High Availability in BMP data-collection

Zhuoyao Lin

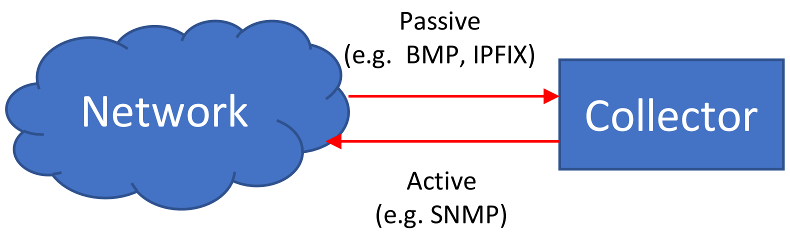
IETF 115 - GROW

I. Motivation

Network Telemetry

Gathering network insights via Network Telemetry is nowadays necessary.

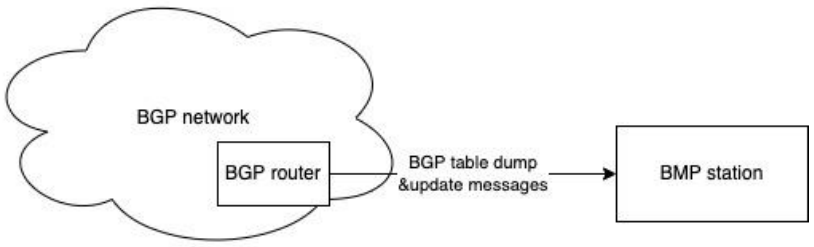
Having a **scalable** and **high-available** solution becomes then imperative.



BGP Monitoring Protocol (BMP)

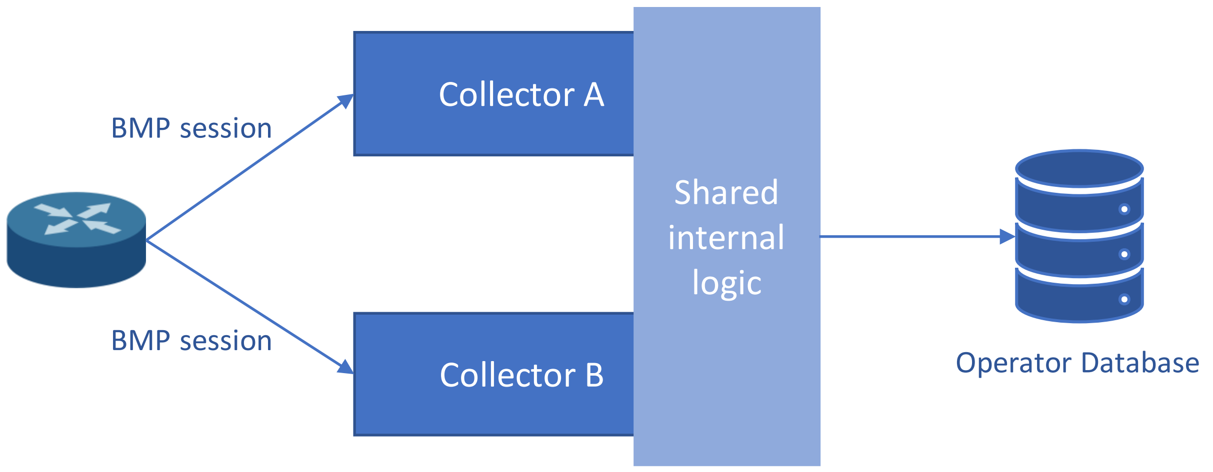
BMP provides access to the different RIBs, as well as a view of BGP updates a router is receiving.

This data is crucial for monitoring networks.



II. Problem Statement: Goal of having BMP high availability

- Every router exposes BMP twice to guarantee **high-availability.**
- The "shared internal logic" works across multiple collectors in multiple locations to guarantee **scalability** by dumping BMP only once in the database.

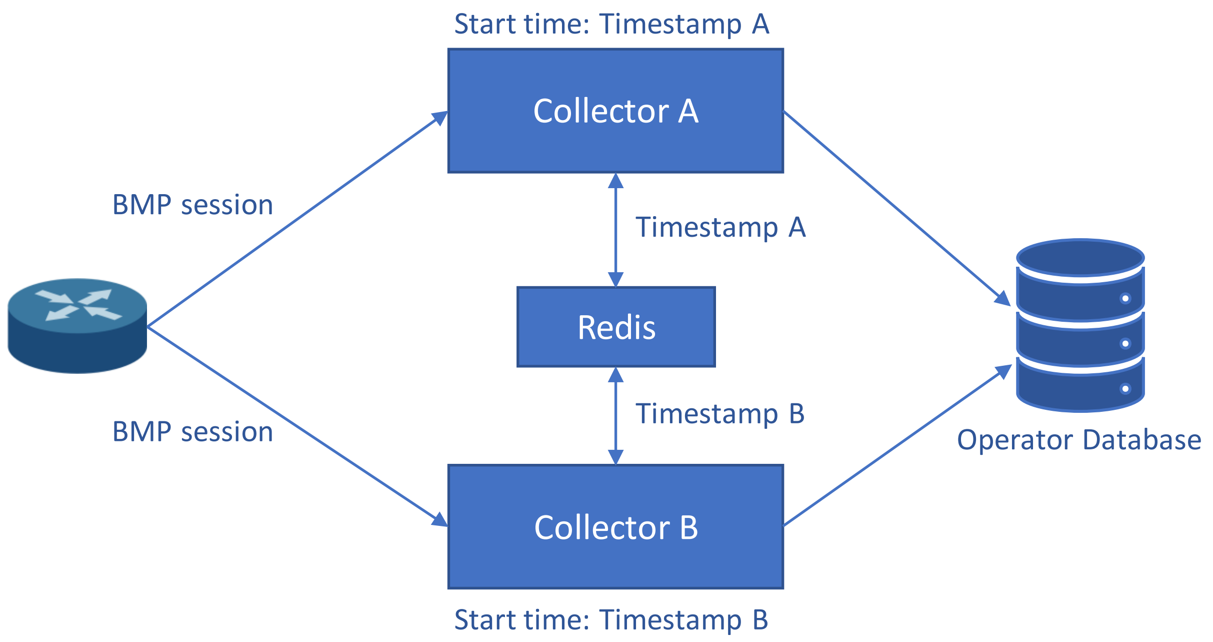


III. Design: Overview

1. Active/Standby feature:
Assign Active/Standby state to collector based on their timestamp

2. Redis for exchanging collector's timestamp:
Collector: (a) writes its timestamp to Redis every second
(b) gets collectors' timestamps from redis every second

3. Signals for maintenance:
Manually configure Active/Standby state in runtime



III. Design: Overview

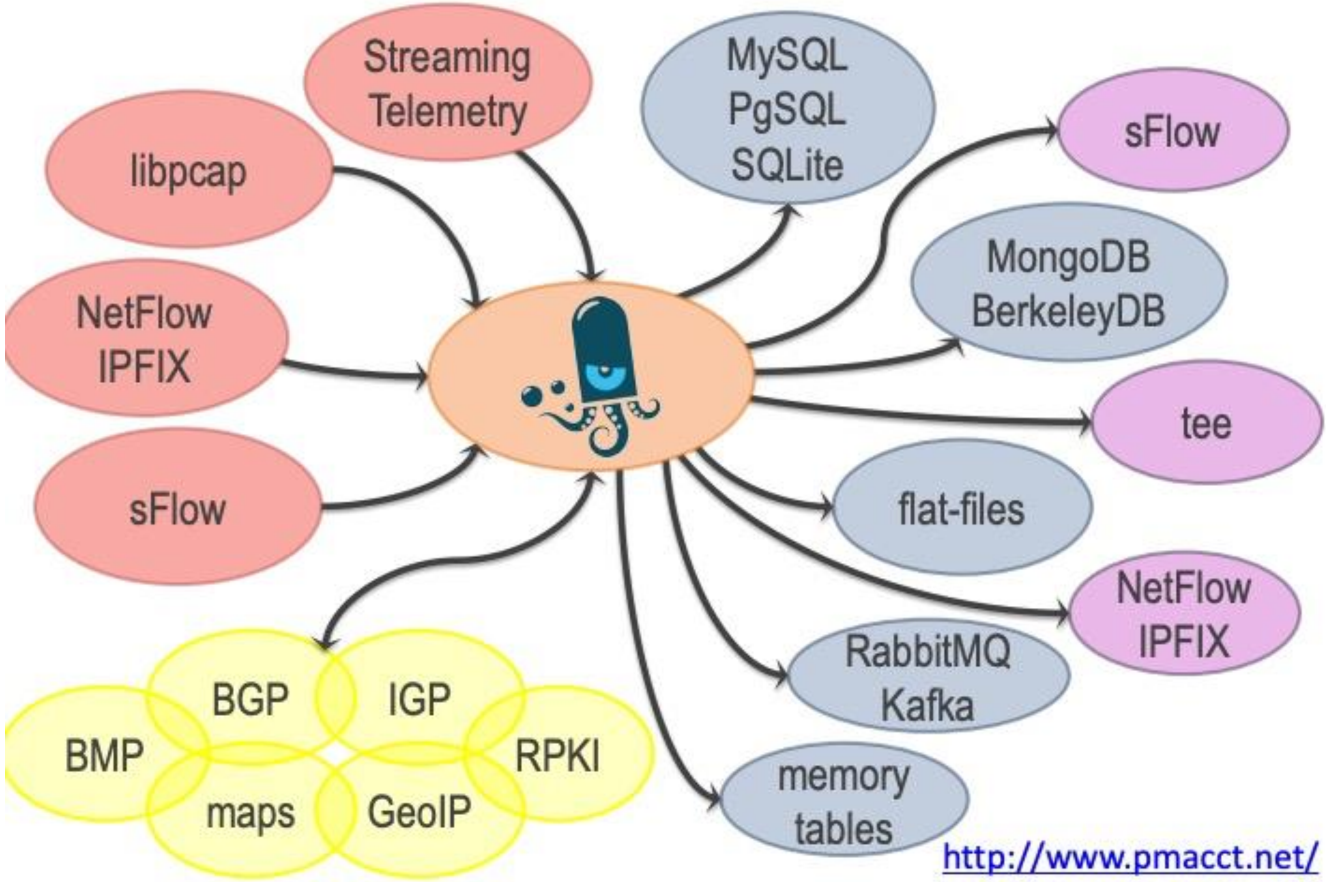
- Every second pmacct writes to Redis:
  Key: [cluster_name] +[cluster_id] +[core_process_name] (with 2s timeout)
  Value: [timestamp]
- Every second pmacct lists from Redis all the available collectors and set a dump_flag accordingly
- If dump_flag == True, then the collector writes the decided BMP messages into the Database (Only one collector has dump_flag == True)

	Collector A	Collector B
There is only collector A's timestamp	Active	----
Collector A has the smallest timestamp	Active	Standby
Collector A has NOT the smallest timestamp	Standby	Active
Redis is unavailable	Active	Active

III. Design: pmacct

pmacct is a small set of multi-purpose passive network monitoring tools.

To achieve our goal we need to implement the software logic in pmacct to make sure BMP data is cached twice but only forwarded once.



III. Design: Active/Standby State

Regular workflow

1. Write collector's timestamp key to Redis with 2 seconds timeout

2. Get all timestamp from Redis

3. Compare all timestamp and set dump_flag accordingly

4. Sleep 1 seconds and repeat

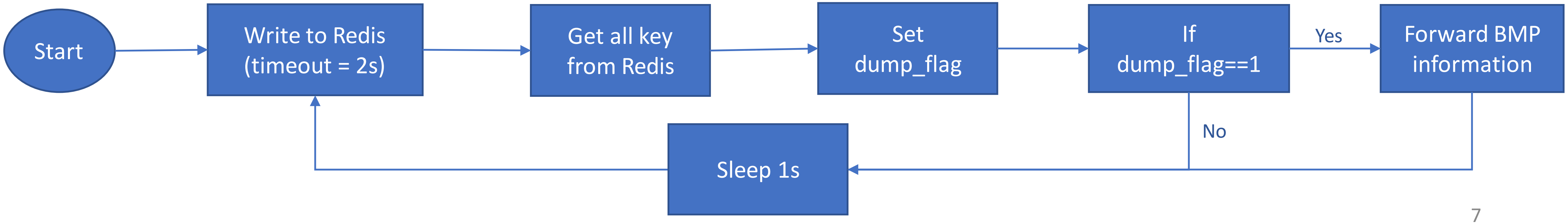
Maintenance mode

1. Catch signal 34

2. Set timestamp to current time

3. Write timestamp to Redis

As a result, the current collector will become standby



III. Design: Caching & Fail-over mechanisms

Caching mechanism

Standby collectors do not export metrics to the Database, but keep the received BMP event in a local cache for 2 seconds.

This allows to have “at least once” guarantee.

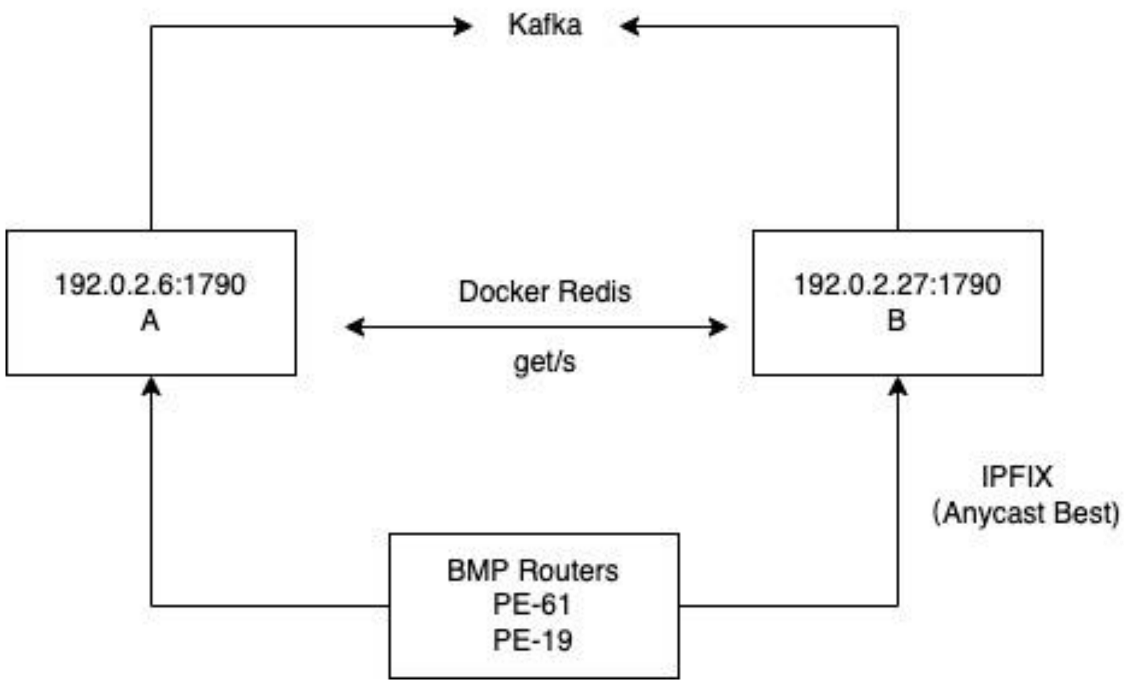
Fail-over mechanism

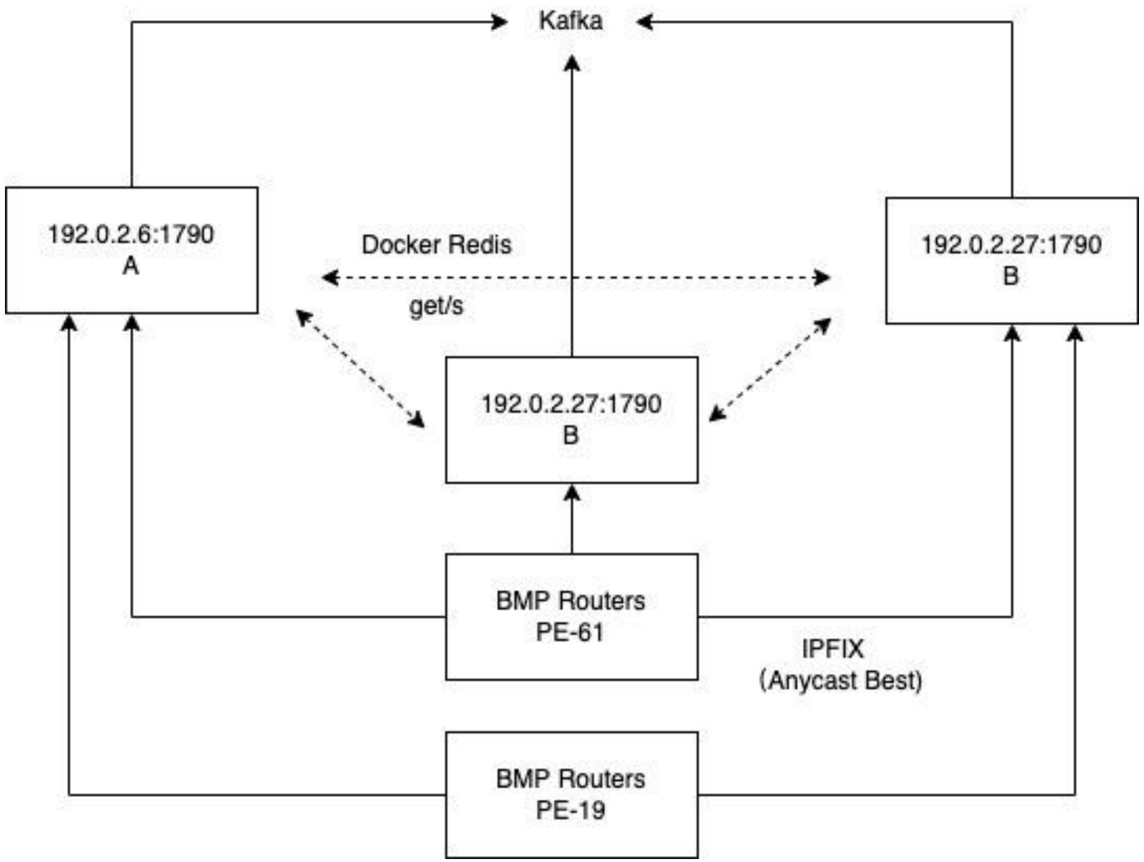
Assuming there are two collectors A (active), B (standby):

1. Collector A crashes (stops writing to Redis)
2. Redis will timeout A’s key in (0, 2] seconds
3. Collector B is now the collector with the lowest timestamp and so becomes active
4. Collector B sends the local cache to the DB
5. Collector B sends all BMP traffic received

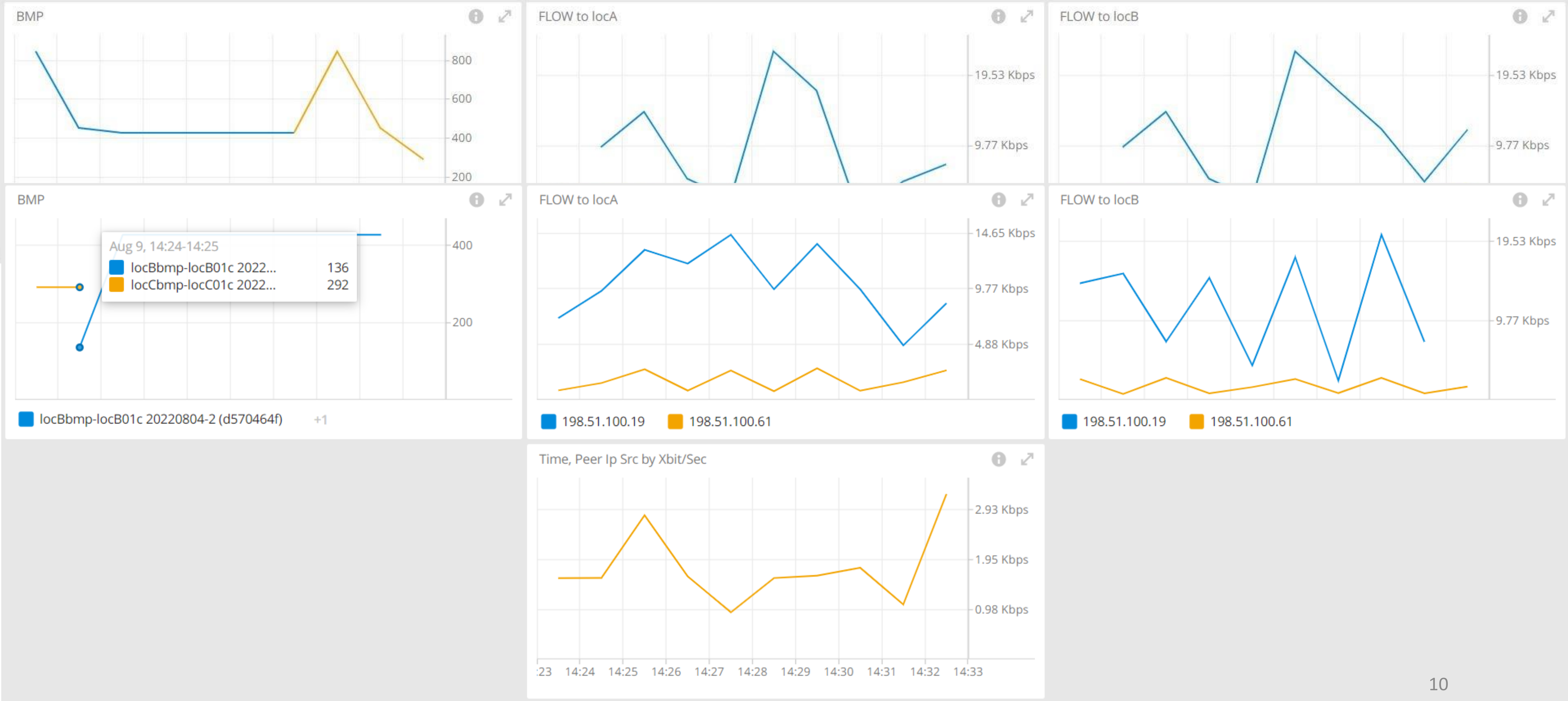
IV. Test Results:
Two & Three Daemon Set-up

Lab Set-up Diagram





IV. Test Results: Two & Three Daemon Set-up



V. Summary

- **Load Balance:** As the number of router increasing, more collectors are introduced to do data load balance. This bring horizontal **scalability**.
- **Data Duplication:** BMP data is exposed twice (or more) to guarantee **high availability**. By itself, this would bring data duplication.
- **Reduce Duplication:** this project designs a system to make sure the BMP data is cached twice (or more) but only forwarded once, to help in **scalability**.

Thanks

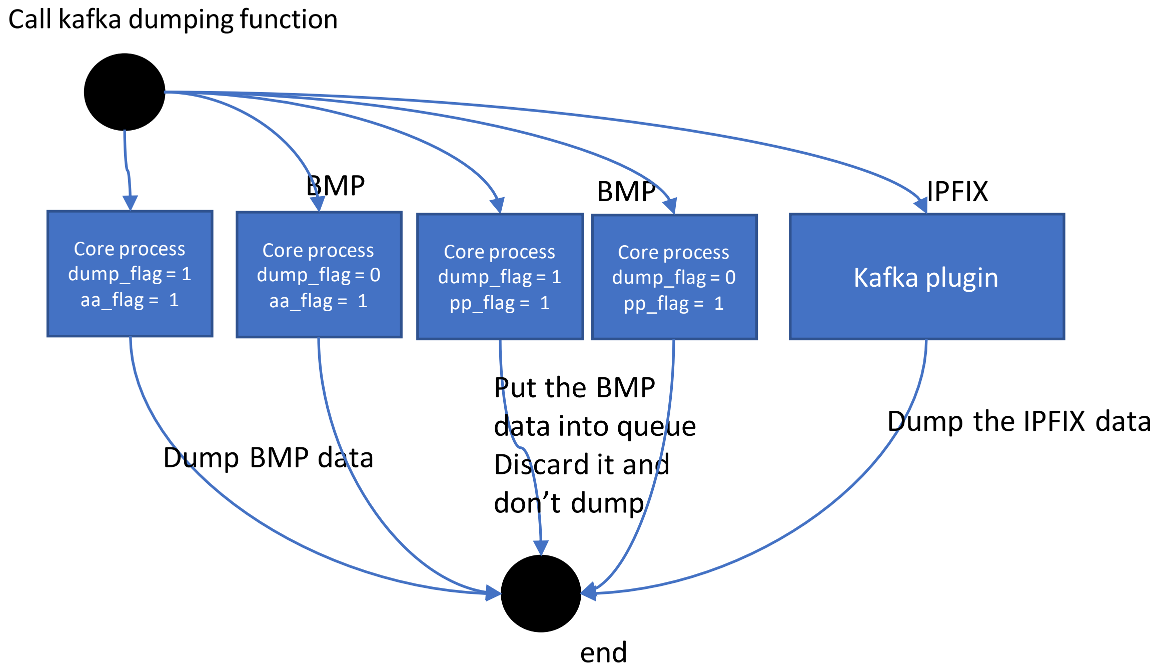
Links:

- Thesis:
- pmacct: https://github.com/pmacct/pmacct

Backup

III. Design: How Active/Standby Affects Dumping Process

Dumping Process



Data Queue