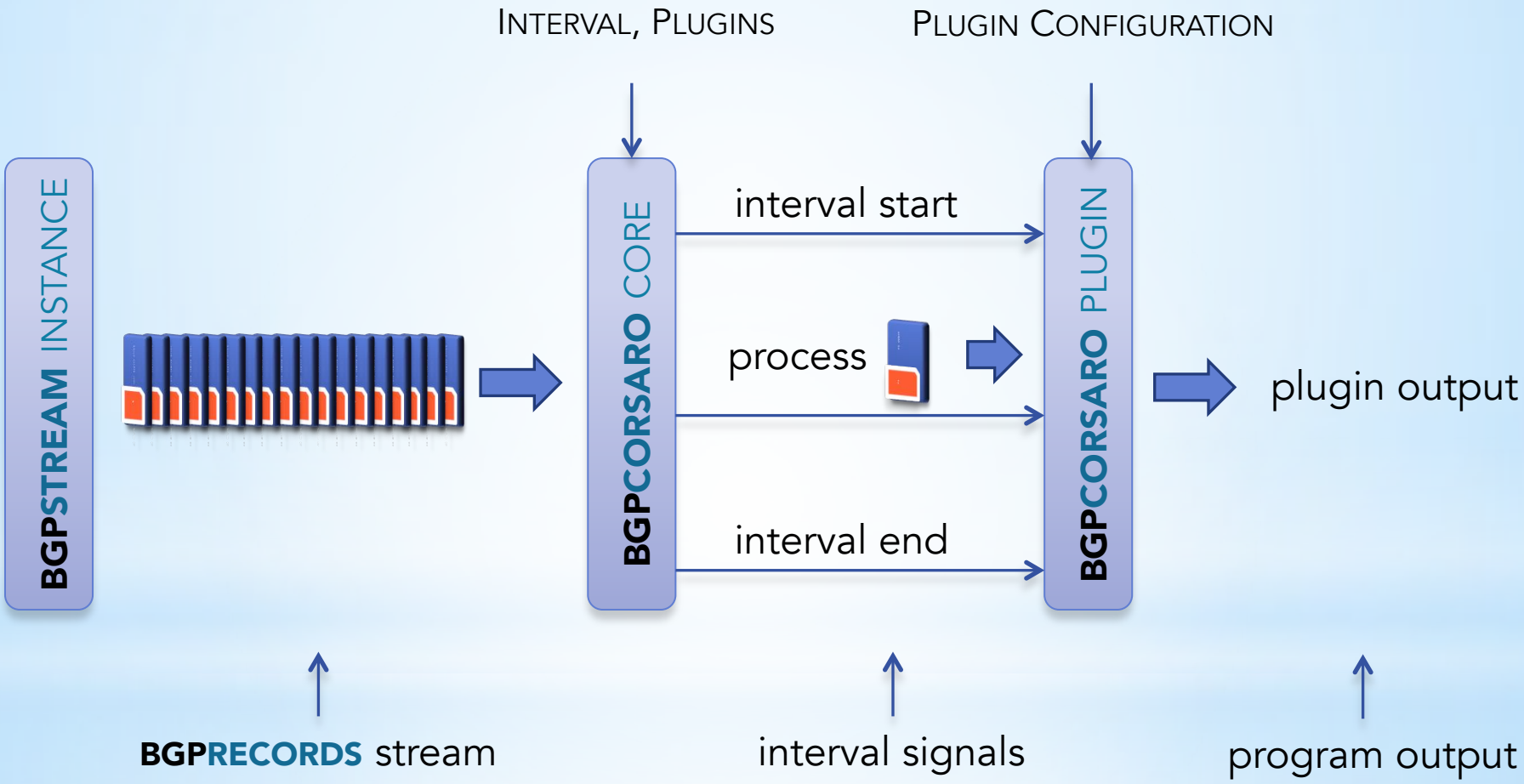
* C tool that transforms a stream of **BGPRECORDS** into a set of structures and metrics representative of specific time intervals

* interval driven

* modular architecture based on plugins



BGPCORSARO architecture



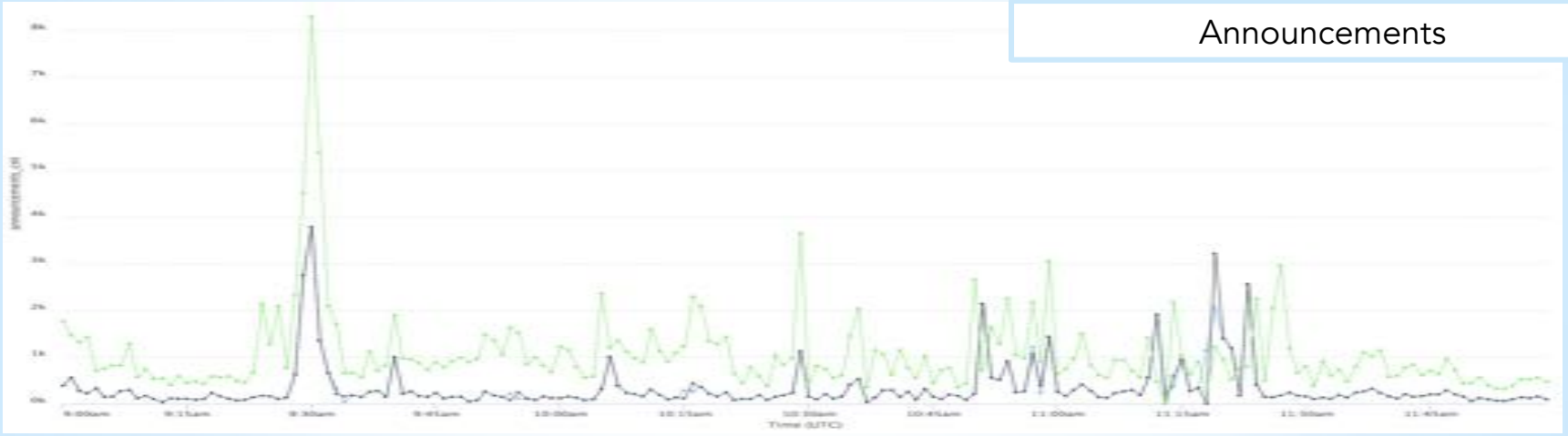
BGPCORSARO plugins

ROUTINGTABLES plugin:

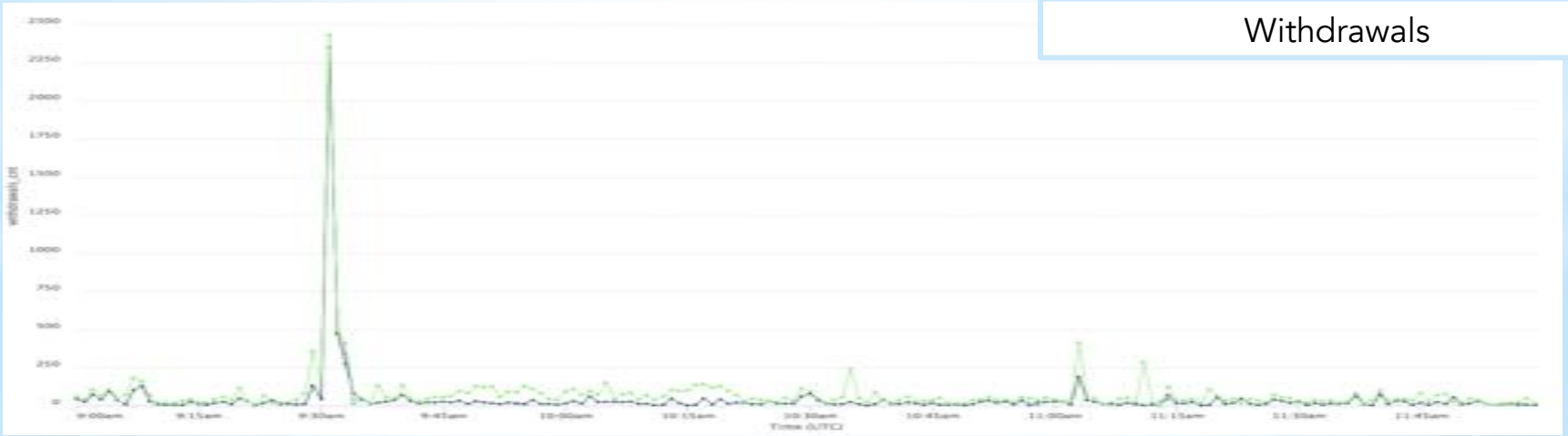
* it maintains the state and the routing table of each peer
  * BGP finite state machine per peer
  * RIBs and updates
  * recover from out of order and corrupted data
* outputs statistics every minute (of BGP time)

BGPCORSARO routingtables plugin

BGPCORSARO routingtables plugin

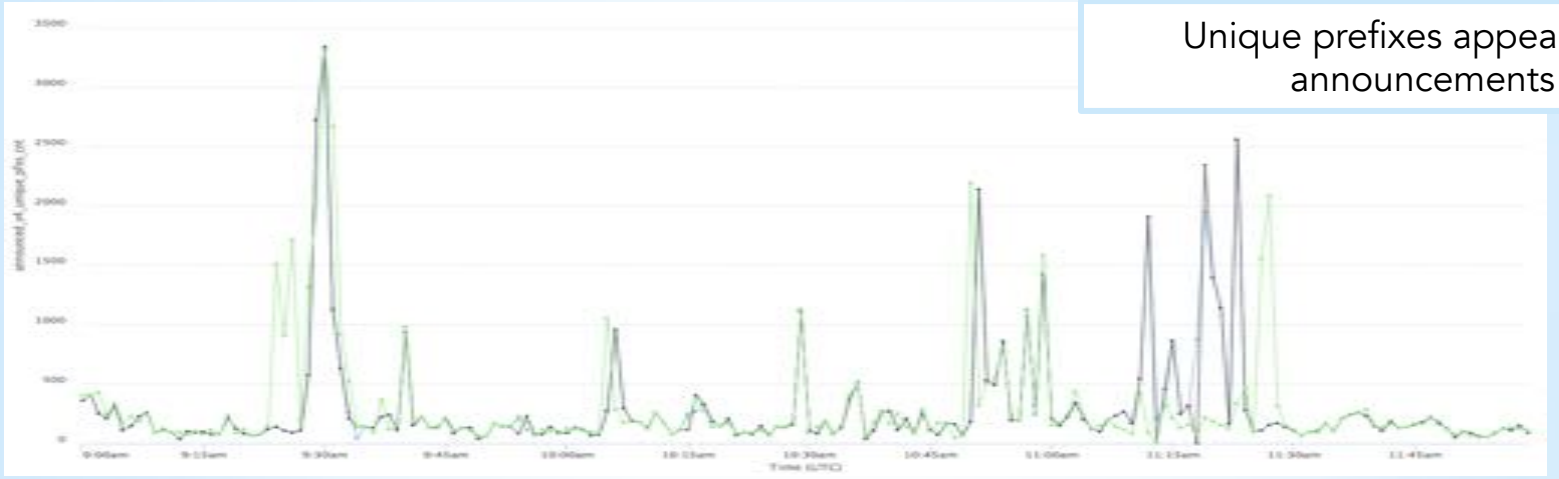


Announcements

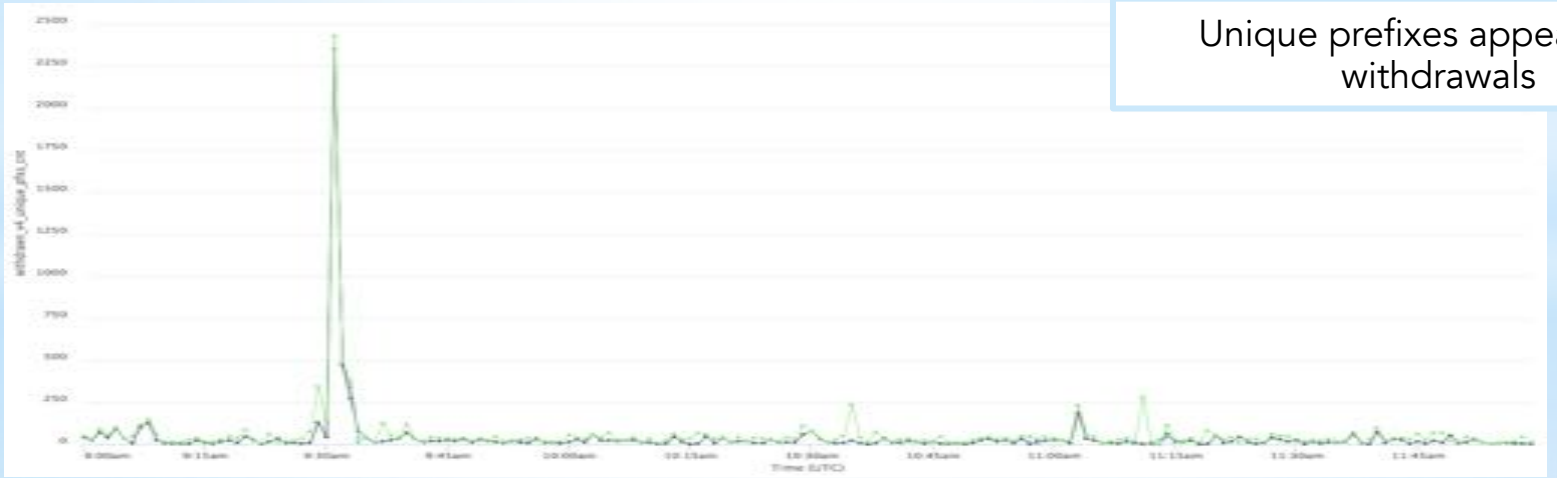


Withdrawals

BGPCORSARO routingtables plugin

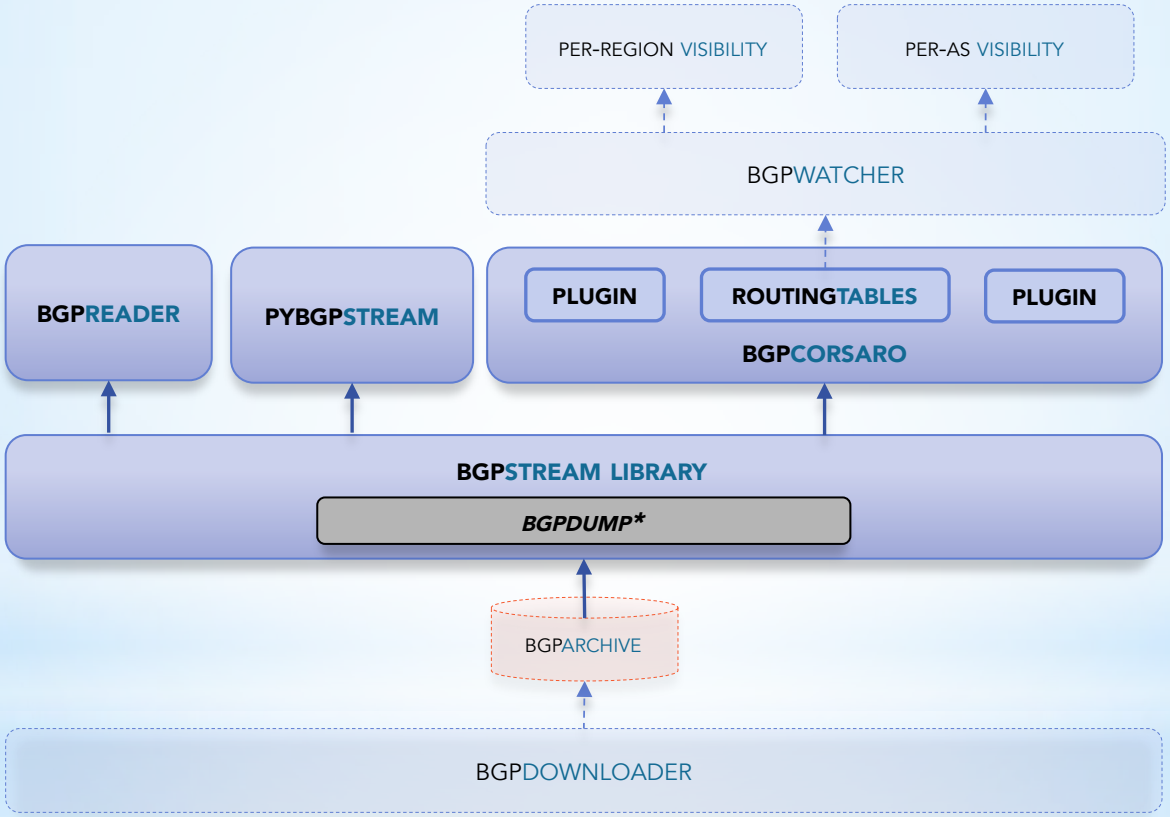


Unique prefixes appearing in announcements



Unique prefixes appearing in withdrawals

BGPSTREAM framework



QUESTIONS ?

THANKS

Chiara Orsini
chiara@caida.org