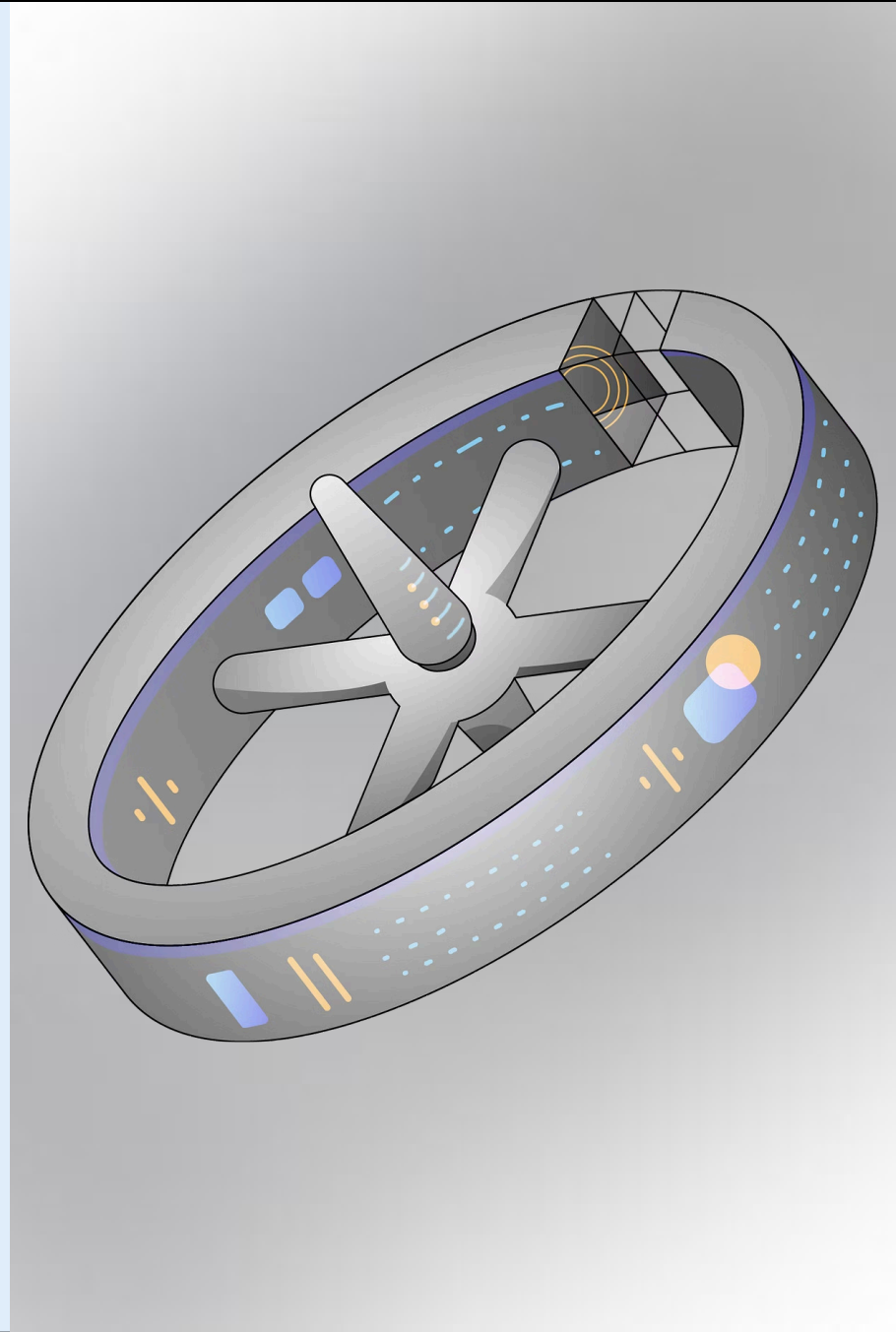
Monitoring 2,800 BGP Sessions using BMP protocol

Pavel Odintsov, Denys Fedoryshchenko

FastNetMon Inc



About myself

I’m a software engineer passionate about network security and the founder of FastNetMon Inc, a company focused on DDoS detection and routing security products for Telecoms

LinkedIn

linkedin.com/in/podintsov

Email

pavel@fastnetmon.com

Problem

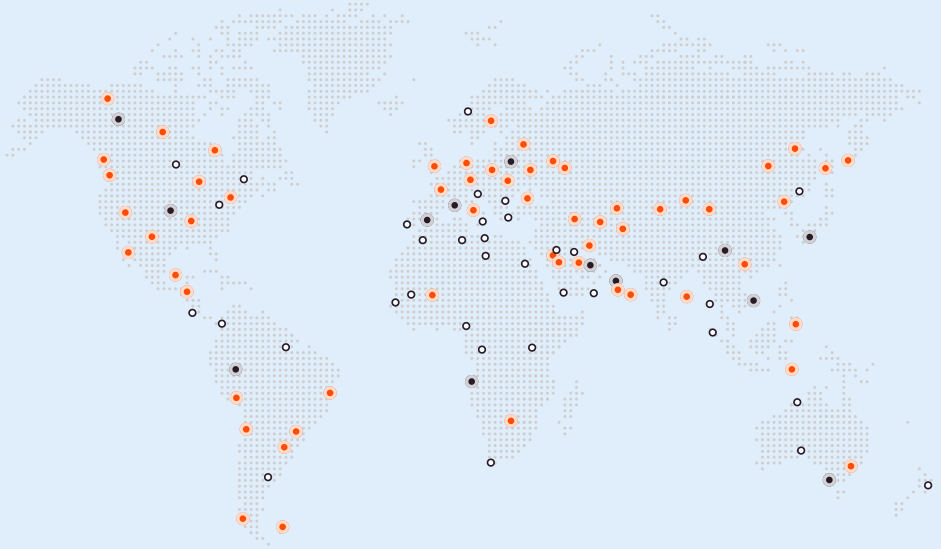
Rapid growth of network scale and number of customers made manual monitoring impossible

Goal

Reactive BGP monitoring with route level precision

The network: Gcore.com

Global Cloud AI Compute and Security Provider



210+

PoPs Worldwide

200+

Tbps Capacity

2,800

BGP Sessions

194 M

Routes

4

Routing Platforms

Nokia · Arista · Cisco · Juniper

SNMP for BGP Monitoring

Standard MIB Fields

- Peer information — local and remote IP addresses and ASNs
- BGP FSM session state via `bgpPeerState`
- Session timers: `bgpPeerFsmEstablishedTime`
- Message counters: `bgpPeerInUpdates`, `bgpPeerOutUpdates`, `bgpPeerInTotalMessages`

Vendor-Specific Extensions

- Total accepted prefixes
- Total advertised prefixes

Verdict

Very limited even for generic BGP session health checks. Lacks granularity to reliably detect BGP issues

gNMI for BGP Monitoring

gNMI includes everything SNMP offers plus detailed, per-session and per-AFI / SAFI visibility with event-driven updates instead of polling every X seconds. Solid support for all vendors: Juniper, Cisco, Nokia and Arista

Prefixes Received

`/./neighbors/neighbor/state/prefixes/received`

Prefixes Accepted

`/state/neighbors/neighbor/state/prefixes/accepted`

Prefixes Installed

`/state/neighbors/neighbor/state/prefixes/installed`

Prefixes Advertised

`/state/neighbors/neighbor/state/prefixes/sent`

gNMI is a reliable protocol for high-level BGP session health checks with per-AFI / SAFI visibility and event-based updates but it still lacks route-level granularity needed for reliable BGP monitoring

Vendor-Specific Options: A Dead End

Fragmented Data

Each vendor exposes unique OIDs, gNMI paths, and data models, preventing a unified view of BGP state.

Fragile Tooling

Schema changes across router software versions frequently break custom monitoring scripts and dashboards

Escalating Maintenance

The operational burden of supporting multiple distinct monitoring stacks grows exponentially with fleet diversity

No Single Pane

Without standardization, achieving a comprehensive, "single pane of glass" view across the entire BGP network is impossible

Full BGP Table as a Monitoring Source

✅ What It Provides

Complete, detailed information about every route in the RIB

❌ What It Misses

No per-session visibility. No notifications when a BGP peer transitions state. Cannot distinguish which peer originated a route change

⚠️ Operational Challenges

Processing a full BGP table at scale requires significant compute, memory and careful tooling selection.

GoBGP Full Table Benchmarks

GoBGP, an open-source BGP implementation written in Go, offers a reliable solution for BGP full table monitoring

GoBGP Neighbor Status

```
gobgp neighbor
Peer         AS     Up/Down     State       | #Received  Accepted
10.0.0.1      65001  1d 02:14:56 Establ      |  1040241    1040166
```

Test Platform Specifications

CPU: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
Memory: 64GB

GoBGP Full Table Benchmarks: Text-based Dump

```
time gobgp global rib -a ipv4 > /dev/null
real 1m13.399s
```

```
ls -lah full_table_bgp.txt
-rw-rw-r-- 1 root root 2.0G Apr 25 13:06 full_table_bgp.txt
```

Example data:

```
   0   1.0.64.0/18         10.0.0.1       4637 7670 1d 02:24:05 [{Origin: i} {Med: 100} {LocalPref: 100} {Aggregate: {AS: 18144,
Address: 10.0.0.2}} {Communities: 65001:64515, 65001:64615} {Extcomms: [valid]}]
```

GoBGP Full Table Benchmarks: JSON-based Dump

```
time gobgp nei 10.0.0.1 adj-in -j > full_table_bgp.json
real    1m15.196s
```

```
ls -alh full_table_bgp.json
-rw-rw-r-- 1 root root 543M Apr 25 13:08 full_table_bgp.json
```


```
{
  "1.0.0.0/24": [{
    "Family": 0,
    "nlri": {"prefix": "1.0.0.1"},
    "age": 1777027326,
    "best": false,
    "attrs": [
      {"type": 1, "value": 0},
      {"type": 2, "as_paths": [{"segment_type": 2, "num": 1, "asns": [13335]}]},
      {"type": 3, "nexthop": "10.0.0.1"},
      {"type": 4, "metric": 100},
      {"type": 5, "value": 100},
      {"type": 7, "as": 13335, "address": "10.0.0.1"},
      {"type": 8, "communities": [123456789, 23456789]},
      {"type": 16, "value": [{"type": 67, "subtype": 0, "value": 0}]}
    ],
    "stale": false,
    "peer-id": "10.0.0.1",
    "peer-address": "10.0.0.1"
  }]
}
```

GoBGP Full Table Benchmarks: gRPC / Protobuf

Using GoBGP to process Full BGP Table via gRPC client with Protobuf-encoded data. Full test code:
https://github.com/FastNetMon/gobgp-full-table-benchmark

```
./gobgp_client_benchmark --peer 10.0.0.1 --binary false --only-binary false --nlri-binary false --attr-binary false
target=127.0.0.1:50051 table=adj-in peer=10.0.0.1 iterations=3 only_binary=0 nlri_binary=0 attr_binary=0 decode=1 best_only=0
```

```
run=1 dests=1040391 paths=1040391 decoded=1040391 ttfb_ms=11804.8 in_recv_ms=27366.9 in_proc_ms=84.5  total_ms=27495.3
run=2 dests=1040401 paths=1040401 decoded=1040401 ttfb_ms=10207.3 in_recv_ms=25556.4 in_proc_ms=81.8  total_ms=25679.9
run=3 dests=1040401 paths=1040401 decoded=1040401 ttfb_ms=10571.7 in_recv_ms=25654.4 in_proc_ms=84.4  total_ms=25781.9
```

~1,040,000

Routes Processed

25.5 – 27.5s

Total Time

Significantly faster than text or JSON dumps — but still requires ~25s per full table pull at 1M routes.

GoBGP Full Table Benchmarks: gRPC with Binary NLRI

```
./gobgp_client_benchmark --peer 10.0.0.1  --binary false --only-binary true --nlri-binary true --attr-binary true
```

```
target=127.0.0.1:50051 table=adj-in peer=87.245.224.162 iterations=3 only_binary=1 nlri_binary=1 attr_binary=1 decode=1 best_only=0

run=1 dests=1040396 paths=1040396 decoded=1040396 nlri_bytes=4149958 attr_bytes=78159655 ttfb_ms=7717.9 in_recv_ms=14763.2
in_proc_ms=337.0 total_ms=15143.4
run=2 dests=1040399 paths=1040399 decoded=1040399 nlri_bytes=4149970 attr_bytes=78161717 ttfb_ms=8385.0 in_recv_ms=15578.7
in_proc_ms=327.8 total_ms=15947.1
run=3 dests=1040388 paths=1040388 decoded=1040388 nlri_bytes=4149926 attr_bytes=78160765 ttfb_ms=8388.0 in_recv_ms=14634.1
in_proc_ms=298.0 total_ms=14969.0
```

~1,040,000

Routes Processed

15 – 15.5s

Total Time

⚠ While faster than previous methods, processing a full BGP table of ~1M routes still takes 15 seconds. It also requires implementing complex logic to parse BGP binary data

Rotonda: fast alternative

Rotonda is an open-source BGP data pipeline project by NLnet Labs, implemented in Rust. It is designed to process BGP and BMP data at scale.

- Written in Rust - modern, memory-safe, high-performance
- Native BGP, BMP and MRT protocol support
- REST API for route queries and JSON snapshots

GitHub: **https://github.com/NLnetLabs/rotonda**

Rotonda Full Table Benchmarks: JSON via API

Rotonda provides a REST API to query its BGP routing information base (RIB) and dump the full table in JSON format.

```
time curl -s http://[::1]:18082/api/v1/ribs/ipv4unicast/routes -o routes.json
real    0m9.629s
```

```
ls -lah routes.json
-rw-rw-r-- 1 root root 542M Apr 25 14:34 routes.json
```

~1,040,000

Routes Processed

9.6s

Total Time

Dumping the full BGP table from Rotonda via its REST API is significantly faster than GoBGP's native JSON dump

GoBGP vs. Rotonda: Memory and Performance Comparison

Tool	Current Memory	Peak Memory	Time
GoBGP	850 MB	5.7 GB	15.5s
Rotonda upstream	1.3 GB	1.4 GB	10s
Rotonda Netomics	1.3 GB	1.1 GB	8.5s

6.5x

Lower Peak Memory (Rotonda Netomics vs GoBGP)

45%

Faster than GoBGP

Full BGP monitoring Limitations

Path Visibility

Only best paths are exported by default. ADD-PATH can extend this to a limited number of additional paths, but full multi-path visibility is not guaranteed across all vendors.

Peer Event Notifications

No native visibility into routing events from peers - peer-up and peer-down notifications exist but filtered route events are opaque

Filtered Route Visibility

Routes dropped by inbound policy are reported as counters only

Still Needs SNMP / gNMI

It does not replace all telemetry. Session-level health metrics and hardware counters still require SNMP or gNMI as a complementary source

BMP: The Right Tool for BGP Monitoring

BGP Monitoring Protocol (RFC 7854)

What BMP Provides

Route Visibility

Exports routes as-is for each peer individually across all policy stages:

- Pre-policy
- Post-policy
- Local RIB

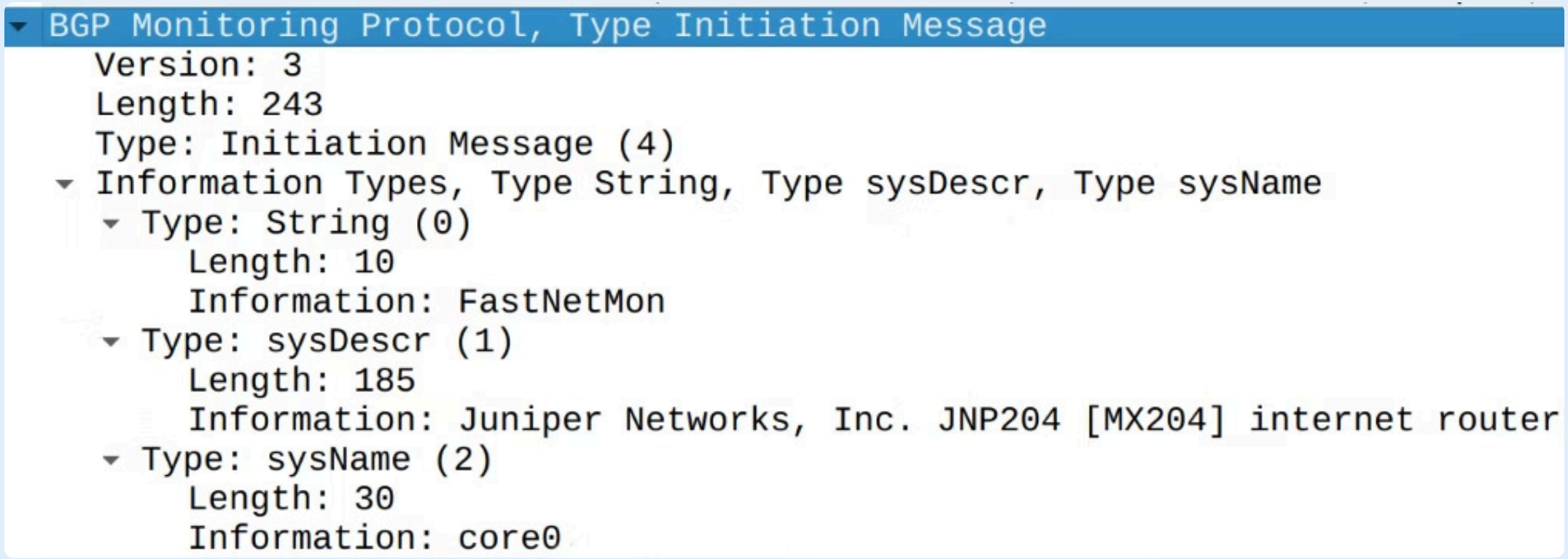
Vendor Support (tested)

Cisco · Juniper · Nokia · Arista · GoBGP · Bird · FRR

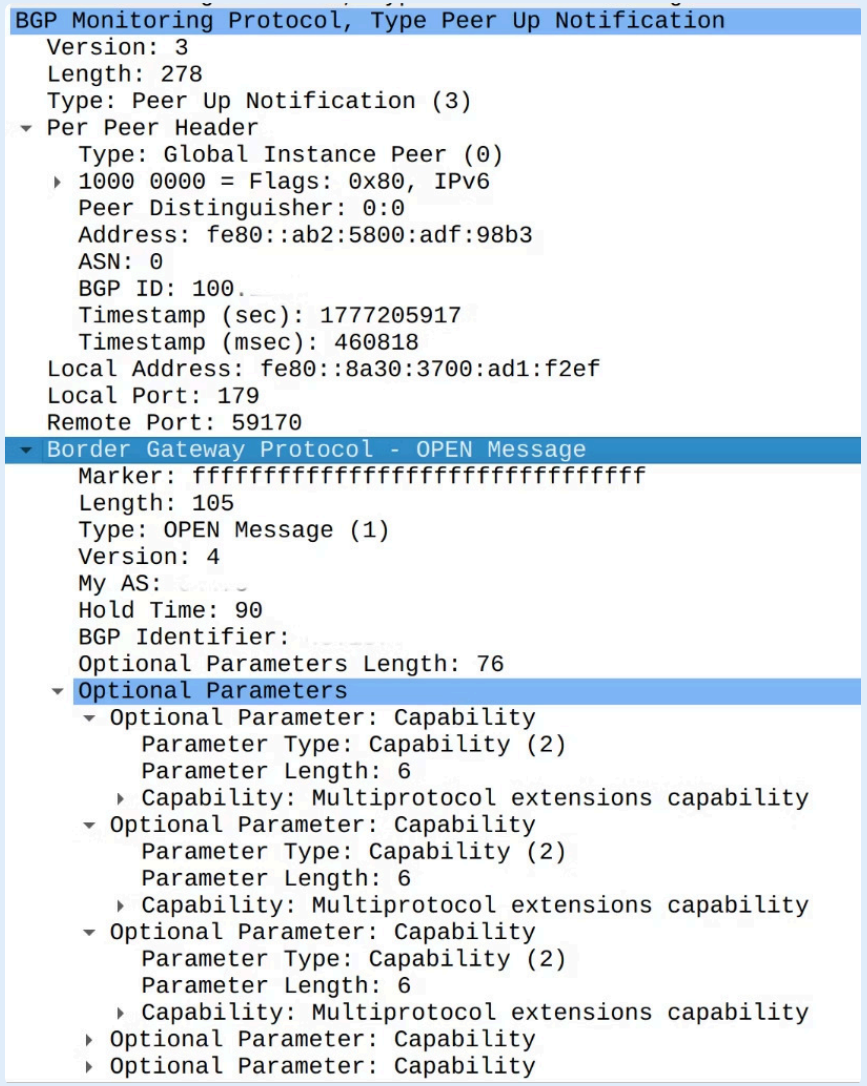
Per-Session Statistics, RFC 7854 Stats

- Prefixes rejected by inbound policy
- Duplicate prefix advertisements / withdraws
- Updates invalidated by CLUSTER_LIST / AS_PATH / ORIGINATOR_ID loops
- Routes in Adj-RIBs-In / Loc-RIB
- Per-AFI/SAFI Adj-RIB-In and Loc-RIB counts
- Treat-as-withdraw counters and duplicate updates

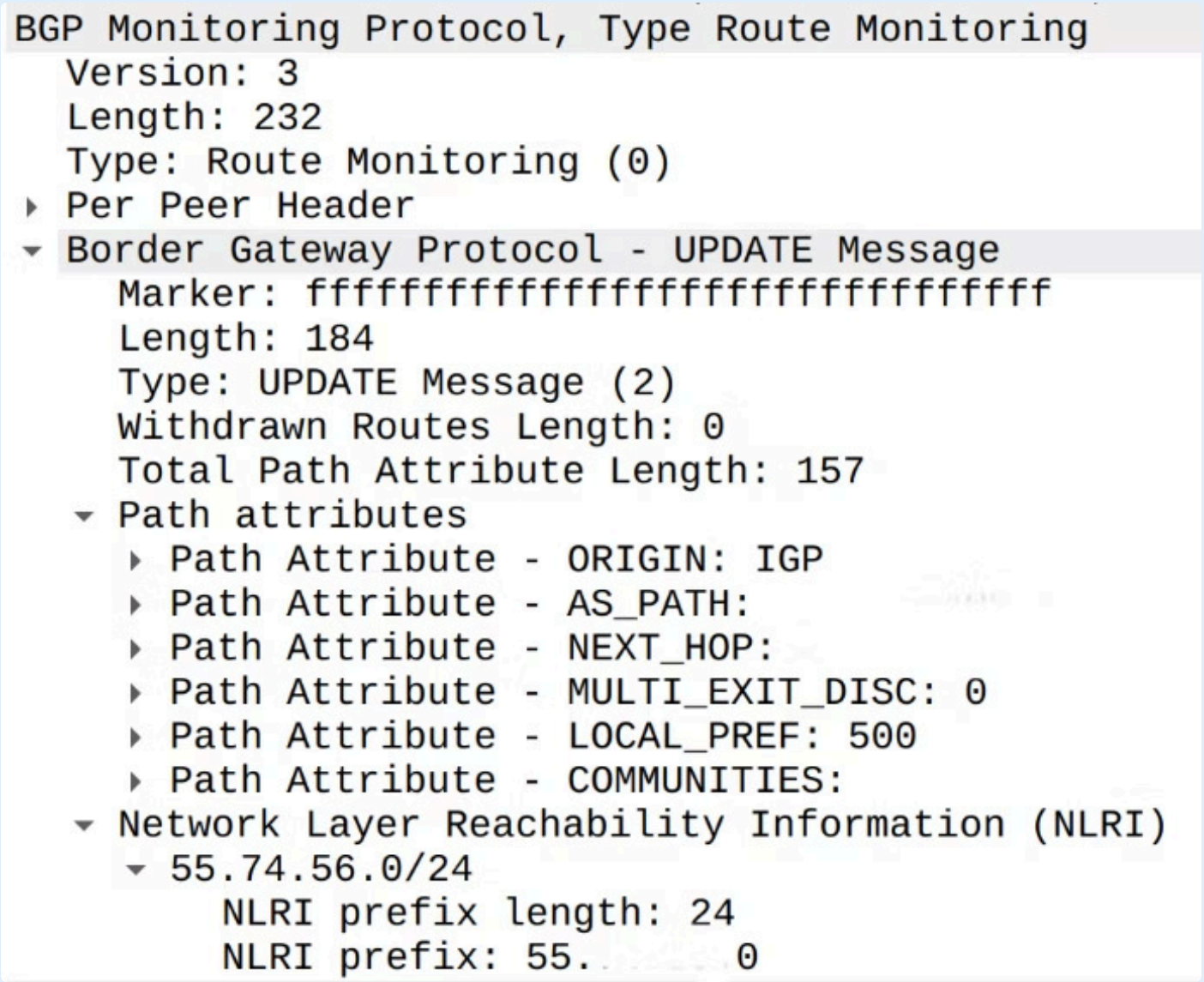
BMP Initiation Message



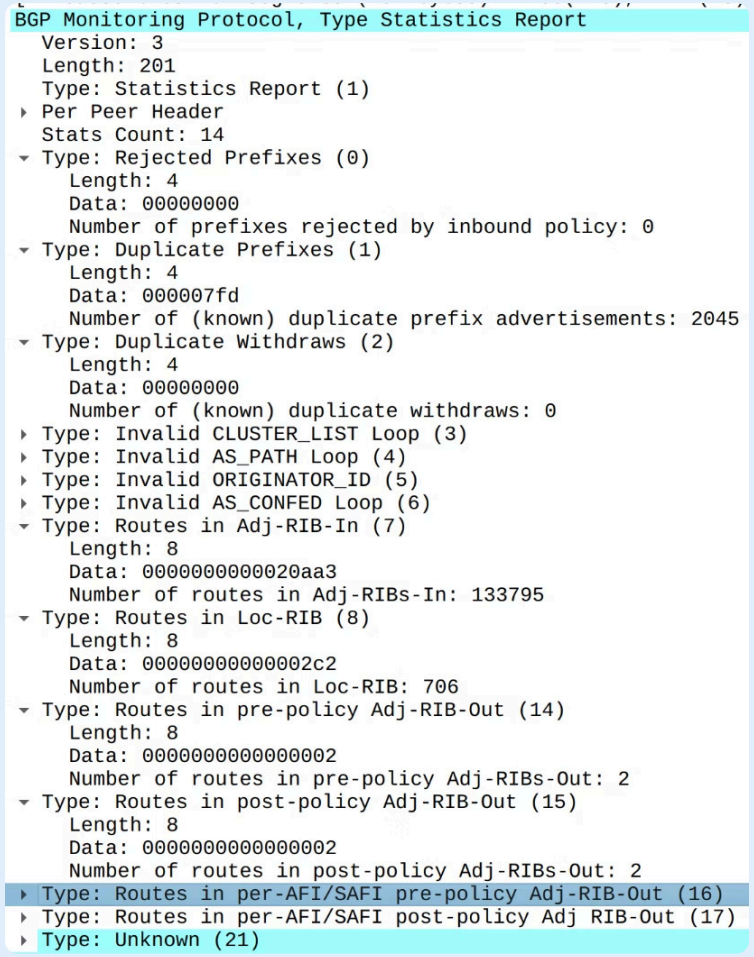
BMP BGP Open Message



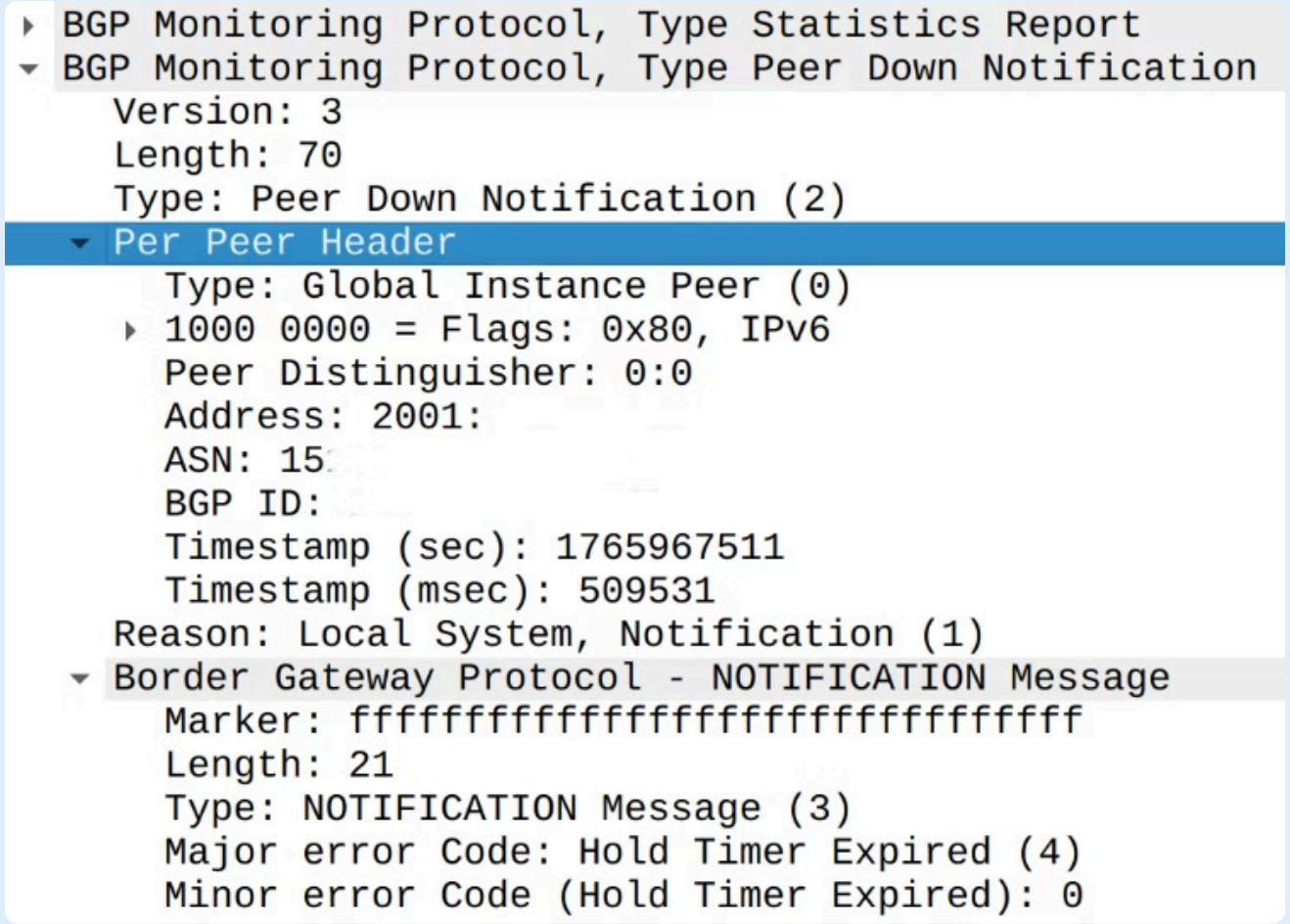
BMP BGP Update Message



BMP Statistics Report



BMP BGP Peer Down Notification



Nokia BMP Configuration Example

```
configure bmp
  station fastnetmon-bmp create
    family ipv4 ipv6 vpn-ipv4 label-ipv6
    stats-report-interval 900
    connection
      station-address xx.xx.xx.xx port 11020
    no shutdown
  no shutdown
```

Next, enable BMP monitoring for your BGP peers

```
configure router bgp
  group internal-peers
    monitor
      station fastnetmon-bmp
      route-monitoring pre-policy post-policy
      no shutdown
```

Juniper BMP Configuration Example

```
routing-options {
    bmp {
        station BMPServerFastNetMon {
            initiation-message "FastNetMon";
            local-address XXX;
            connection-mode active;
            monitor enable;
            route-monitoring {
                pre-policy;
                post-policy;
            }
            station-address xx.xx.xx.xx;
            station-port 11020;
            statistics-timeout 300;
        }
    }
}
```

Cisco XR BMP Configuration Example

```
bmp
 server 1
  host xx.xx.xx.xx port 11020
  description FastNetMon BMP Server
  update-source Loopback0
  initial-delay 60
  initial-refresh delay 60
  stats-reporting-period 1800
```

Then, enable BMP monitoring for specific BGP neighbors:

```
router bgp <ASN>
 neighbor <BGP_PEER_IP>
  bmp-activate server 1
```

Arista BMP Configuration Example

```
router bgp XXXXX
   bgp monitoring
   monitoring station OBMP_LAB
      update-source Loopback0
      connection address xx.xx.xx.xx
      connection mode active port 11020
```

Our BMP Daemon Requirements

Per-Session Routing Tables

Fast Query Performance

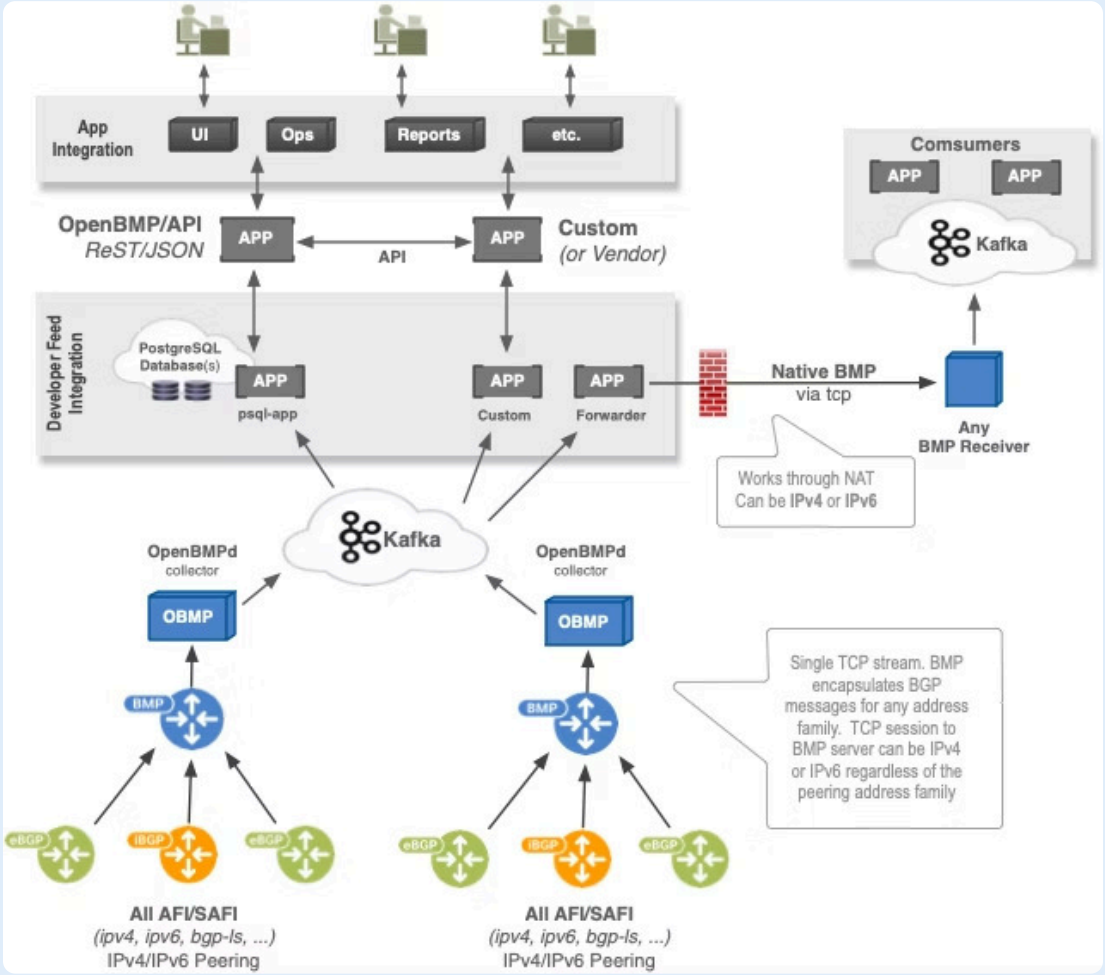
Exceptional Scalability

Fast Restarts

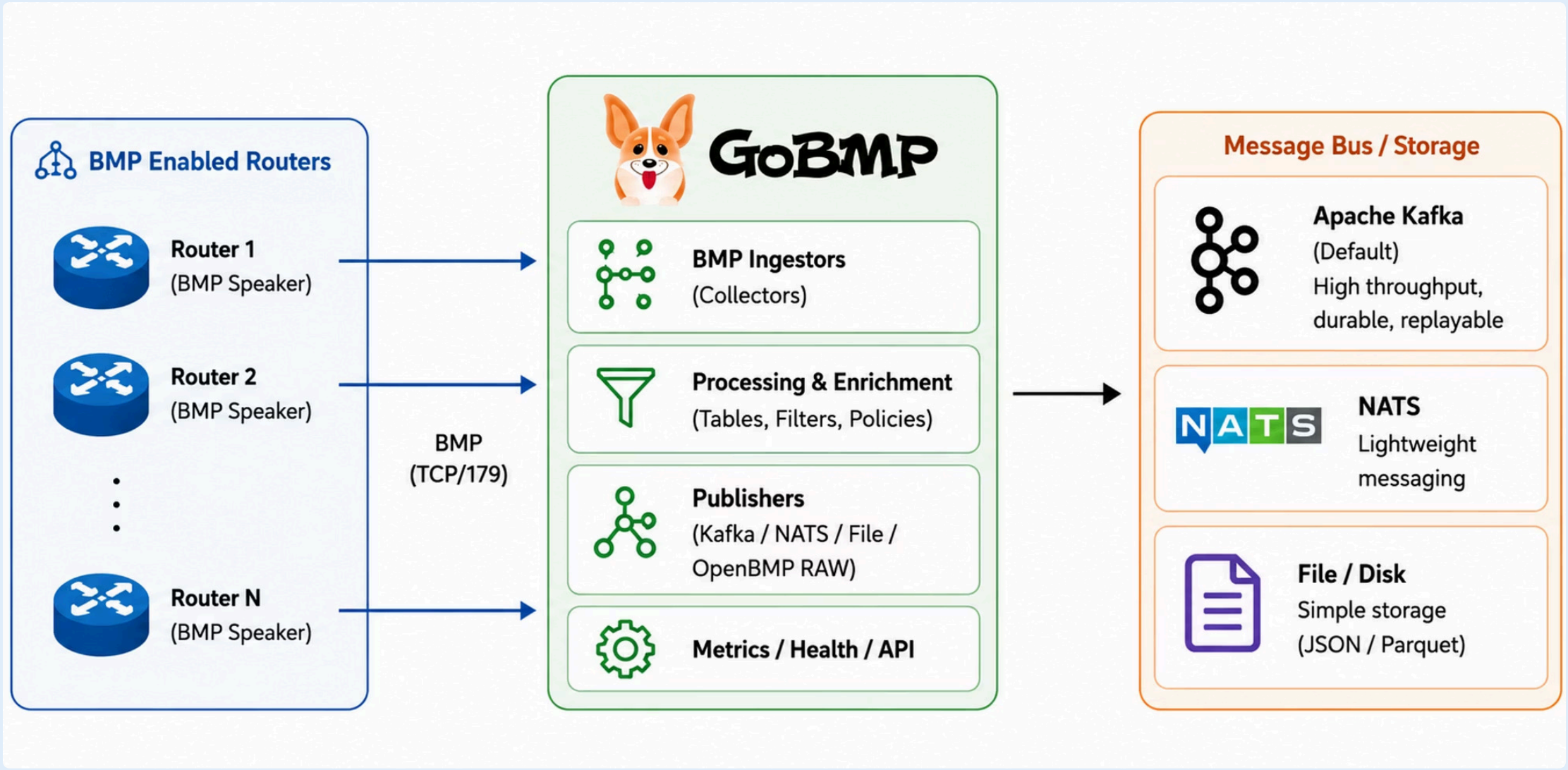
Open-Source BMP Daemons

- OpenBMP
- GoBMP
- pmacct
- Rotonda

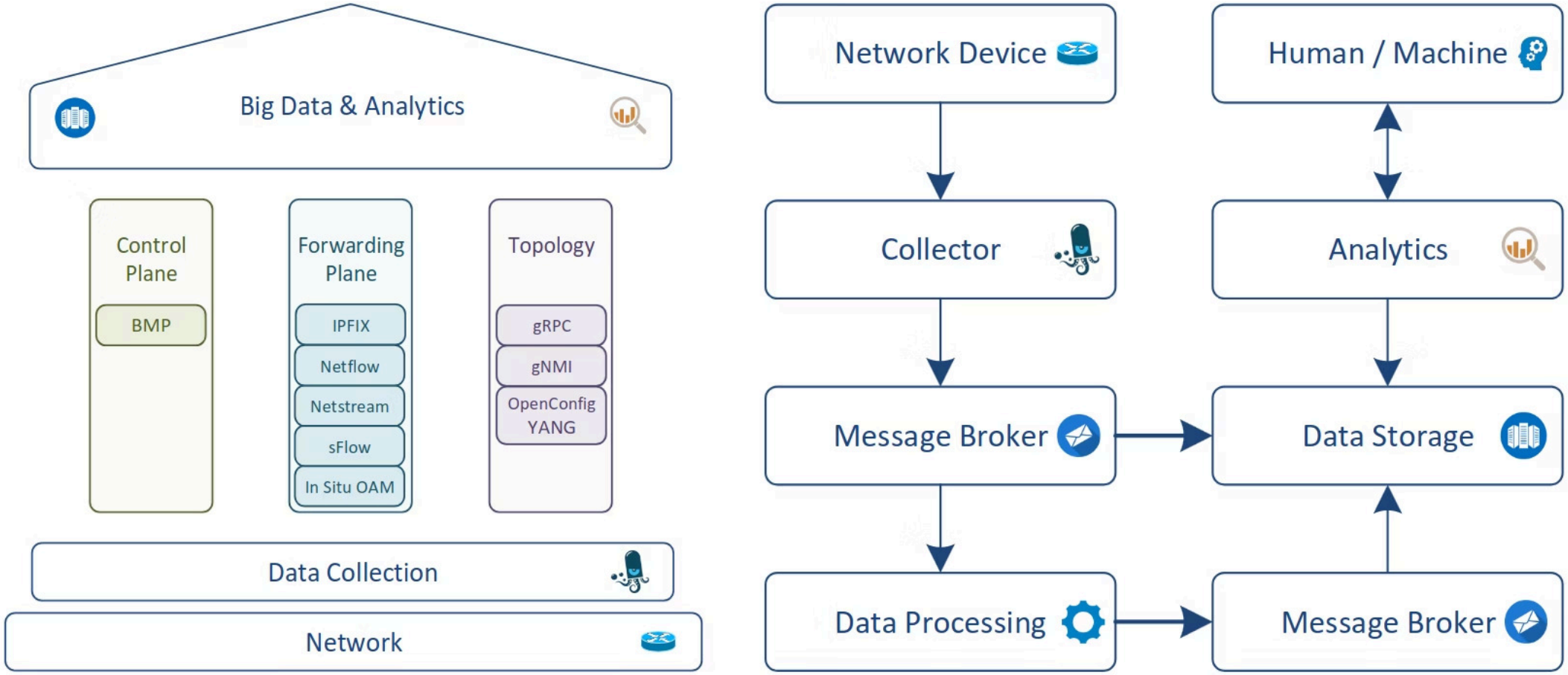
OpenBMP: Kafka, C++, stalled development



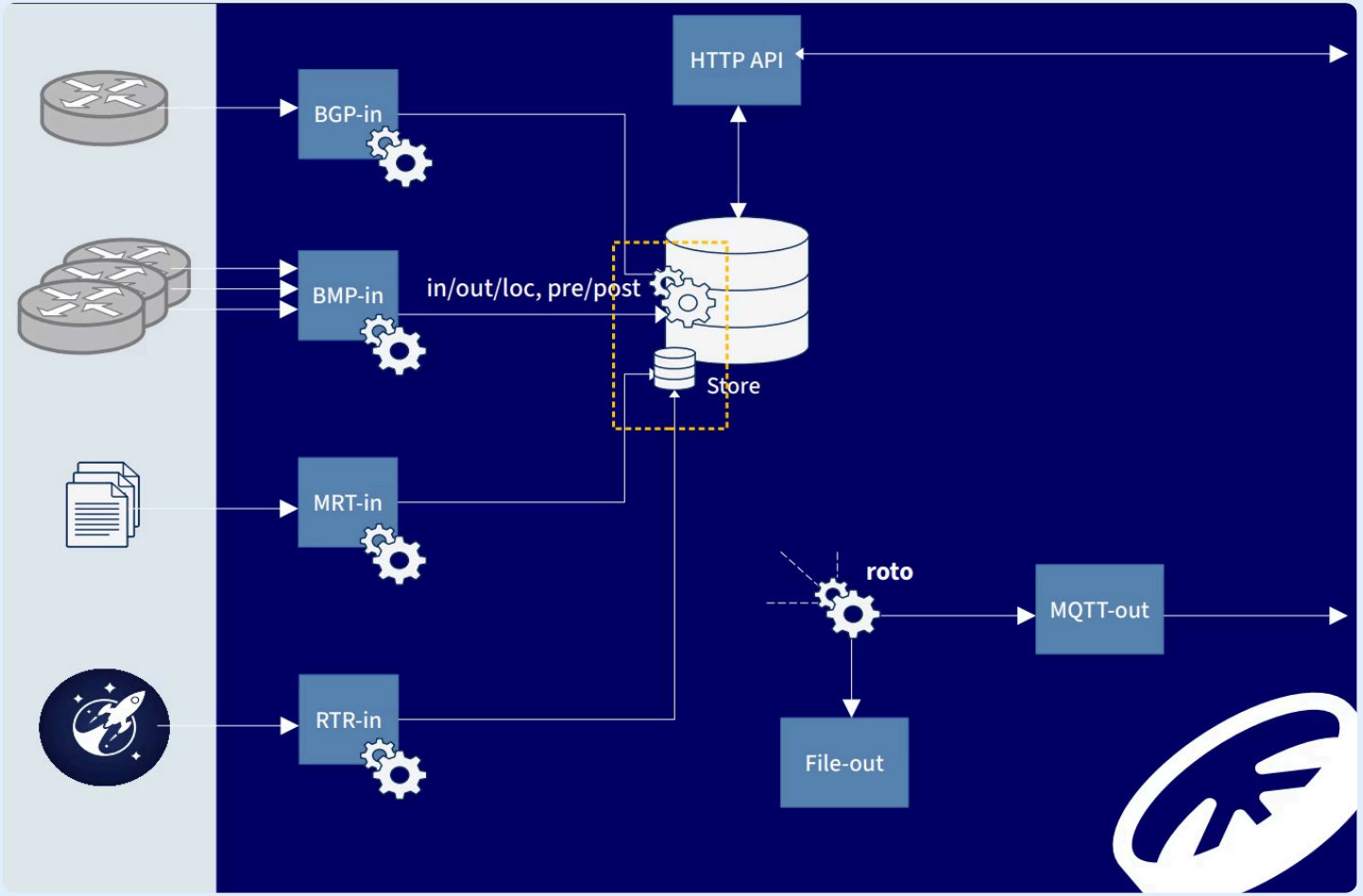
GoBMP design: Go, Kafka / NATS



Pmacct: Kafka and more



Rotonda as BMP collector



Production BMP Environment

CPU

AMD Ryzen 9 7950X3D
16-Core Processor

Memory

128 GB RAM

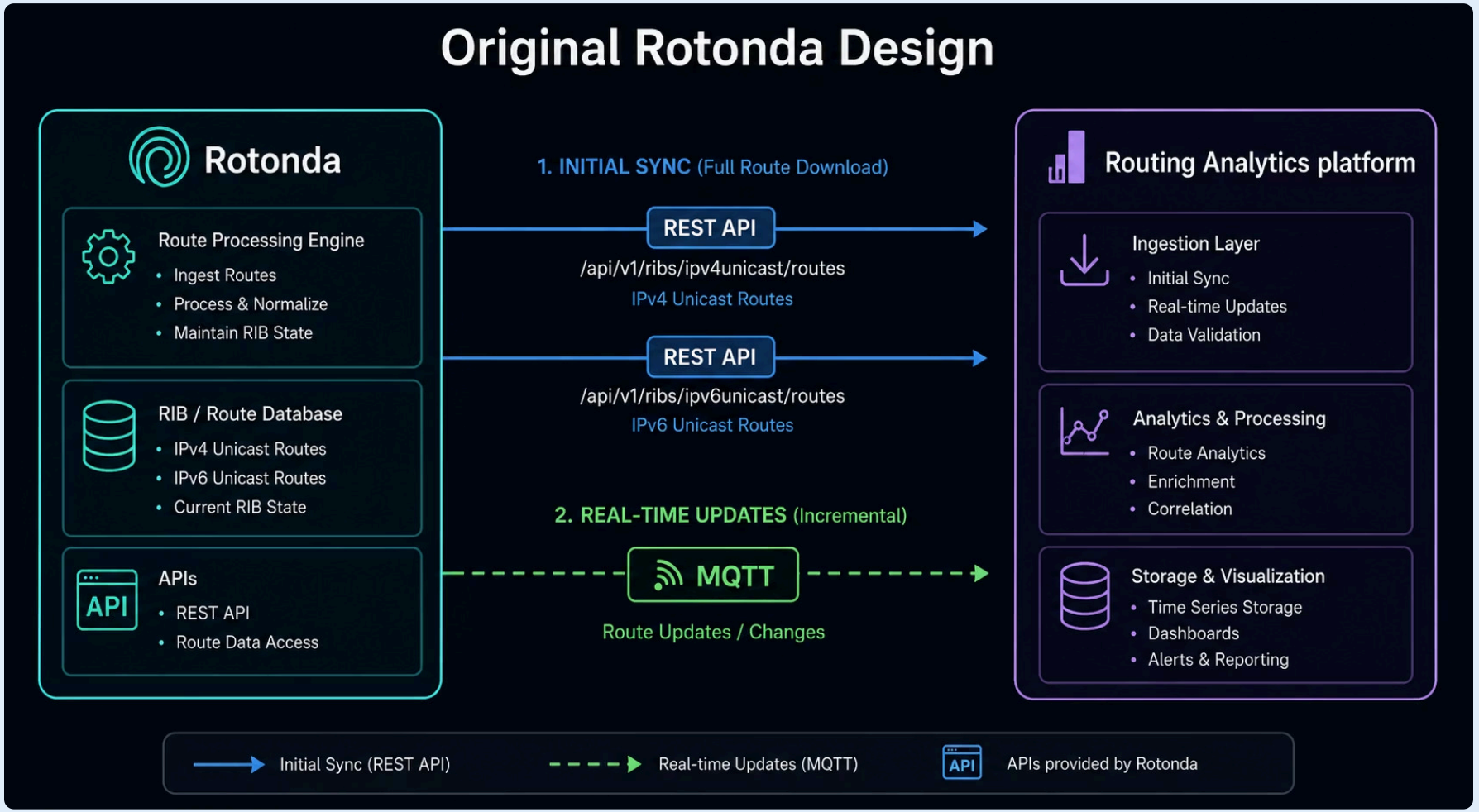
Platforms

Nokia · Arista
Cisco · Juniper

Rotonda design v1

- Initial sync via **/api/v1/ribs/ipv4unicast/routes**
- Initial sync via **/api/v1/ribs/ipv6unicast/routes**
- MQTT for updates

Rotonda design v1: diagram



MQTT is not easy in Rontonda

```
filter bmp_in(bmp_msg: BmpMsg, ingress_info: IngressInfo) {
    accept
}

filter rib_in_pre(route: Route, ingress_info: IngressInfo) {
    let aspath = route.fmt_aspath();
    let comms = route.fmt_communities();
    let lcomms = route.fmt_large_communities();

    let json = f"{{\
        \"feed\":\"[[ ansible_fqdn ]]\",\
        \"ipv4\":\"[[ ansible_default_ipv4.address ]]\",\
        \"ipv6\":\"[[ ansible_default_ipv6.address ]]\",\
        \"type\":\"rib\",\
        \"event\":\"announce\",\
        \"peer_asn\":\"{ingress_info.peer_asn().fmt()}\",\
        \"peer_addr\":\"{ingress_info.peer_address()}\",\
        \"prefix\":\"{route.fmt_prefix()}\",\
        \"as_path\":\"{aspath}\",\
        \"communities\":\"{comms}\",\
        \"large_communities\": \"{lcomms}\"\
       }}";

    // Create output entry
    let e = output.entry();

    // Attach timestamp + JSON payload
    e.timestamped_custom(json);

    // Flush entry to configured backend (MQTT only)
    output.write_entry();

    accept
}
```

MQTT in Rotonda is sensitive to packet loss

```
Jan 22 14:01:13 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:01:18 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:01:23 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:01:28 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:01:31 mex1 rotonda[80906]: mqtt: Connected to MQTT server at xxx.xxx.xxx.xxx:1883
Jan 22 14:01:31 mex1 rotonda[80906]: mqtt: MQTT connection error: Mqtt state: Io error: Custom { kind: ConnectionAborted,
  error: "connection closed by peer" }
Jan 22 14:01:33 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:01:38 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:01:43 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:01:48 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:01:53 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:01:58 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:02:03 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:02:08 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:02:13 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:02:18 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:02:23 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:02:28 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:02:31 mex1 rotonda[80906]: mqtt: Connected to MQTT server at xxx.xxx.xxx.xxx:1883
Jan 22 14:02:31 mex1 rotonda[80906]: mqtt: MQTT connection error: Mqtt state: Io error: Custom { kind: ConnectionAborted,
  error: "connection closed by peer" }
Jan 22 14:02:33 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
Jan 22 14:02:38 mex1 rotonda[80906]: mqtt: Publishing failed: MQTT Timeout
```

BMP → Rotonda → BMP Pipeline

The solution: use Rotonda as a **BMP collector and re-emitter**. Downstream consumers connect to Rotonda's BMP output socket and receive an initial full table dump followed by incremental updates — eliminating the need for repeated full JSON snapshots.

```
Feb 25 11:35:03 rotonda[180544]: bmp-out: BMP client connected from [::1]:57064
Feb 25 11:35:04 rotonda[180544]: bmp-out: Initial table dump started
Feb 25 11:35:04 rotonda[180544]: bmp-out dump: found 2728 peers of type BgpViaBmp
Feb 25 11:35:04 rotonda[180544]: bmp-out dump: sent Peer Up for 2728 peers in 0.00s
Feb 25 11:40:22 rotonda[180544]: bmp-out dump: RIB walk sent 68,897,286 routes in 318.07s
Feb 25 11:40:22 rotonda[180544]: bmp-out dump complete — 2728 peers, 68,897,286 routes, 9816.66 MB in 318.08s
```

2,728 Peers

68.9M Routes

Complete RIB walk completed in ~318 seconds

9.8 GB

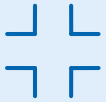
Total data transferred over BMP stream

Memory problems with Rotonda: 112G on 80M routes

```
top - 12:48:01 up 108 days, 17:29,  2 users,  load average: 3.95, 3.43, 3.30
Tasks: 389 total,   3 running, 386 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.5 us,  5.7 sy,  0.0 ni, 87.3 id,  4.5 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 128457.4 total,   1695.5 free, 127422.3 used,    354.3 buff/cache
MiB Swap:   4091.0 total,      0.2 free,   4090.8 used.   1035.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    237 root      20   0       0      0      0 R 100.0   0.0 185:38.36 kswapd0
 180544 rotonda   20   0  110.5g 105.3g      0 S  85.7  83.9     8w+4d rotonda
 180765 bgpviewd  20   0   24.7g  17.2g    452 S  40.9  13.8     67,01 bgpviewd
 384590 root      20   0   50404  26112   1536 R  40.5   0.0   0:14.45 landscape-sysin
 384536 root      20   0   14972   3328   2048 S   4.3   0.0   0:00.59 sshd
 369234 mosquit+  20   0   14736   2304   1536 S   0.3   0.0   1:03.71 mosquitto
 372450 root      20   0       0      0      0 I   0.3   0.0   0:17.12 kworker/24:2-events
      1 root      20   0   22340   5632   1280 S   0.0   0.0   1:41.01 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:01.13 kthreadd
      3 root      20   0       0      0      0 S   0.0   0.0   0:00.00 pool_workqueue_release
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-rcu_g
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-rcu_p
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-slub_
      7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-netns
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
     11 root      20   0       0      0      0 I   0.0   0.0   2:43.37 kworker/u64:0-ext4-rsv-conversion
     12 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-mm_pe
     13 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_kthread
```

Netomics BGP



Gzip Support

Enables compression for API responses, drastically reducing bandwidth usage and transfer times for large BGP table dumps

JSON Streaming

Optimized for processing and delivering large JSON datasets incrementally, avoiding memory bottlenecks for full BGP table pulls.

TLS Support

Integrates Transport Layer Security for all API communications, ensuring encrypted and authenticated data transfers.

BMP Emission

Adds the capability for Rotonda to act as a BMP speaker, forwarding processed BGP information to other monitoring systems.

Learn more: **https://github.com/fastnetmon/rotonda**

Netomics BGP: BMP improvements

- Adds bmp-tcp-out support for exporting RIB/BMP state to BMP clients.
- Includes BMP message building, client state management, status reporting, metrics, and documentation.
- BMP output production features
  - Adds ACL support for client access control
  - Adds TLS support with certificate/key configuration
  - Adds dynamic client buffering
  - Supports dumping multiple ingresses and transitioning from initial dump to live updates.
- BMP protocol correctness
  - Improves PeerUp/PeerDown lifecycle handling
  - Fixes peer identity/state synchronization between BMP input and output
  - Adds correct IPv4/IPv6 peer/nexthop handling.
  - Sends graceful restart capability and End-of-RIB markers so downstream BMP consumers receive complete dump boundaries

Netomics BGP: BGP improvements

Adds TCP MD5 authentication for BGP peers through md5_key configuration

```
[units.bgp-in-v4]
type = "bgp-tcp-in"
listen = "0:0:01:179"
my_asn = 42880001
my_bgp_id = [1,2,3,4]

[units.bgp-in-v4.peers."169.254.169.254"]
name = "Core4"
remote_asn = [ 65001 ]
md5_key = "SecretKey"
protocols = ["Ipv4Unicast"]

[units.bgp-in-v6]
type = "bgp-tcp-in"
listen = "[2001::1]:179"
my_asn = 42880001
my_bgp_id = [1,2,3,4]

[units.bgp-in-v6.peers."2001:19f0:ffff::1"]
name = "Core6"
md5_key = "SuperSecret2"
remote_asn = [ 65001 ]
protocols = [ "Ipv6Unicast"]
```

Netomics BGP: API improvements

- HTTP API memory and compression improvements
  - Enables gzip API responses.
  - Streams JSON responses to reduce memory usage for large outputs.

Netomics BGP: scalability improvements

- RIB memory and withdrawal behavior
  - Adds optional retention of withdrawn route attributes.
  - Adds path attribute deduplication to reduce memory use.
  - Improves withdrawal handling under concurrent peer activity.

- MRT to BMP export compatibility
  - Fixes MRT-derived path attribute encoding so MP_REACH_NLRI is emitted correctly for BMP output

Thank You!

LinkedIn

linkedin.com/in/podintsov

Email

pavel@fastnetmon.com