LINEヤフーの Private Cloudを支える LBaaSの進化

LINEヤフー株式会社

石崎 優

LINEヤフー

石崎 優

Cloud Infrastructure本部
NetDev部 ネットワークフォワーディングチーム

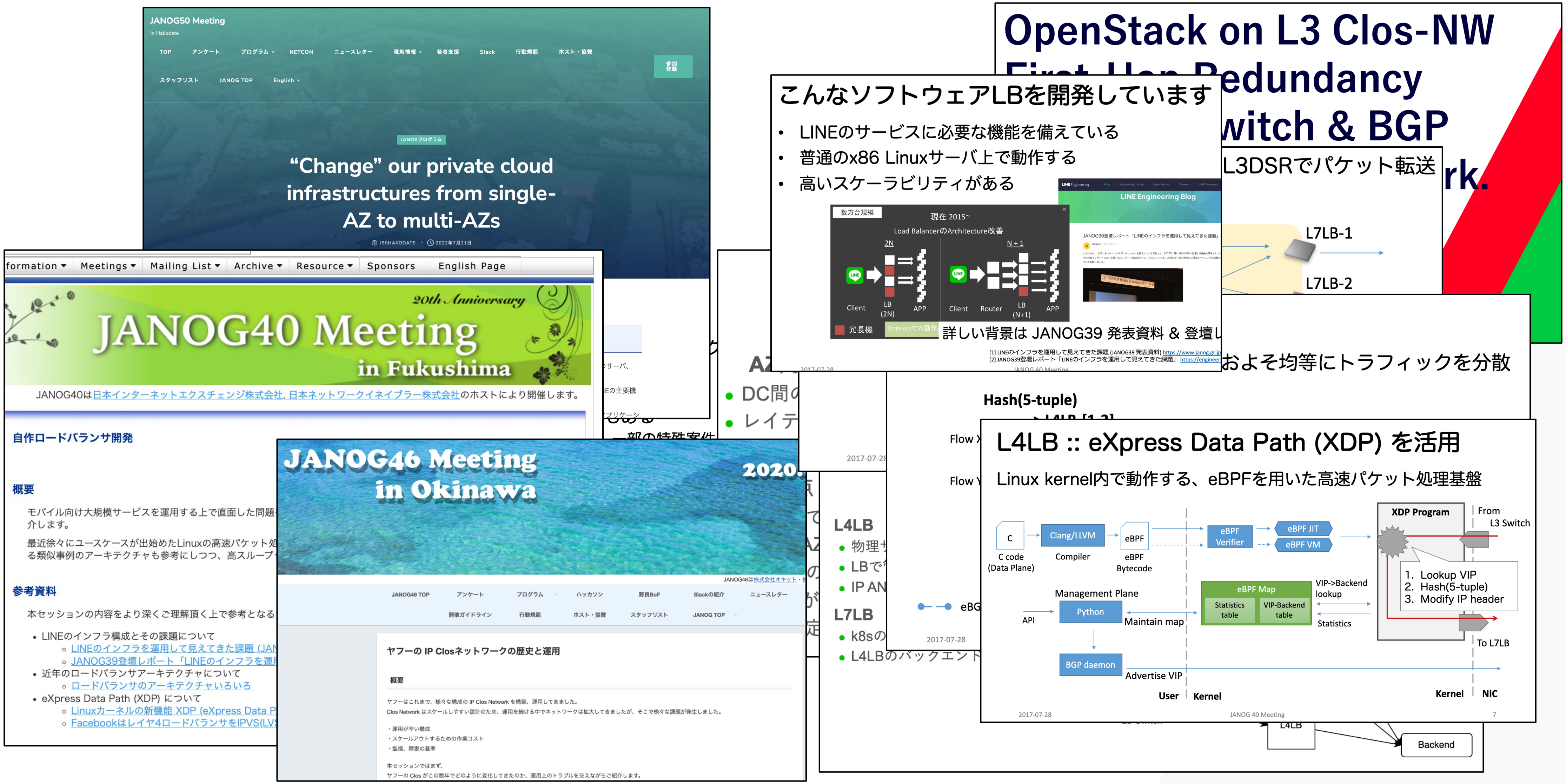
2022 新卒入社

2023 旧プライベートクラウド基盤における
仮想ネットワーク周りの運用・開発

2024 統合プライベートクラウドFlavaの
LBaaSの設計・開発を担当

2025 Flava Network LBのProduct Owner

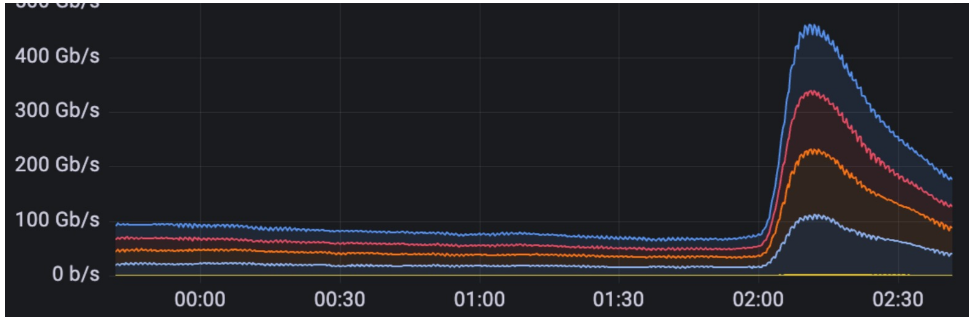
LINEヤフーのSoftware LBaaSのこれまで



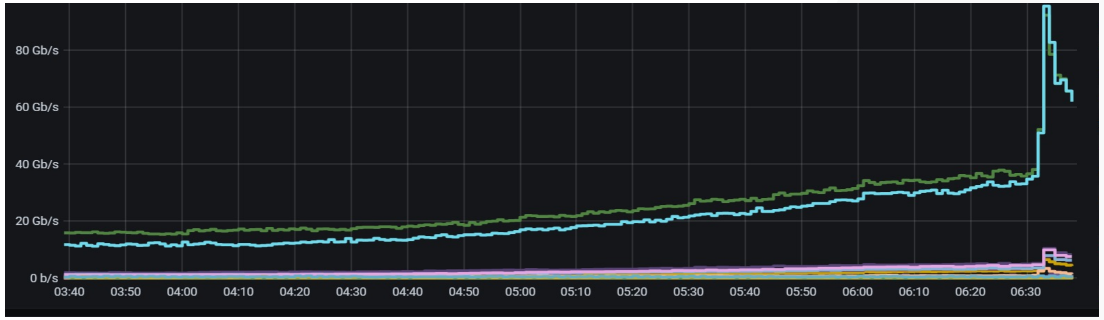
LINEヤフーのSoftware LBaaSの今

	LINE	Yahoo! JAPAN
Hypervisor	10,000+	20,000+
Virtual Machine	117,000+	210,000+
Database Instance	8,500+	5,800+
Kubernetes Cluster	1,000+	1,700+
Peak LBaaS Traffic	460Gbps+	230Gbps+

New Year Traffic

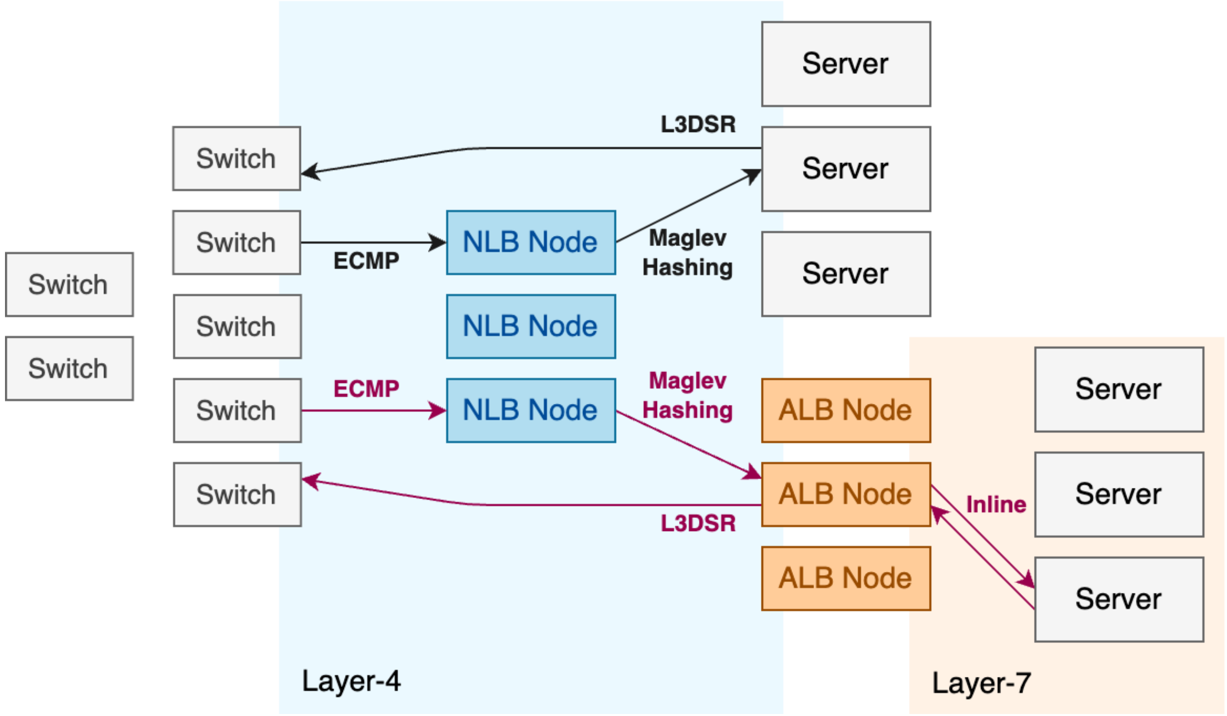


地震速報Push通知後



実際にほぼ全てのLBをSoftware LBaaSに置き換え数年間運用。
汎用サーバによる大幅なコスト削減、柔軟な設定、スケーラビリティ、耐障害性の高さ。

LINEヤフーのSoftware LBaaSの今



➢ 実際に大規模Trafficを受けつつ運用してみて生じた課題と解決について一部紹介。

旧LINE、旧Yahoo! JAPANで開発してきたSoftware LBaaSを発展させ、
統合プライベートクラウドFlavaのSoftware LBaaSを開発・投入。

大量Flow時のThroughput改善

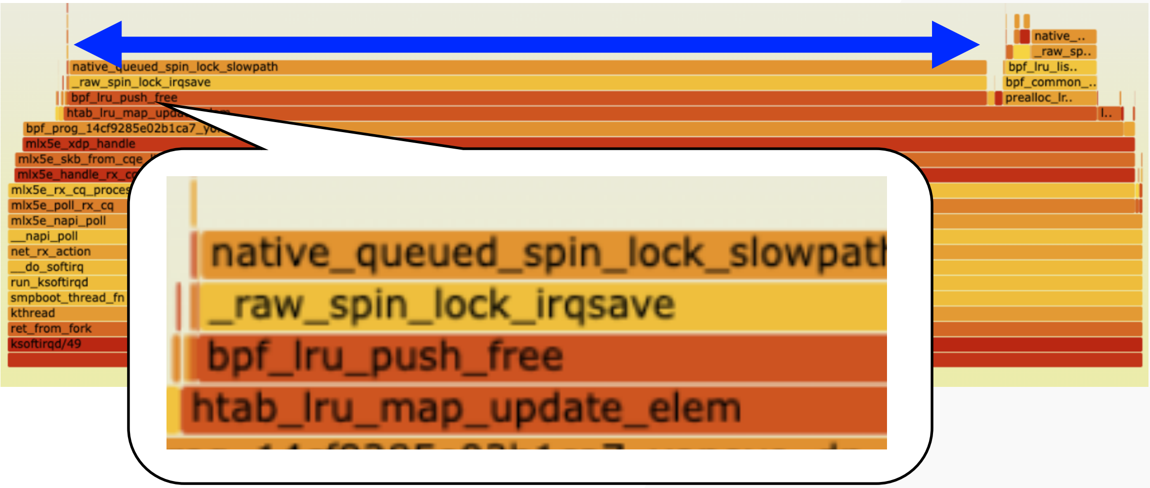
課題

LBではMaglev Hashing Tableの変化により転送先として選択されるbackendが変わることで
既存のsessionが落ちてしまわないように、転送先として選択したbackendのcache tableを持つ。

➢ **flow数が2,000を超えたあたりでthroughputが著しく劣化し始める。**

目標とするNICのラインレートが実現できない。集約率低下。コストの増加。

eBPF mapのBPF_MAP_TYPE_LRU_HASHは全CPU間で共有。flowが多くなると各CPUがLRUの追い出しのために共有のカウンタをロックすることで遅くなることが判明。

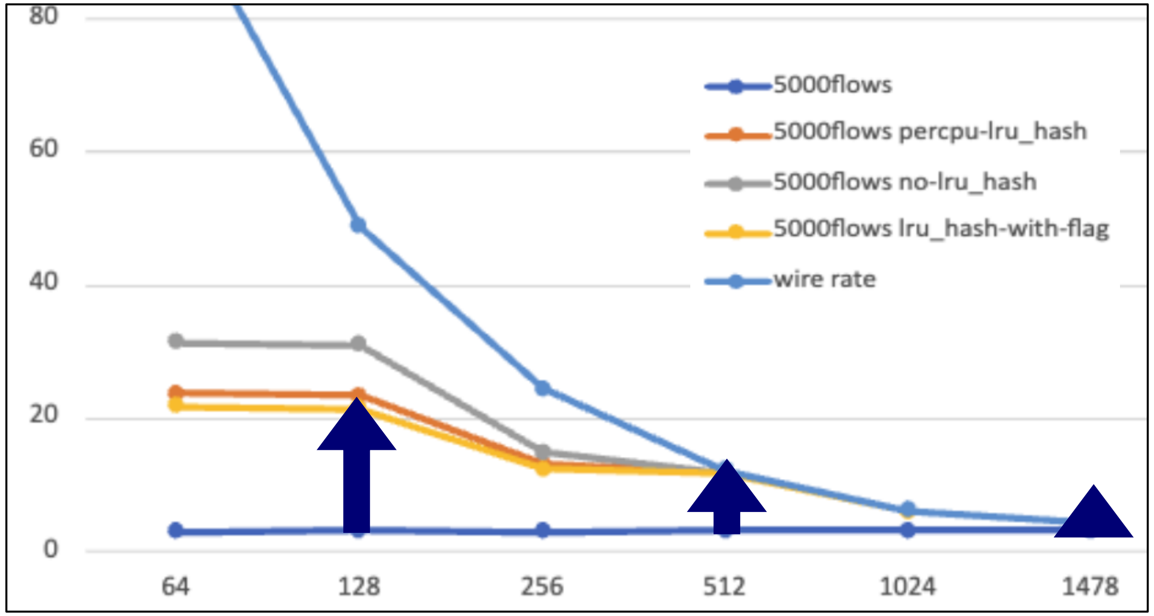


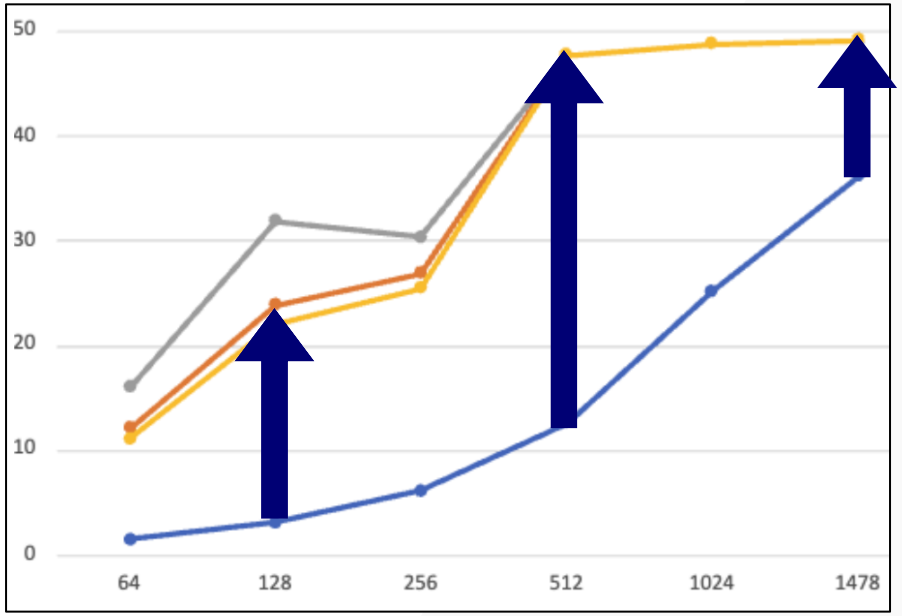
大量Flow時のThroughput改善

解決

eBPF mapのBPF_F_NO_COMMON_LRUフラグをつけることでCPUごとにLRUカウンタを持ち、追い出し候補を選出。(実際の追い出し時には全体で使われていないかチェックされる)

➢ ワイヤーレート限界の同時接続flow数(数万flow)までthroughputが向上。





Health Check Agentの性能改善

課題

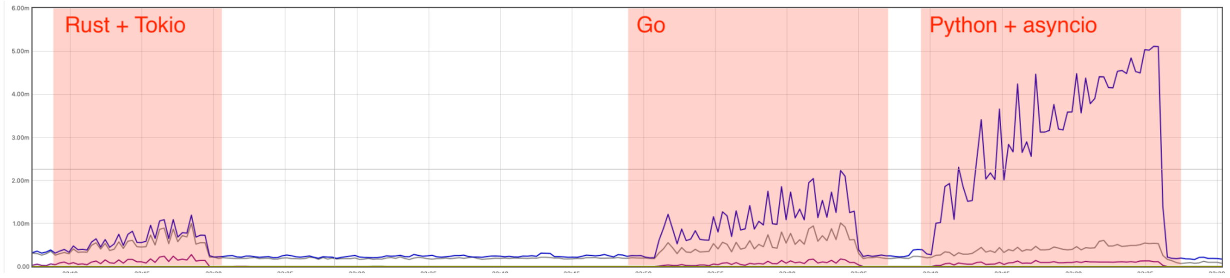
既存のHealth Check Agentの性能限界によりLB nodeにデプロイされるVIPの数が多くなると
1つ1つのHealth Checkかかる時間が大幅に増加、最悪の場合Agentが停止。

➢ **Health Check結果の反映ができない。**

1台のLB nodeにデプロイできるVIP数を制限するため、
LB nodeの集約率を下げてコスト増加の要因に。

既存のHealth Check Agentの詳細な性能解析により
大幅な性能改善の余地があることが判明。

CPU Usage



Result

In this benchmark, we measured the duration of H/C probes and identified when timeouts begin to occur by varying the number of concurrent probes.
Each benchmark was executed three times per configuration, and the reported results are the average of these runs.
Cells highlighted in orange indicate that a timeout (3 seconds) occurred.

	Python (sec)	Go (sec)	Rust (sec)
3000 (Req)	2.65	0.96	0.63
5000 (Req)	4.33	1.06	0.73
8000 (Req)	6.04	1.38	1.08
12000 (Req)	8.15	2.26	1.63
16000 (Req)	10.0	2.43	1.98
20000 (Req)	11.8	3.29	2.15
24000 (Req)	14.4	3.80	3.06
28000 (Req)	16.4	4.03	3.34

Health Check Agentの性能改善

解決

- 複数の非同期ランタイムを比較し、高いパフォーマンスと安定性を達成したRustのtokioで再実装。
- ヘルスチェックに最適化したHTTPクライアントを自作し、必要な処理のみ実行。
- ヘルスチェックはクライアント側のTLSハンドシェイクが大量に発生する特殊なワークロード。
  クライアント側の鍵交換のスループットが良いTLSライブラリ（rustls）を使用。

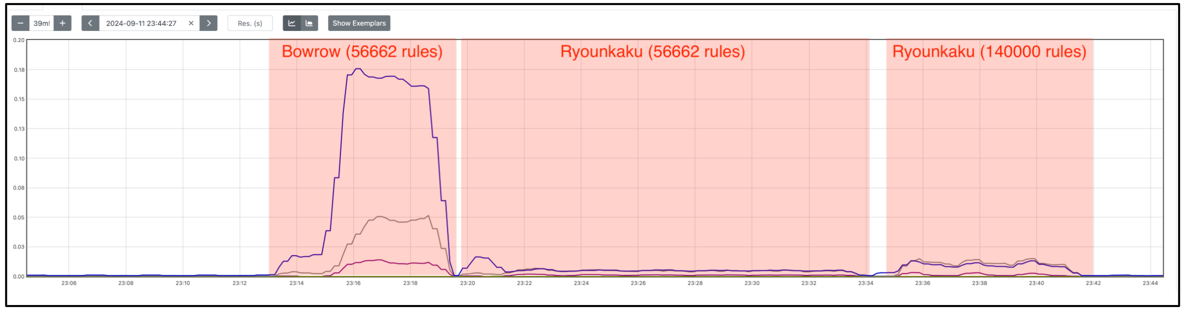
➢ CPU利用率を1/20以下に抑えられた。

大量のhealth checkを同時に走らせることが可能に。集約率の向上。

Result

We can see the following observations for Ryounkaku to be compared with Bowrow

- Ryounkaku consumes **1/20** CPU time compared with Bowrow.
- Ryounkaku handles **140000 conditions** without any failures such as timeout by keeping low CPU time for Bowrow with 56662 conditions.



LB Hashing Table更新速度改善

課題

まとまった数のbackendが同時にup/downするとLBの転送先の更新に時間がかかる (数十秒~数分)
他のbackendの更新にも影響を与えてしまっていた。

➢ **downしたbackendへの振り分けが止まらない。**

一定量のbackendのup/down状態の変化はよく起こる。
- ユーザのメンテナンス等での大量のbackend操作
- LB自体のメンテナンス操作
- 一時的なNetwork障害

NLBの転送先決定アルゴリズムとして
GoogleのMaglev Hashingを利用している。
各nodeでbackend serverの死活状態が変化する度に再計算される。
この更新処理に時間がかかっていることがプロファイリングできた。

Background:

Previously, our API paths used unique IDs for VIP, port, and backend,
requiring additional queries to retrieve these IDs before making requests.
This approach led to unnecessary complexity and increased the number of requests.

Key Issues

Performance Bottleneck:
When the status of many backends changed simultaneously,
the system required tens of seconds to process all updates due to excessive ID lookups.

```
Enabling/disabling one backend tak
2,000 backends (100VIPs, 20 backen
```

Loss of Idempotency:
Using IDs caused failures when trying to
As a result, if communication with the so

What To Do

Update API path parameters to use VIP a
Introduce a new hash map (inverse map)

- `VIP Address` → `VIP ID`
- `VIP Address` + `Protocol` + `Port N`
- `VIP Address` + `Protocol` + `Port N`

Background:

Previously, when backends wen up or down, the maglev hash table was recalculated individually for each backend.
The calculation had a complexity of **O(MlogM)** on average and **O(M²)** in the worst case.

With the current M=13313, the calculation took approximately 30ms per backend.
In a scenario with 100 VIPs, each having 2 ports and 10 backends, a total of 2,000 backend changes would take over a minute to process.

This delay caused the system to become stuck and unable to process backend changes correctly when multiple backends went down simultaneously.

Additionally, when multiple backends changed state simultaneously, the hash table recalculations occurred sequentially.
This led to shuffling, where the chosen destination backend changed repeatedly during processing, negatively affecting connection stability.

Since maglev hash tables are maintained per port, there was an opportunity to optimize the process by grouping backend changes per port and parallelizing hash table recalculations across ports.

What Was Done:

1. Batch Processing of backend changes

This PR updated the API and internal implementation to process backend state changes (up/down) by port,
instead of handling them individually for each backend.

2. Parallel processing across ports

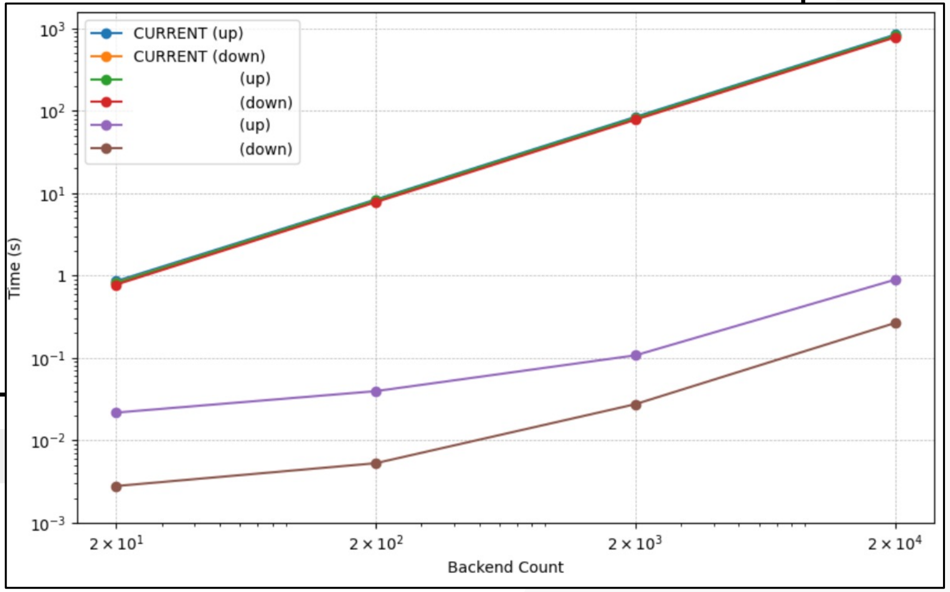
LB Hashing Table更新速度改善

解決

Maglev Hashing Tableの計算の工夫とeBPF map上のデータ構造を見直すことで
1000倍以上の計算高速化に成功

➢ backendの死活状態が即時LBの転送に反映されるように

- 計算対象のサーチをO(1)に
- VIP/PortごとにTable計算は独立させられるので並列化
- down/upしたbackendの集合をバッチ計算



Results

Files were placed 10 times for up and 10 times for down,
and the measurement results show the time taken to process these files.

20 backends (1 VIP, 2 ports)		200 backends up (10 VIPs, 2 ports)		2,000 backends (100 VIPs, 2 ports)		20,000 backends (1,000 VIPs, 2 ports)	
up	down	up	down	up	down	up	down
858 ms	821 ms	8.4 s	8.1 s	84.5 s	80.2 s	841 s	804 s
817 ms	778 ms	8.2 s	7.8 s	81.9 s	78.0 s	817 s	781 s
21.8 ms	2.8 ms	39.7 ms	5.3 ms	108 ms	27.6 ms	895 ms	267 ms

アーキテクチャの重要性

Software LBaaSは開発して終わりではない。最初から作り直す時間もない。

優先度の高い仕事は他にも

- ユーザサポート
- 機能追加、バグ修正
- 他プロダクトの開発

➢ **責任を持って運用し続ける必要性。**

こんなシステムは避けたい

- アラートや事故が多いシステム
- 何かあったら手動オペレーションが必要なシステム
- 簡単にスケール限界を迎えるシステム
- etc.

➢ **インフラとしてLBaaSは確実な安定稼働が求められる。**
(自分たちで開発したSoftwareだからこそ言い訳できない)

アーキテクチャの重要性

システムの安定性や運用時の負荷はD-PlaneよりもC-Planeで決まる

エンジニアとしての前提

- 処理高速化
- 必要な技術について知る
- 最新の技術をキャッチアップ
- 手を動かして作る
- AIを活用

システムアーキテクチャ

- トレードオフ
  - スケーラビリティとデータ整合性
  - 分散システム
  - 非同期システム
- 耐障害性・外乱耐性の高いアーキテクチャ
  - 障害パターンをどのように網羅するか
  - 障害発生時に加えて復帰時、フラップ時
  - 忘れていたでは済まされない

安定稼働するアーキテクチャを選ぶ重要性
アーキテクチャの選球眼を磨く重要性

Appendix

From Hardware LB to Software LB

Hardware LB

- **Expensive** dedicated hardware appliances. **High annual support and licensing cost.**
- **Low scalability** and **limited fault tolerance** due to **1+1 redundancy**.
- **Long lead time** to purchase, deliver, and deploy new appliances.

Software LB

- Built on **low-cost general-purpose servers** with **in-house software.**
- **High scalability** and **strong fault tolerance** through **distributed architecture**.
- **Short lead time** to add general-purpose servers.
- **Add new features** via software updates.
- **Seamless integration** with our private cloud.

Network LB and Application LB

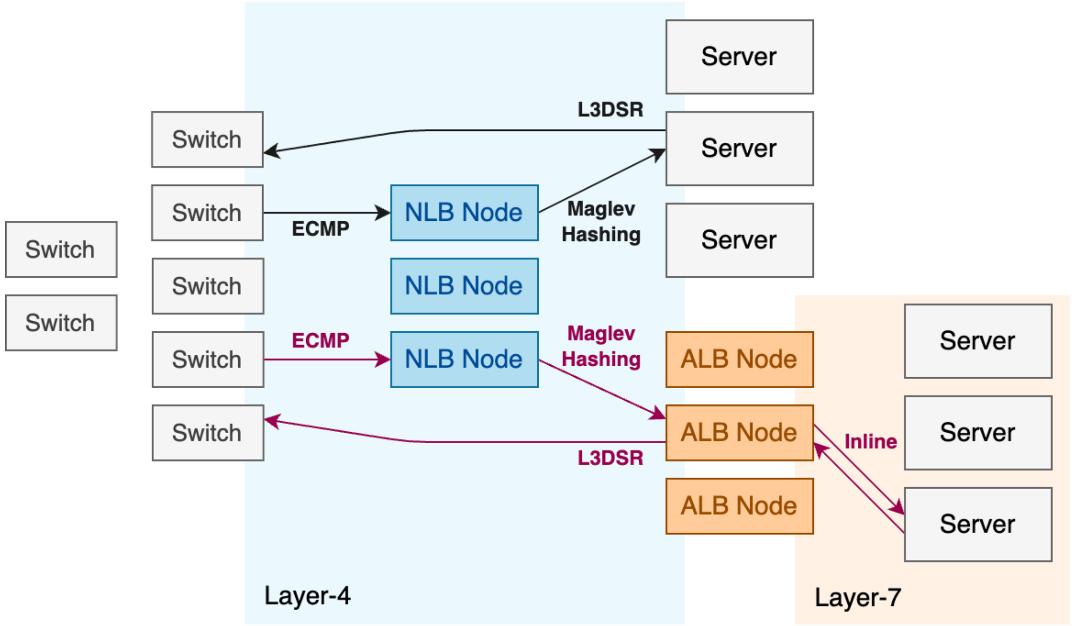
Network Load Balancer

- Operates at Layer 4 (Transport layer).
- High-throughput, low-latency routing.
- Users must manage anything above the TCP/UDP.

Application Load Balancer

- Operates at Layer 7 (Application layer).
- Performance is lower than Network Load Balancer.
- Handle HTTP/HTTPS traffic with advanced routing rules (e.g., URLs, headers).

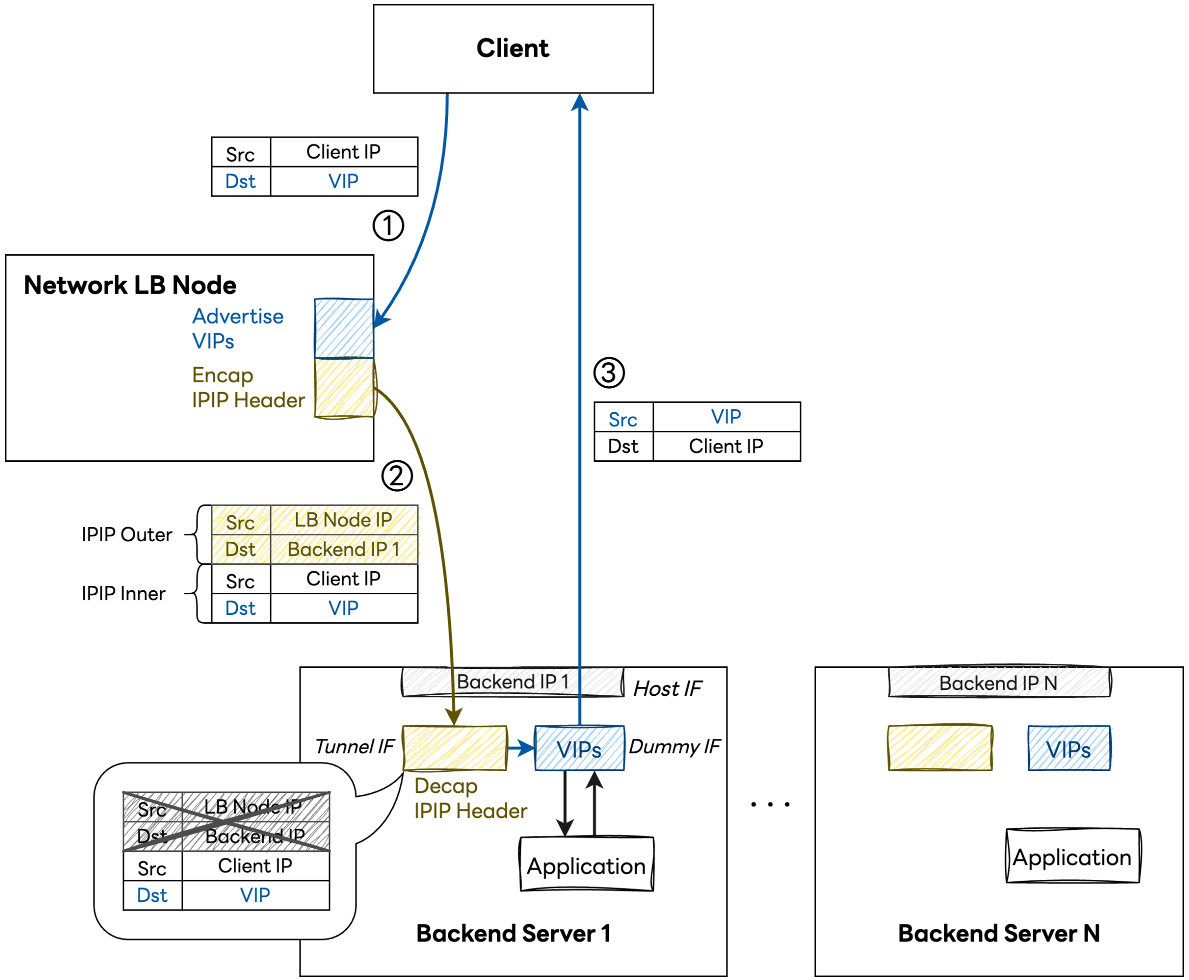
Our Software LBaaS



- Network LB traffic: **Traffic routed via NLB nodes to servers.**
  Servers return traffic directly to clients.
- Application LB traffic: **Traffic routed via NLB nodes, then ALB nodes, to servers.**
  ALB node – server communication is inline (proxy mode). ALB nodes return traffic directly to clients.

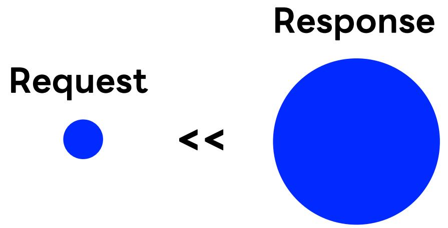
NLB nodes and ALB nodes are implemented on general-purpose servers.

Direct Server Return: IPIP Encapsulation



Direct Server Return (DSR)

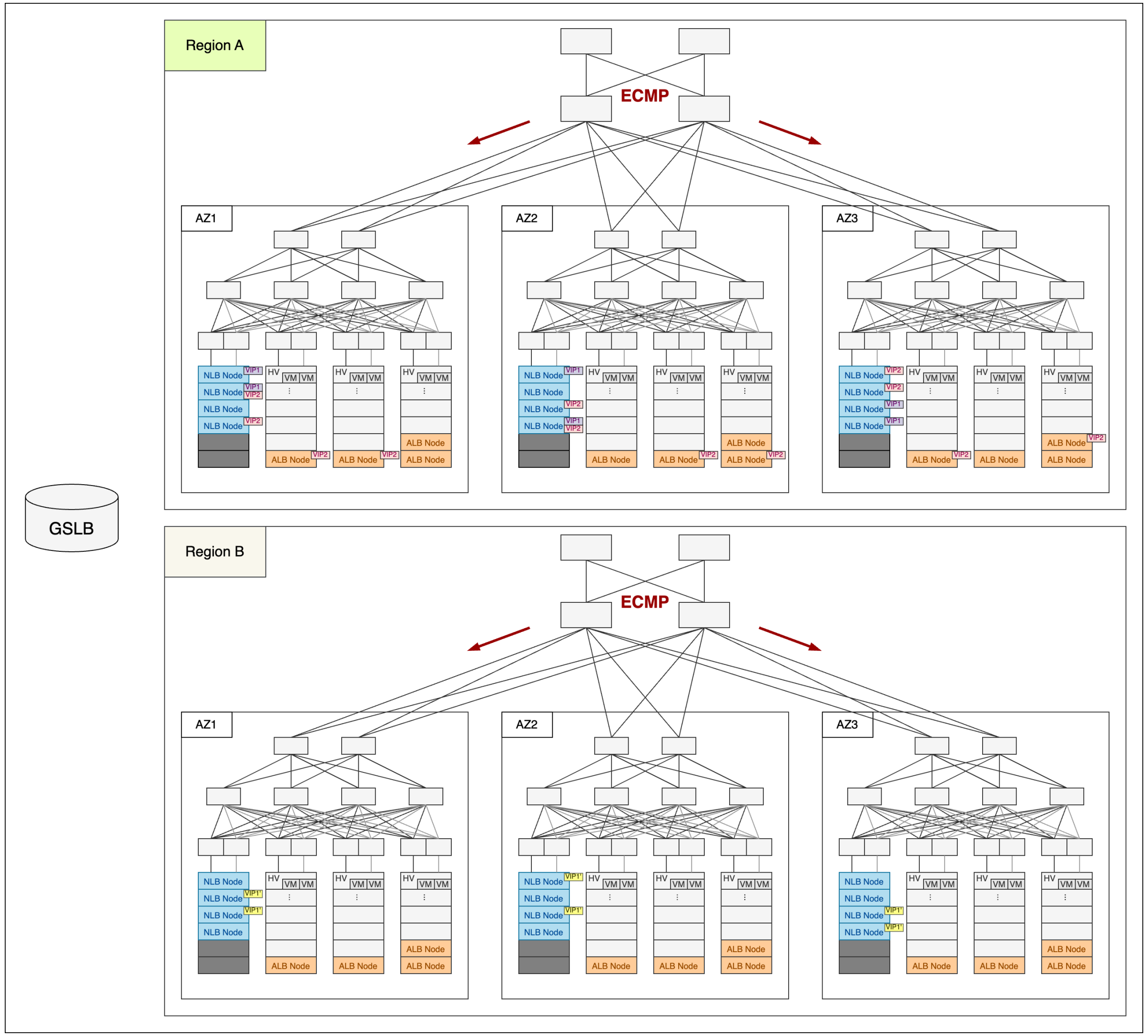
- Request goes through the LB node.
- Server responds directly to the client.



Significantly reduces LB nodes' load.
Low latency / High throughput

1. Client traffic to a **Virtual IP (VIP)** reaches LB nodes.
2. The LB node **encaps** the packet with **IPIP** to forwards it a backend server. The backend **decaps** it, restoring the original.
3. The backend responds to the client with the **VIP** as the source.

Multi-Region, Multi-AZ



High Availability of Load Balancer

Same VIP deployed across multiple LB nodes for redundancy.

VIPs Scheduling to LB Nodes

- VIPs are scheduled across AZs.
- Similar to pod scheduling in Kubernetes.

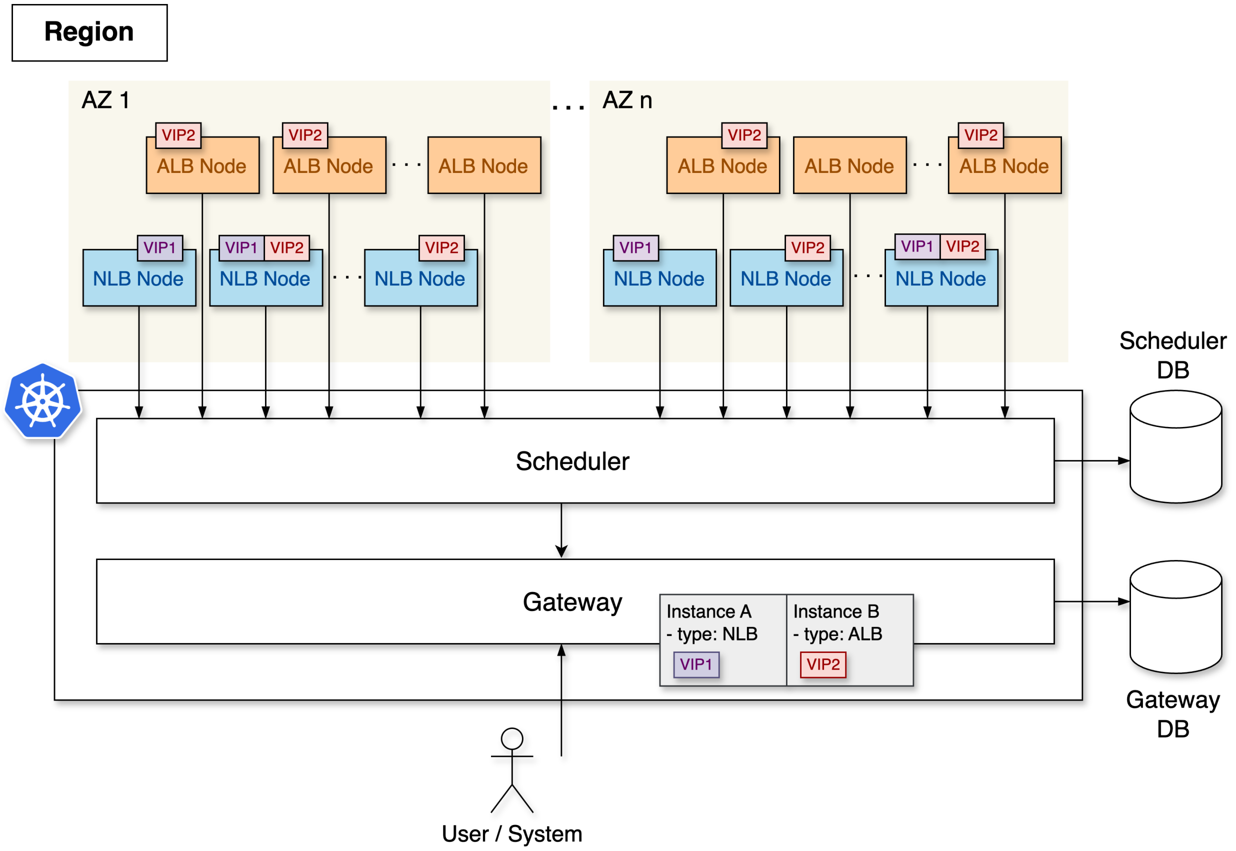
Traffic Distribution (3 steps)

1. Across **regions** via **GSLB**
2. To **Network LB nodes** via **ECMP (Equal-Cost Multi-Path)** through switches in CLOS networks
   (Based on 5-tuple hash: src/dst IPs, ports, and protocol)
3. To **backend servers** from **Network LB nodes**

Consistent Backend Selection

- ECMP **lacks session consistency**, so all LB nodes must select the same backend per session.
- **Maglev Hashing** ensures identical selection hash tables across LB nodes. (Each LB node builds its own table.)

Our Software LBaaS: Control Plane



Gateway

Responsible for user interaction with the LB system.

- Handles user requests for LB instances.
- Assigns VIPs to newly created LB instances.
- Requests the scheduler to schedule VIPs.

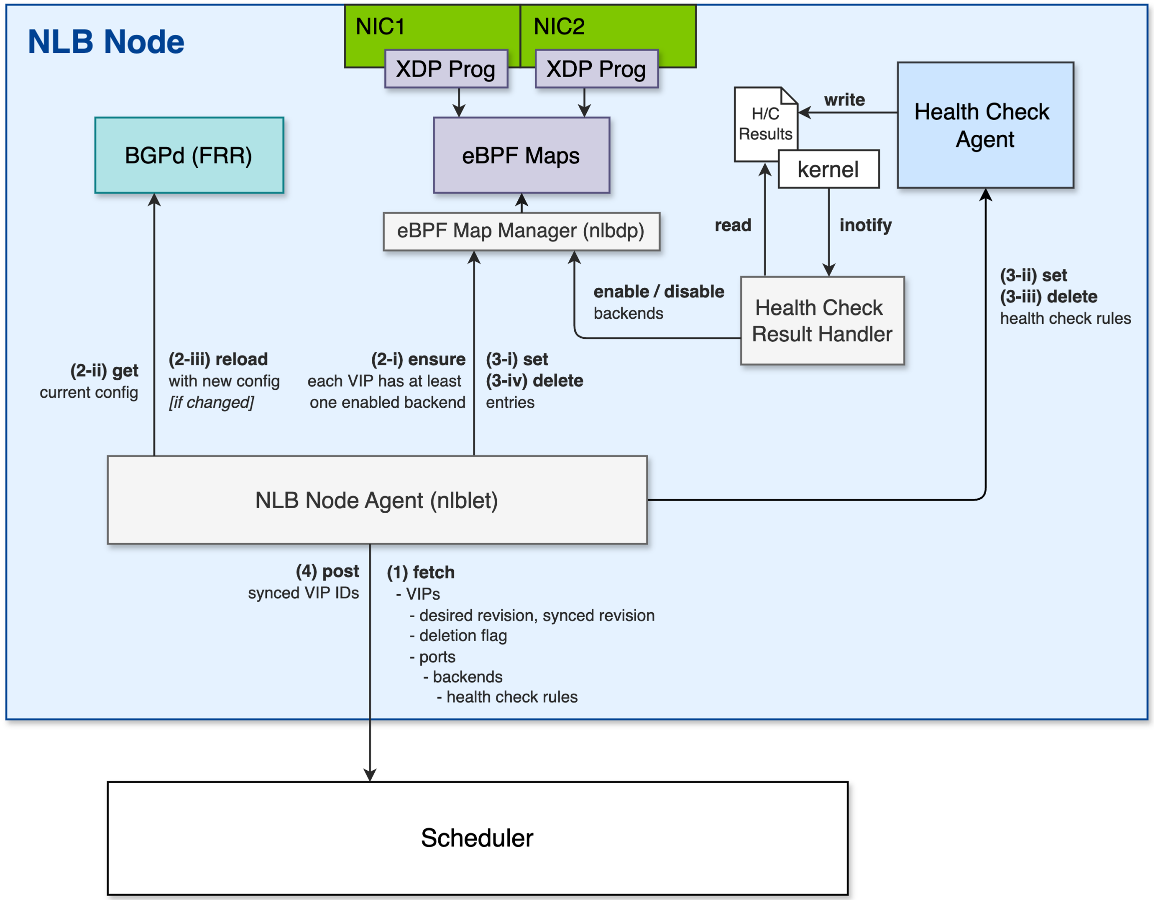
Scheduler

Schedules each VIP to multiple nodes based on its replica count.

- Considers AZ diversity.
- Prioritizes nodes with fewer VIPs.

Independent scaling and loose coupling.

Network LB Node

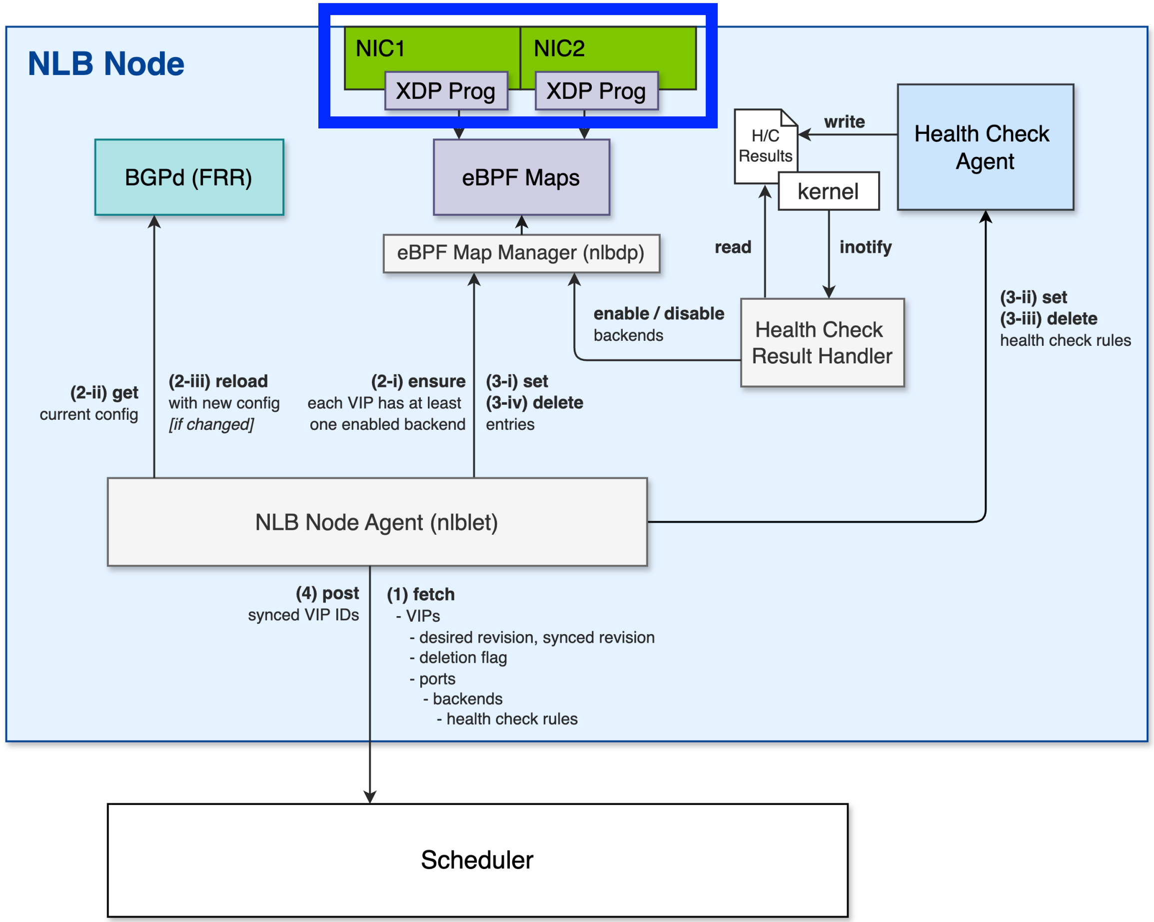


Built on a **general-purpose server** with in-house software.

4 Key Components

- Packet Forwarding
- Health Check
- VIP Advertisement
- Reconciliation

Network LB Node: Packet Forwarding



XDP (eXpress Data Path)

A technology to process **incoming packets** before they reach the kernel's network stack.

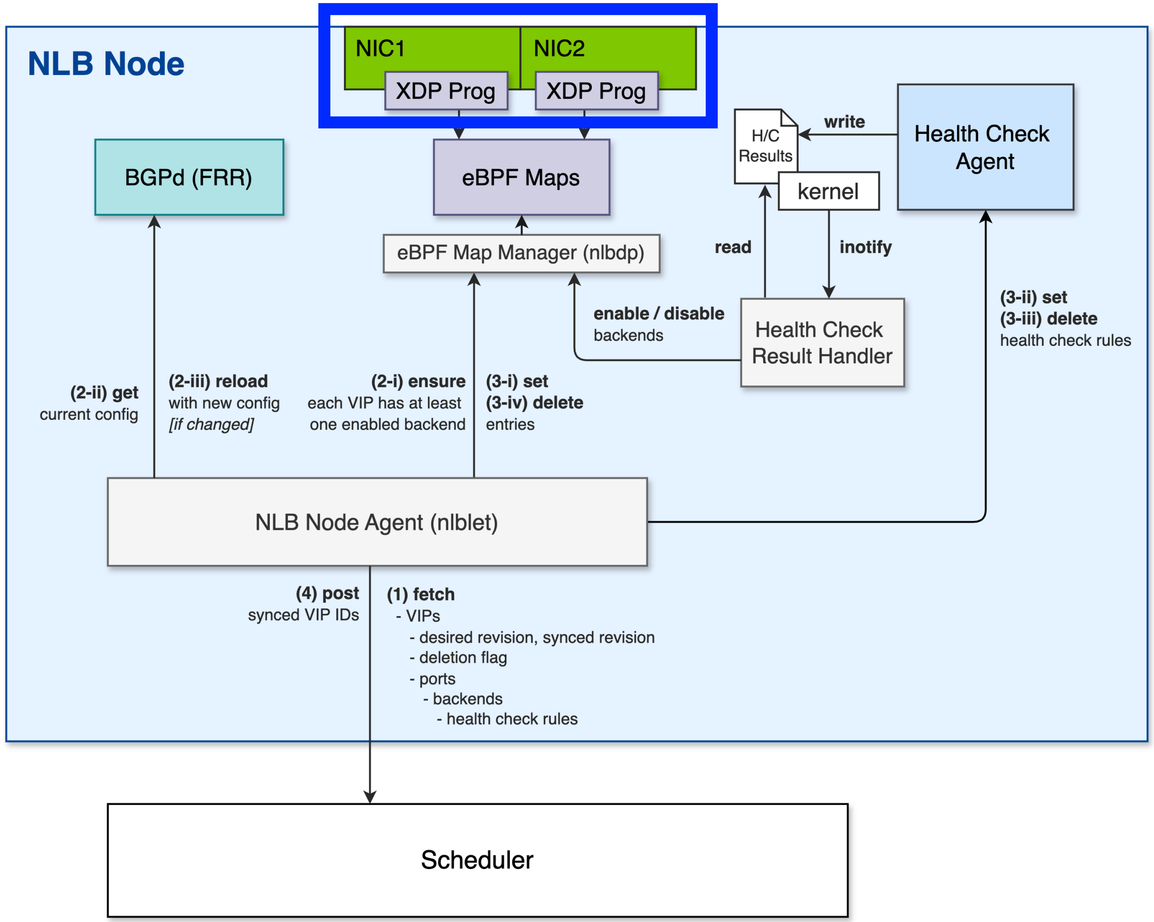
- XDP programs can be embedded in NIC drivers **(XDP native mode).**
- **Immediately after** the transfer from NIC queues to a ring buffer in main memory via DMA,

  an XDP program processes the packets and forwards them to the appropriate backends.

Extremely high performance

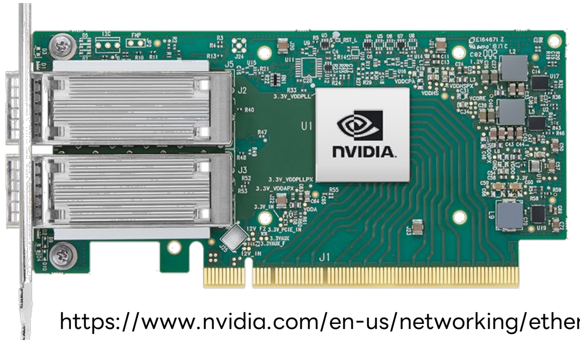
through bypassing memory allocation, copying, and multiple layers in the kernel's network stack.

Network LB Node: Packet Forwarding



NVIDIA® Mellanox® ConnectX®-5

- **25BgE dual-port** (50Gbps)
- Support **XDP** in the driver.
- Support **SR-IOV**.
  (for c-plane and d-plane separation)

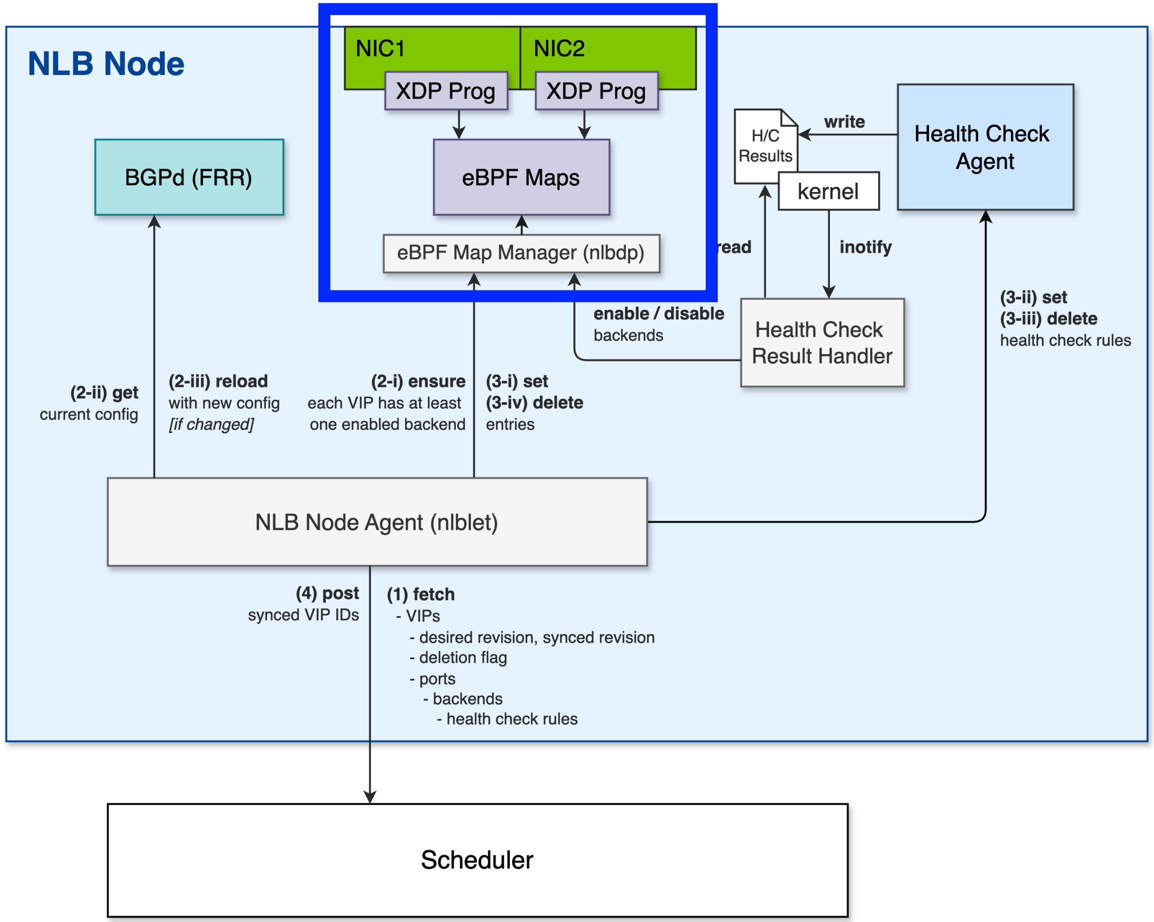


https://www.nvidia.com/en-us/networking/ethernet/connectx-5

The XDP program encaps packets destined for VIPs with IPIP to forward them to the appropriate backends.

Written in **C**, compiled into **XDP bytecode**, and attached to the **NIC**.

Network LB Node: Packet Forwarding



eBPF Maps

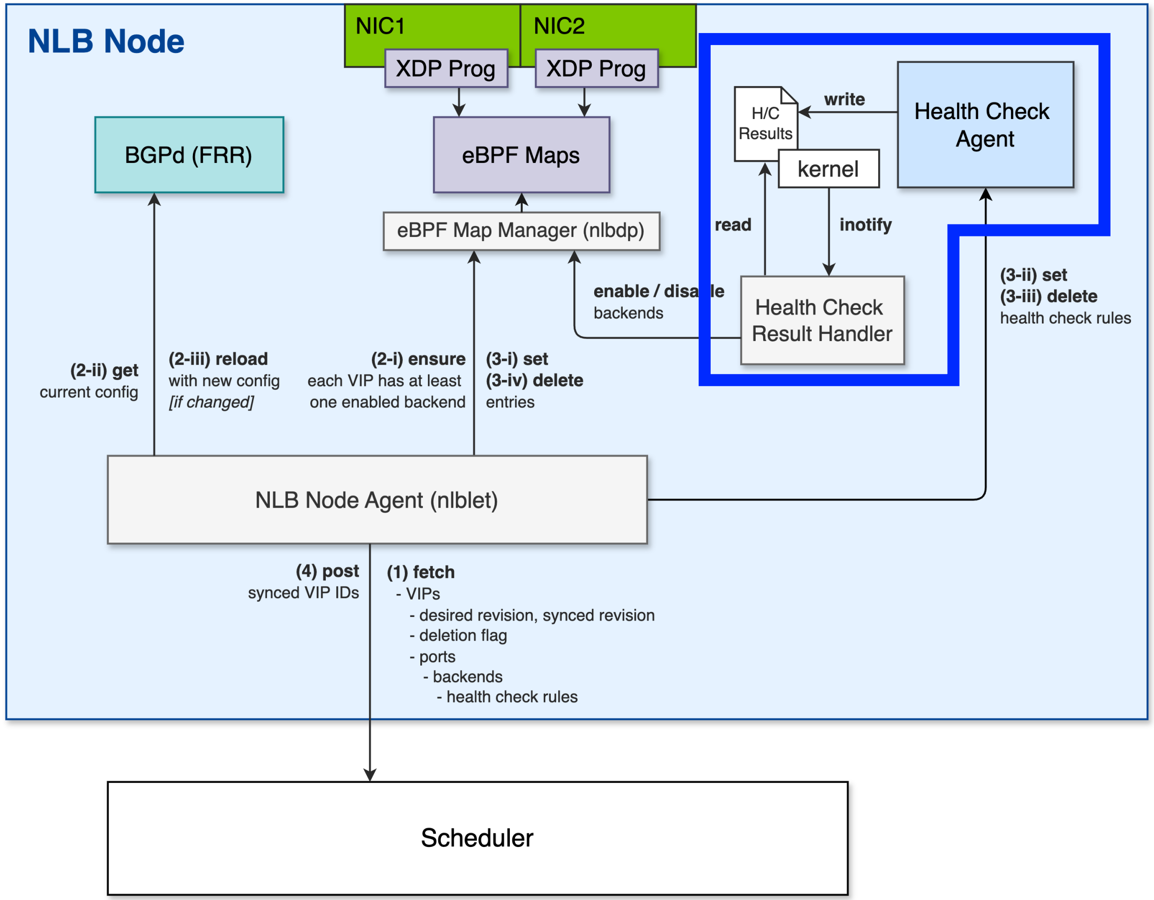
A data structure in main memory designed for XDP to ensure efficient, safe processing and data sharing with user space.

- **Store the forwarding rule tables.**
- **The XDP program looks up the tables** for each packet to determine its destination.

eBPF map manager (API server)

- **Proxies map operations** from **user-space, simplifying map management**.
- **Recalculate and store LB hashing tables per VIP/port** pair in a map whenever the active backend set for that pair changes.

Network LB Node: Health Check



Health Check Agent

Performs **real-time backend monitoring** to exclude down backends from traffic distribution.

Supports **multiple protocol** checks:

- **ICMP**, **TCP**, and **HTTP(s)**.
- Determine backend status by combining multiple check results.
- Redundant checks will be consolidated.

Runs checks continuously at short intervals for each VIP backend, requiring high performance.

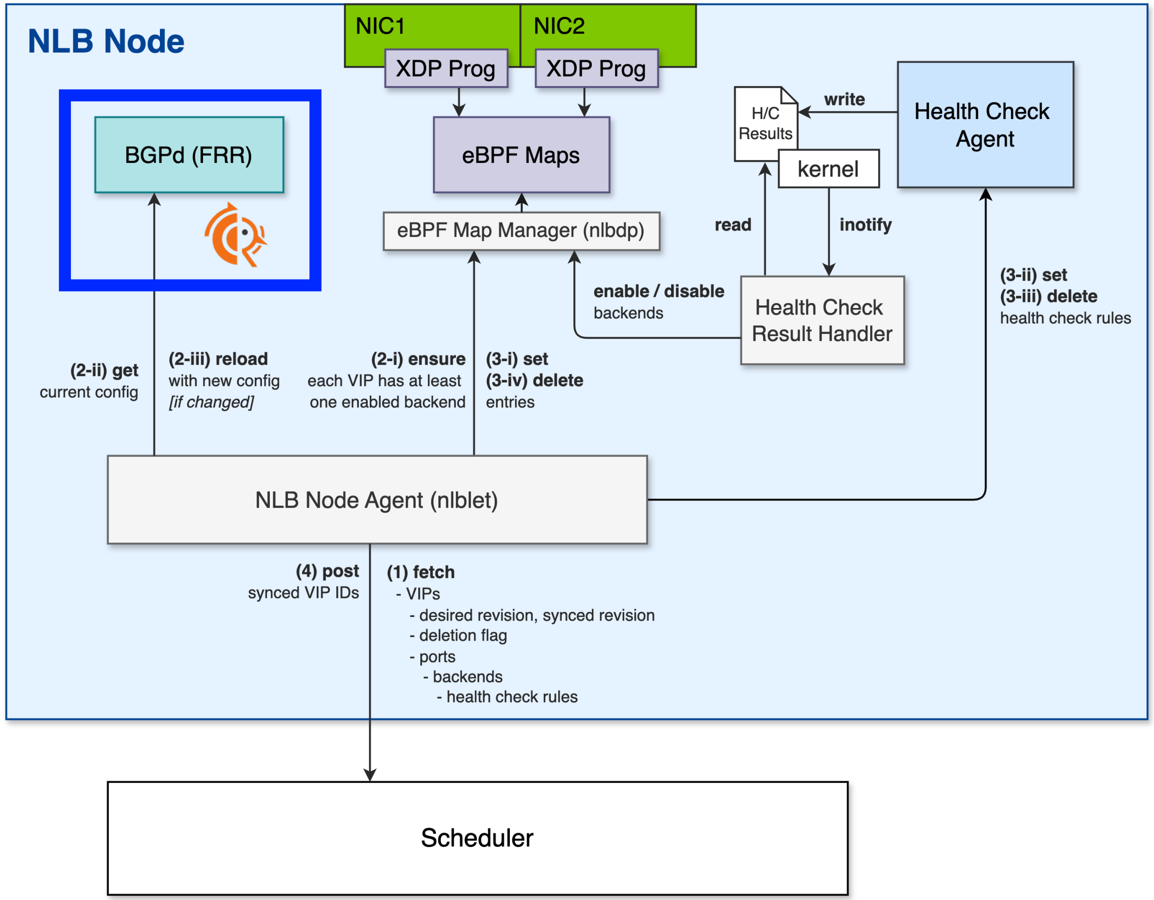
(We implemented the agent in Rust.)

Health Check Result Handler

- Changed health check results are written to a **file**.
- A separate Handler process **syncs the eBPF maps**, triggered by the file update.

Files act as queues, enabling **asynchronous processing** between checks and their application, and preventing stuck during large-scale result changes.

Network LB Node: VIP Advertisement



VIP Advertisement

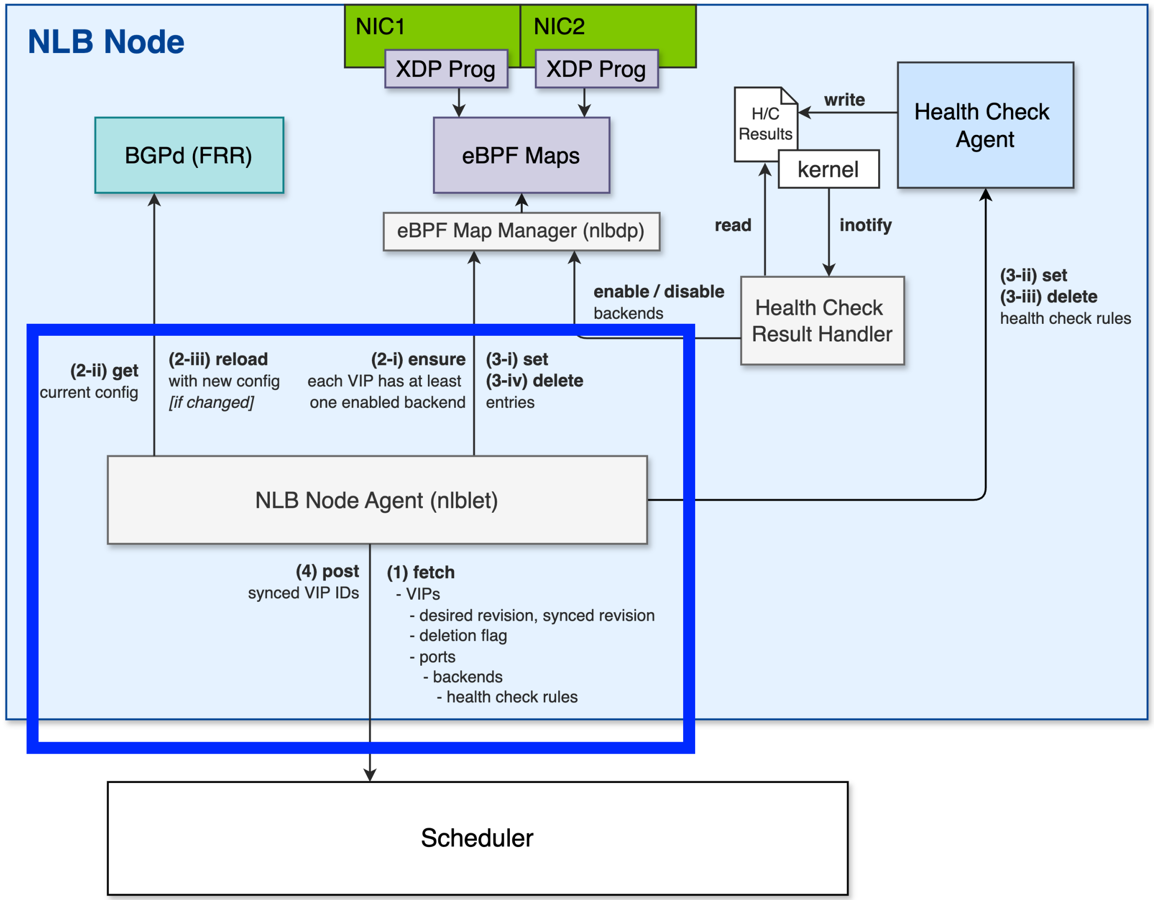
Handled by **FRR**, an open-source software router.

Uses **BGP** to **inform** neighboring networks
to route VIPs to the LB node.

(Advertises /32s for IPv4 VIPs, /128s for IPv6 VIPs.)

Ensures VIP traffic reaches the LB node.

Network LB Node: Reconciliation



NLB Node Agent (nlblet)

(Named after kubelet in Kubernetes.)

Reconciliation loop to keep the node synchronized.

1. **Fetches VIP data** for its node from the external scheduler.
2. **Determines VIPs to advertise** and **reloads BGPd** with the new config if changed.

   *) Only VIPs with at least one enabled backend forwarding rule in the eBPF map.

3. **Syncs the eBPF map** and **the health check agent** settings.
4. **Report synced VIP IDs** to the scheduler.

LINEヤフー