

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

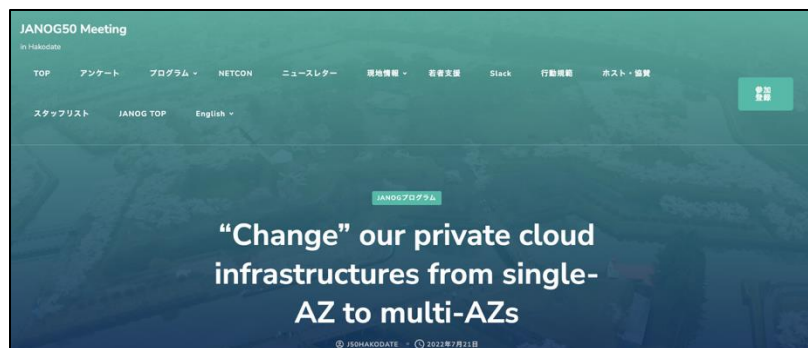
2022 新卒入社

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

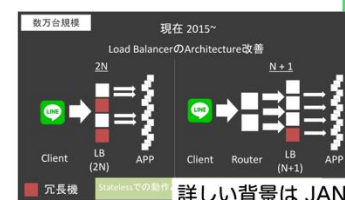
2025 Flava Network LBのProduct Owner

LINEヤフーのSoftware LBaaSのこれまで



こんなソフトウェアLBを開発しています

- LINEのサービスに必要な機能を備えている
- 普通のx86 Linuxサーバ上で動作する
- 高いスケーラビリティがある



詳しい背景は JANOG39 発表資料 & 登壇スライドを参照してください

OpenStack on L3 Clos-NW

First Hop Redundancy

Switch & BGP

L3DSRでパケット転送

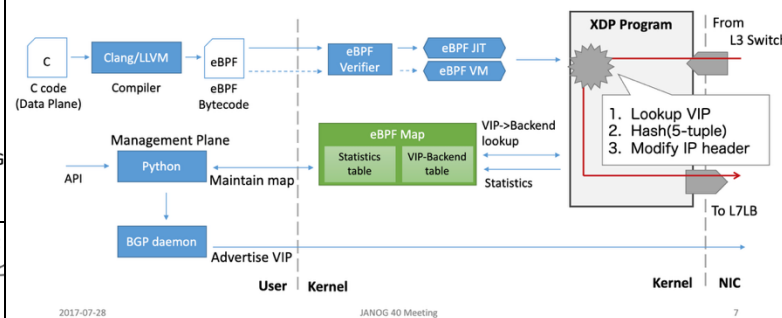


およそ均等にトラフィックを分散

Hash(5-tuple)

L4LB :: eXpress Data Path (XDP) を活用

Linux kernel内で動作する、eBPFを用いた高速パケット処理基盤



L4LB

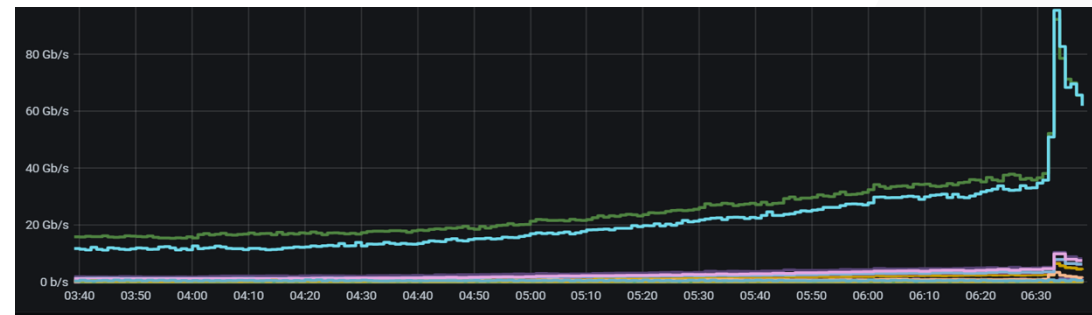
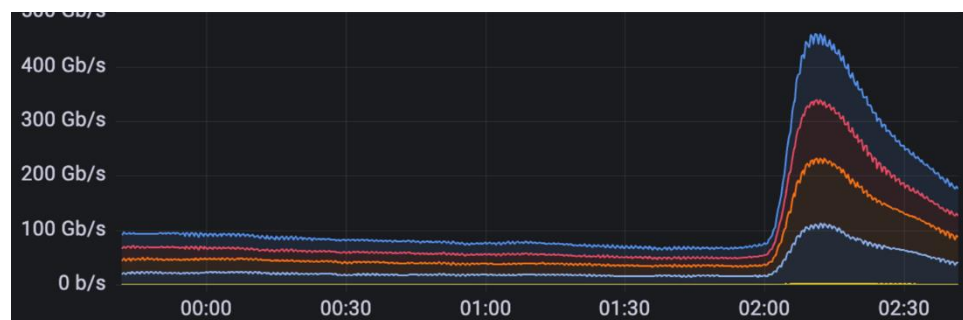
- 物理サーバ
- LBで管理
- IP ANY

L7LB

- k8sのPodsとして構築
- L4LBのバックエンドと

LINEヤフーのSoftware LBaaSの今

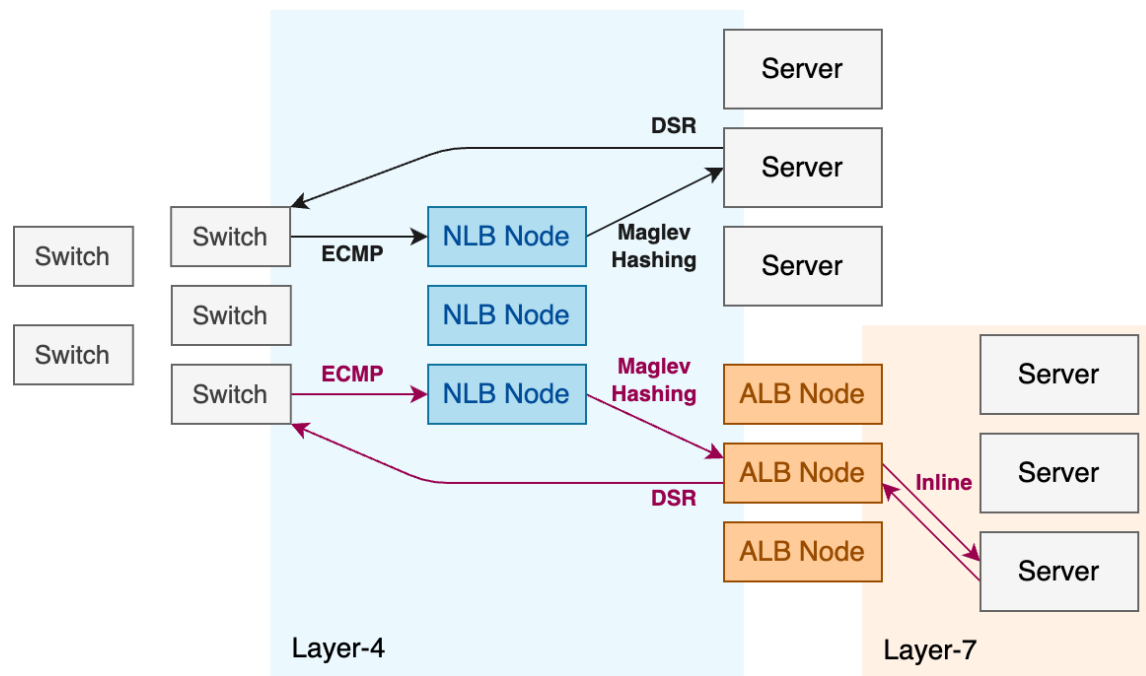
	LINE	Yahoo! Japan
Hypervisor	10,000+	10,000+
Virtual Machine	117,000+	210,000+
Database Instance	8,500+	5,800+
Kubernetes Cluster	1,000+	1,700+
Peak LBaaS Traffic	460Gbps+	230Gbps+



実際にはほぼ全ての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。

旧LINE、旧Yahoo! Japanで開発してきたSoftware LBaaSを発展させ、
統合クラウドFlavaのSoftware LBaaSを開発・投入。

LINEヤフーのSoftware LBaaSの今



- LTなのでLBやXDPの構成について最初から説明はできない...
- 実際到大規模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。

詳しい話は終わった後やブースで対応します！

Throughputの大幅改善

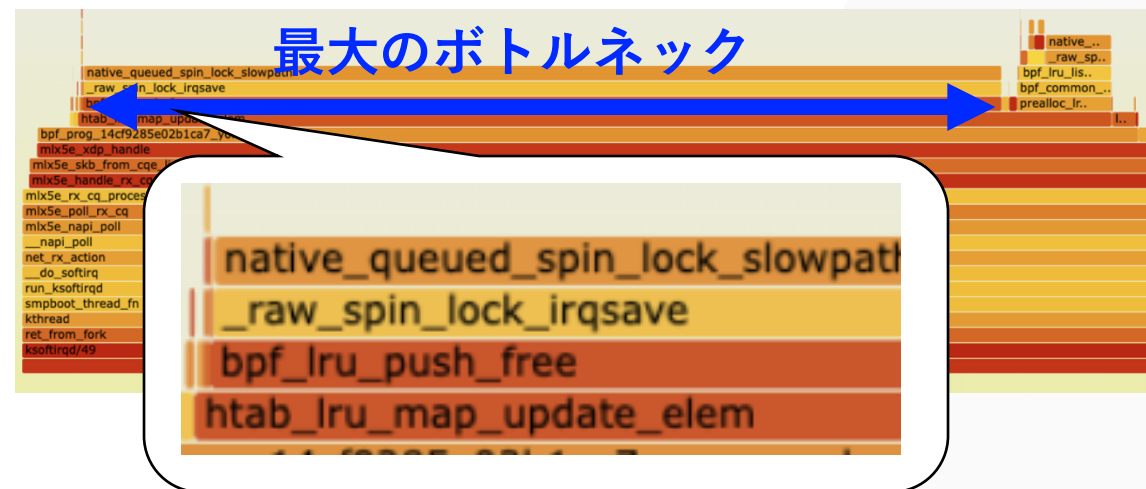
課題

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。

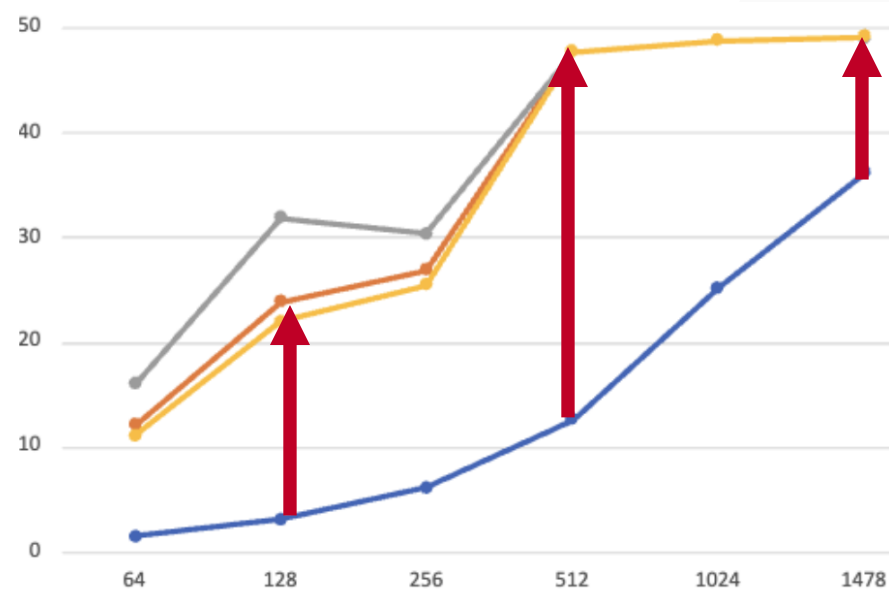
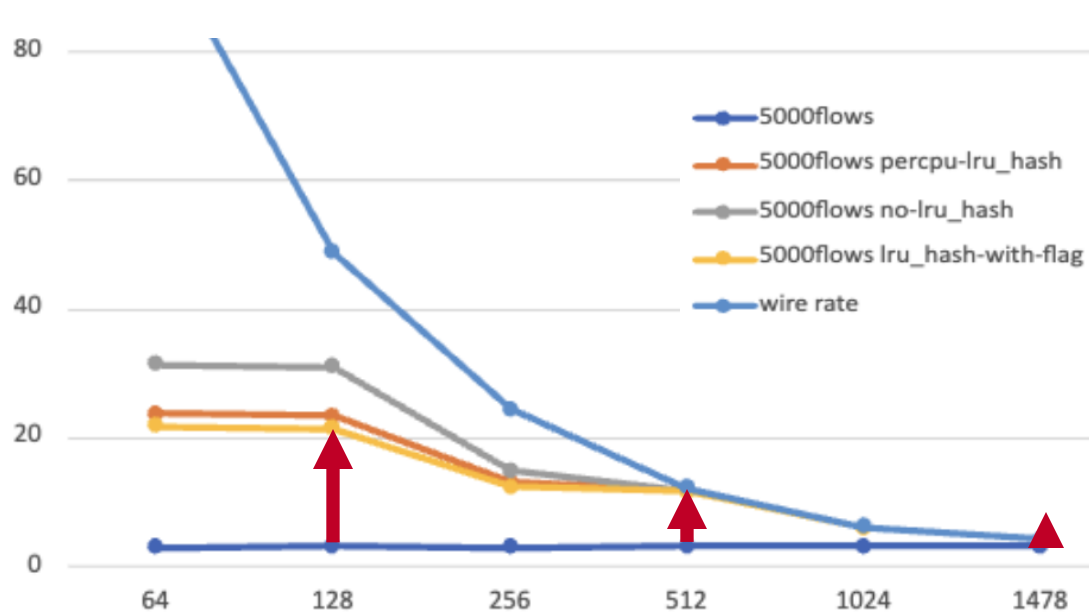


Throughputの大幅改善

解決

eBPF mapの**BPF_F_NO_COMMON_LRUフラグ**をつけることでPER CPUごとにLRUカウンタを持ち、追い出し候補を選出。(実際の追い出しは全体で使われていないか確認)

⇒ ワイヤレート限界の同時接続flow数(数万flow)までthroughputが向上



Health Check Agentの性能の大幅改善

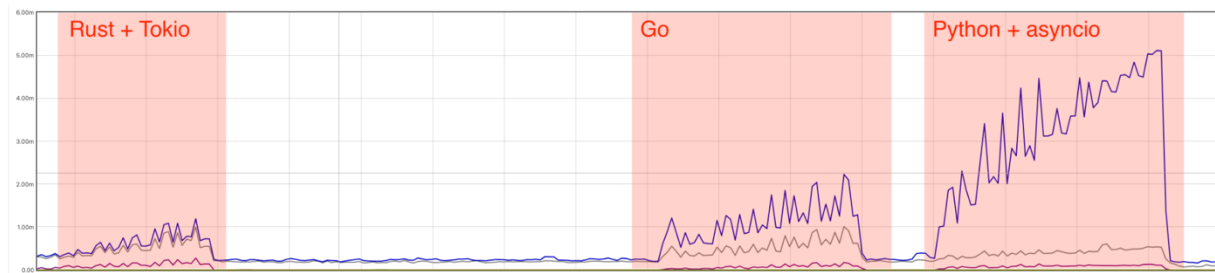
課題

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の数が増えると1つ1つのhealth checkの処理かかる時間が大幅に増加、最悪health check agentが停止。死活監視の即時反映ができない。

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の詳細な性能解析と、抜本的な性能改善の必要性があった。

CPU Usage



Result

In this benchmark, we checked how much duration does H/C probes take and when timeout starts to occur by varying the number of probes at once.

All the benchmarks are executed 3 times for each setting. And the results are average. We used this script.

In this result, the cell colored in orange means timeout in 3 seconds was 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の性能の大幅改善

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、余分な処理を削ぎ落とす。
- 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。

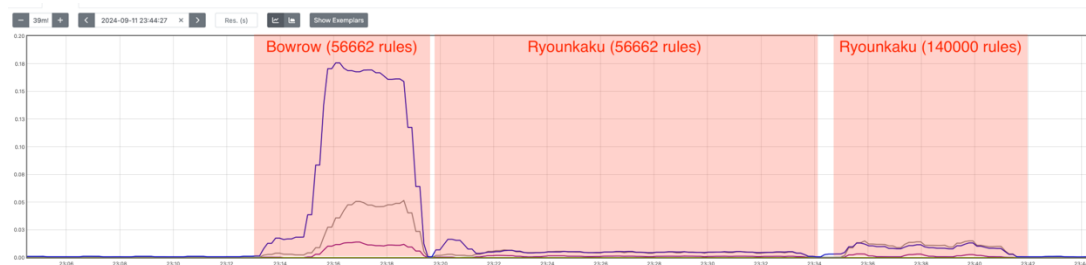
⇒CPU利用率を1/20以下に抑えられた。

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

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。
この計算部分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
2,000 backends (100VIPs, 20 ba

Loss of Idempotency:

Using IDs caused failures when trying to update a backend.
As a result, if communication with the backend failed, the system would not know if the backend was still down.

What To Do

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

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

Background:

Previously, when backends went up or down, the maglev hash table was recalculated individually for each backend. The calculation had a complexity of $O(M \log M)$ on average and $O(M^2)$ 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

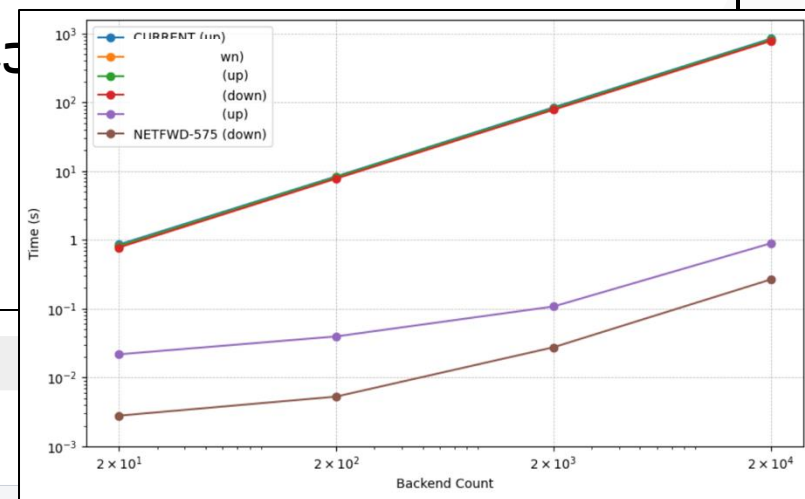
LB Hashing Table 計算の高速化

解決

Maglev Hashing Tableの計算の工夫とeBPF map上のtable構造を見直すことで
ほとんど全てのケースで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.

nlblet	nlbdp	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
v0.8.38	v0.3.3-kernel-5.14.0	858 ms	821 ms	8.4 s	8.1 s	84.5 s	80.2 s	841 s	804 s
PR-653	PR-71	817 ms	778 ms	8.2 s	7.8 s	81.9 s	78.0 s	817 s	781 s
PR-657	PR-74	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たくさんある。

- 重要度の高いユーザのサポート
- 機能追加、バグ修正
- 次なるプロダクトの開発

⇒ **作ったものは責任を持って運用する必要性。**

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
- すぐにスケール限界を迎えるシステム
- etc.

インフラとしてLB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が決定する。

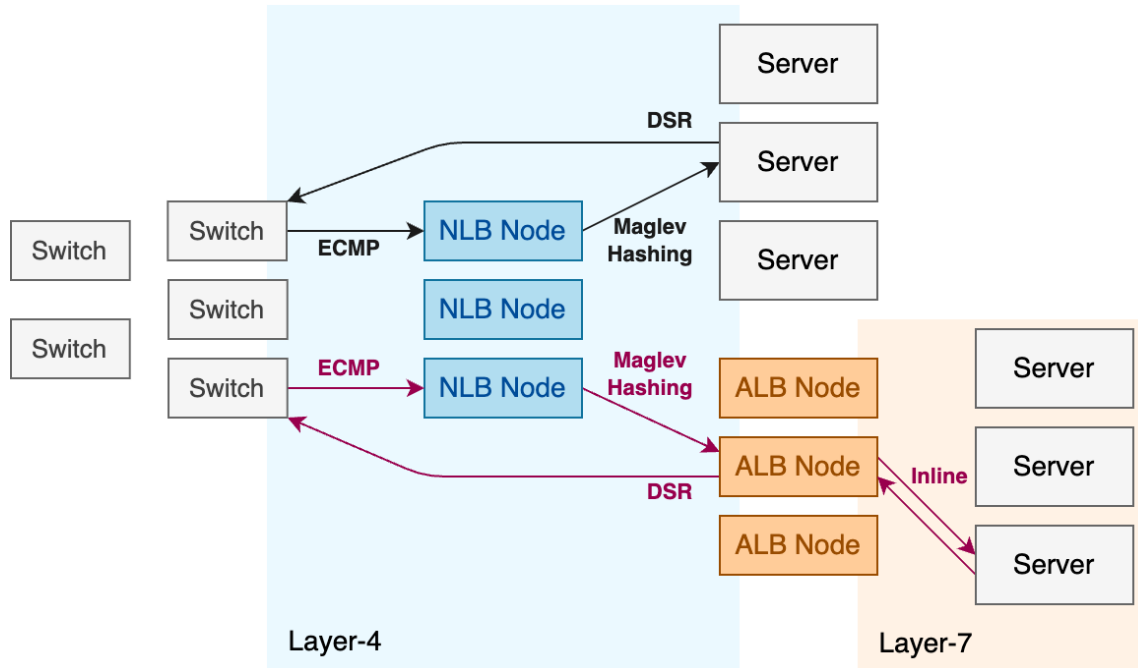
これらはエンジニアとしての前提・醍醐味

- 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
- 高速化

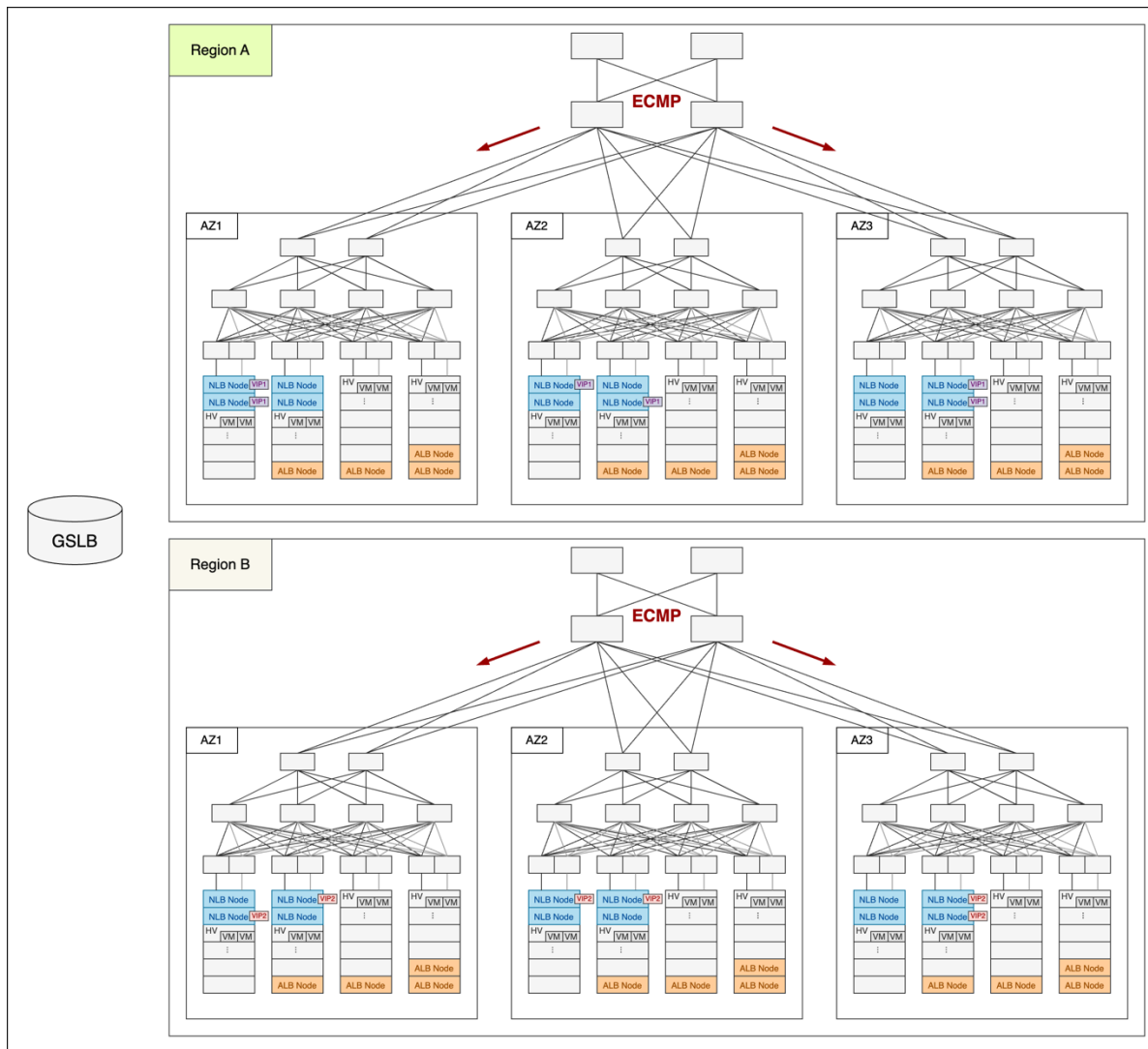
安定稼働するためのアーキテクチャを選べる重要性
選球眼のためのアーキテクチャについて学ぶことの重要性

Appendix

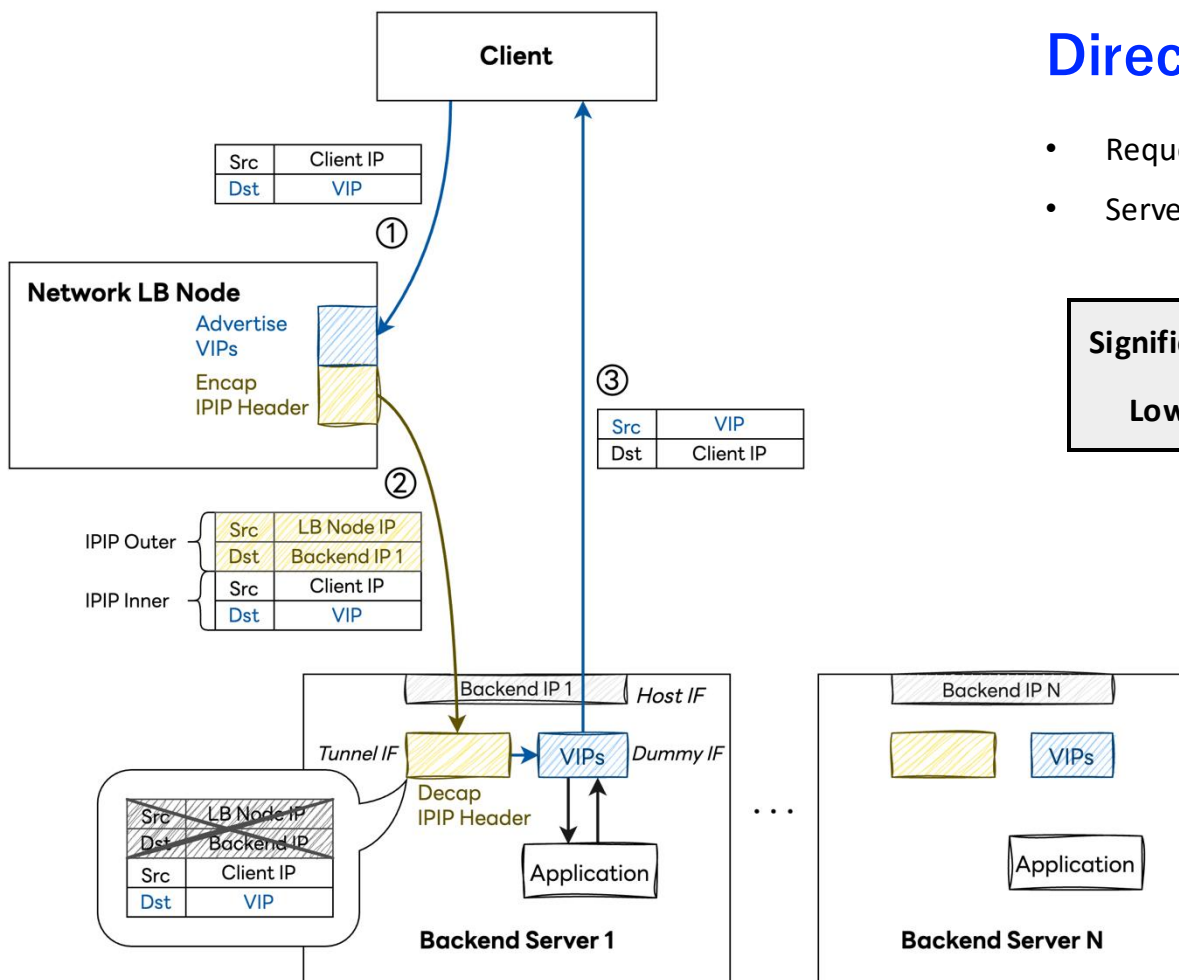
Our Network LB and Application LB



Multi Region, Multi AZ



Direct Server Return: IPIP encapsulation



Direct Server Return (DSR)

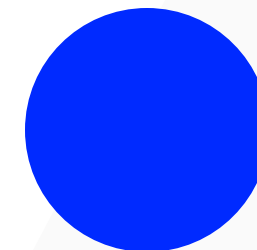
- Request goes through the LB node.
- Server responds directly to the client.

Request



<<

Response

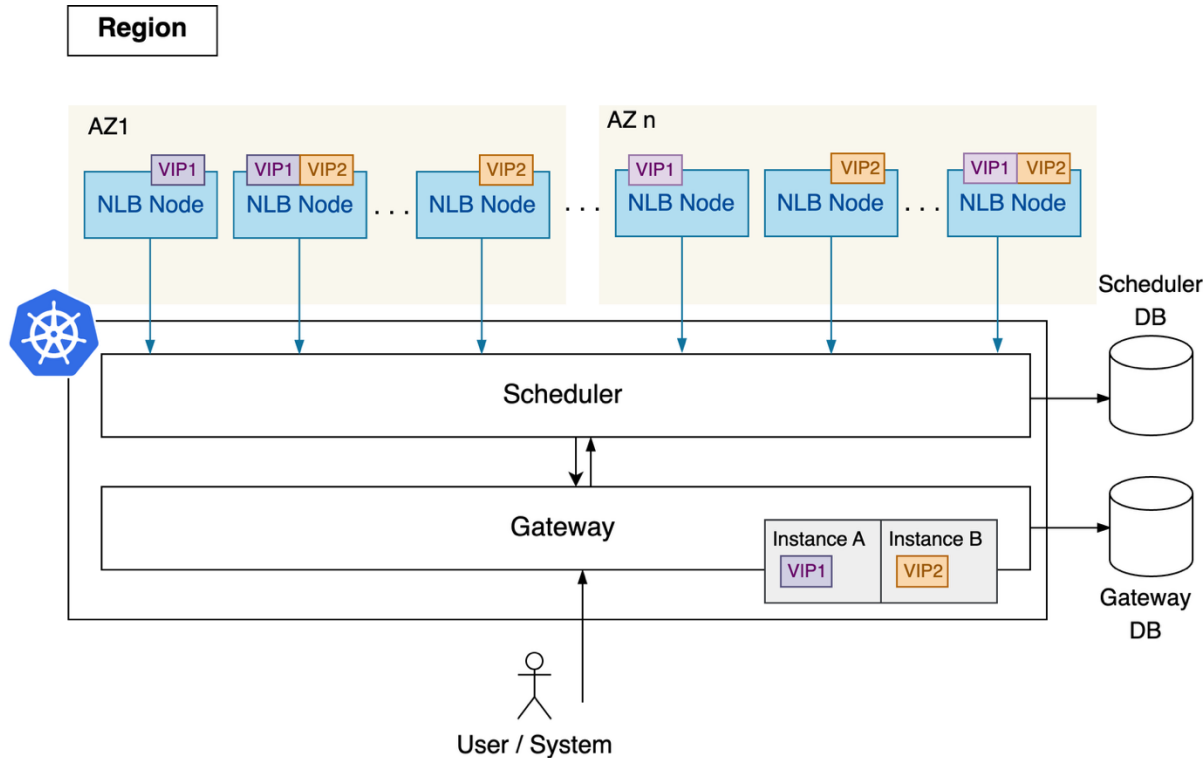


Significantly reduces LB nodes' load.

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.

LB 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: 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.

NVIDIA® Mellanox® ConnectX®-5

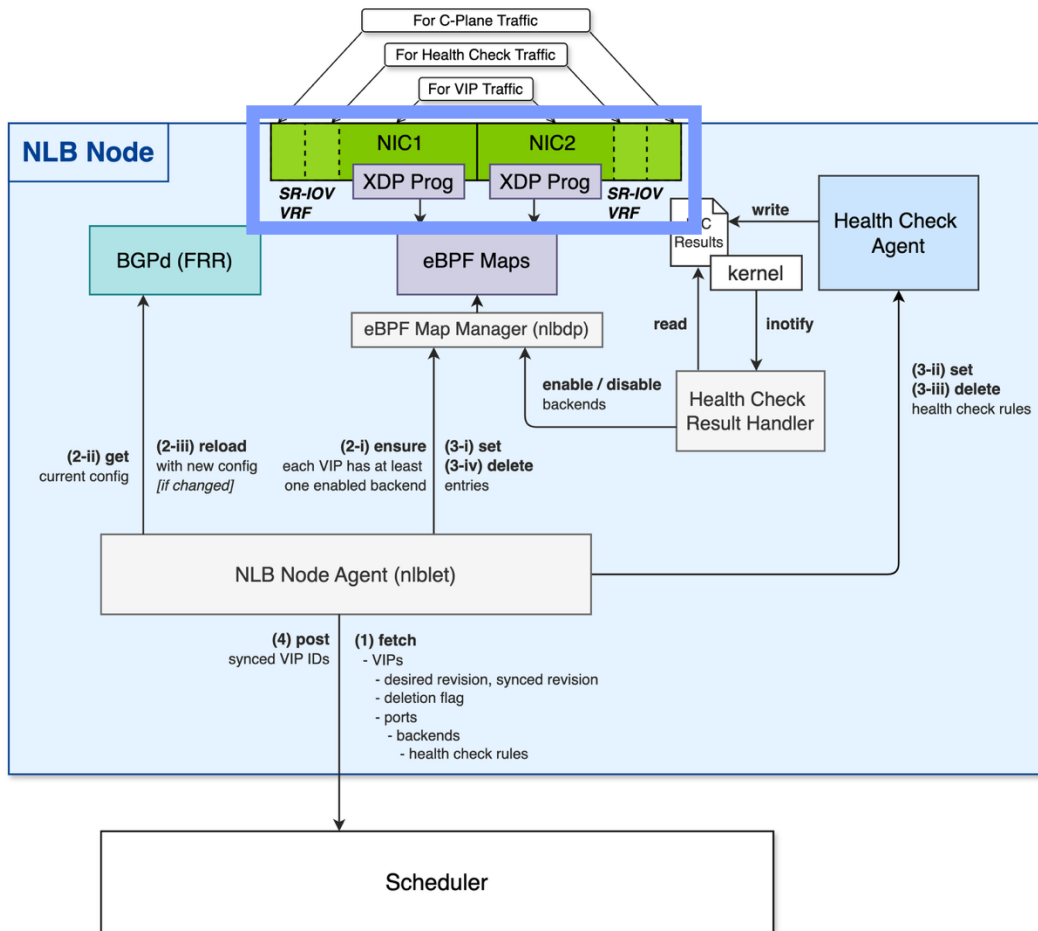
- **25BGE dual-port** (50Gbps)
- Support **XDP** in the driver.
- Support **SR-IOV**.
(for c-plane and d-plane separation)



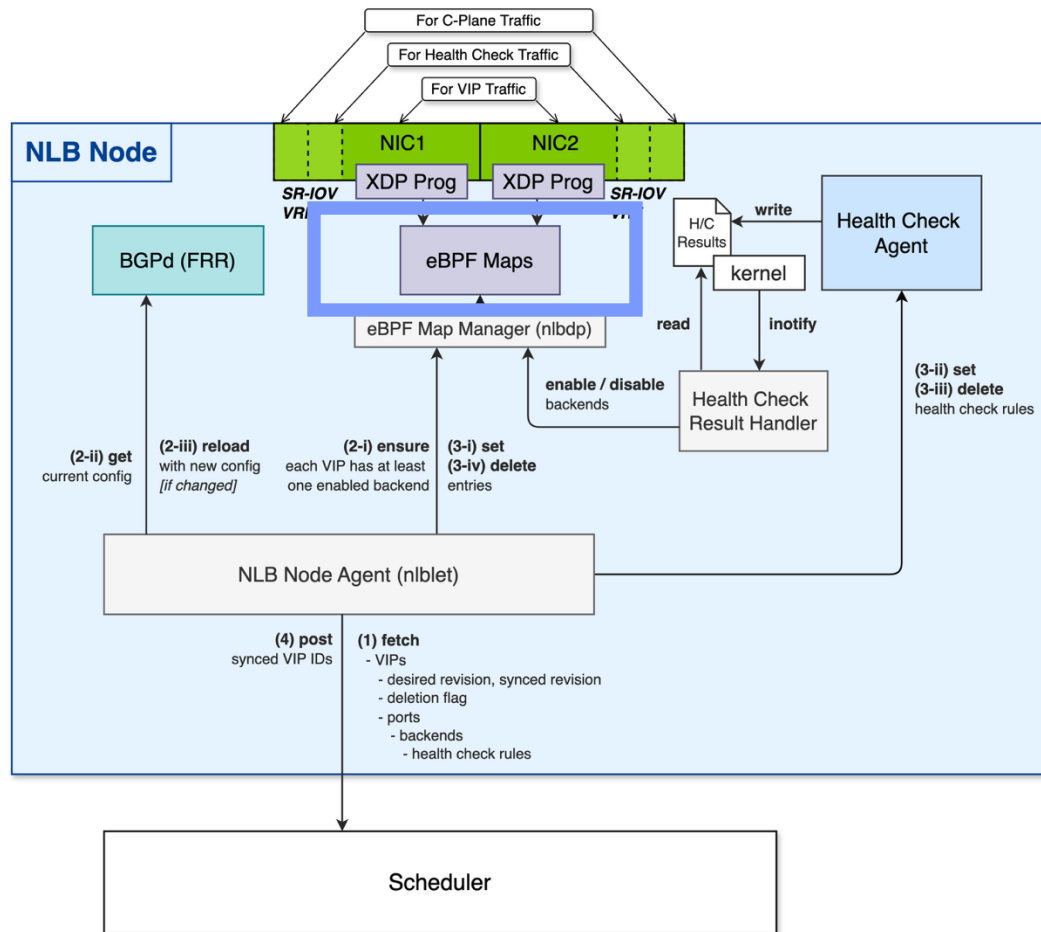
<https://www.nvidia.com/en-us/networking/ethernet/connectx-5>

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 applications, simplifying map management.

Network LB Node: VIP Advertisement

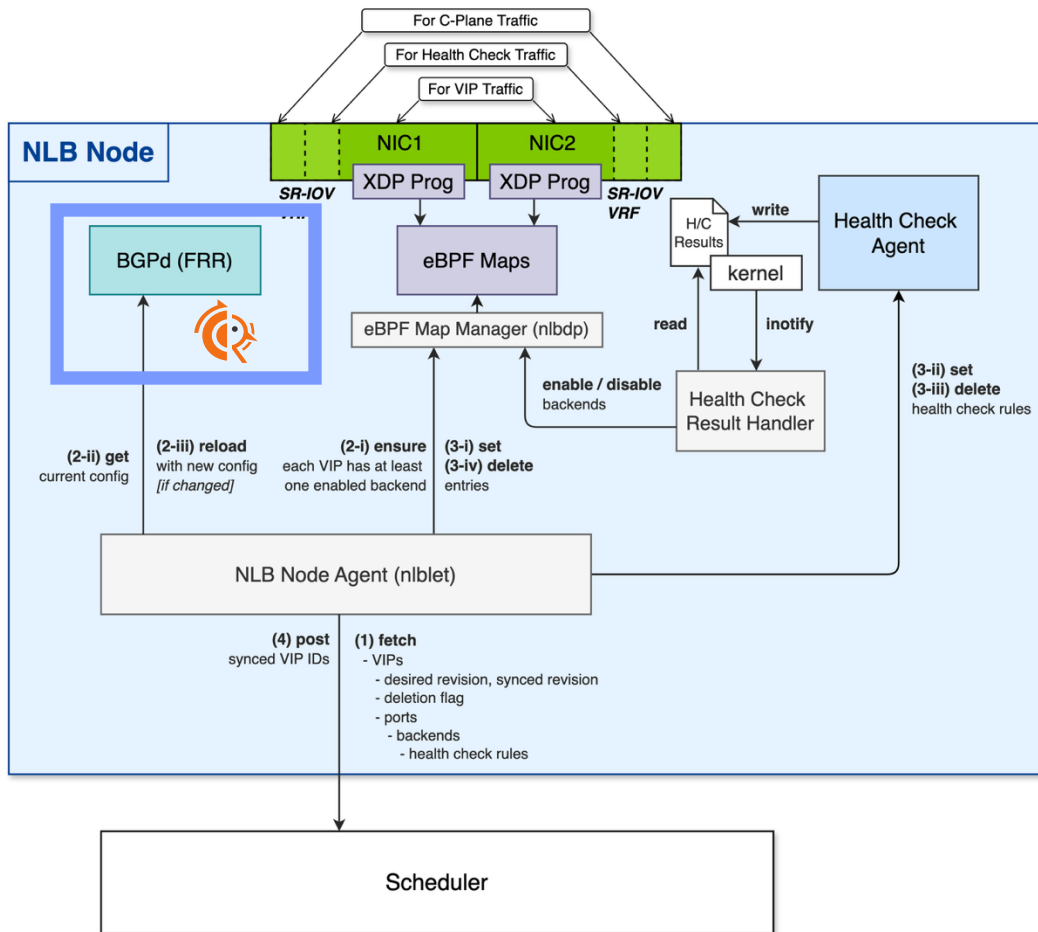
VIP advertisements

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: Health Check

Health Check Agent

Performs real-time backend monitoring.
Supports **multiple protocol** checks:

- **ICMP**, **TCP**, and **HTTP(s)**.
- Determine backend status by combining multiple check results.

(Redundant checks will be consolidated.)

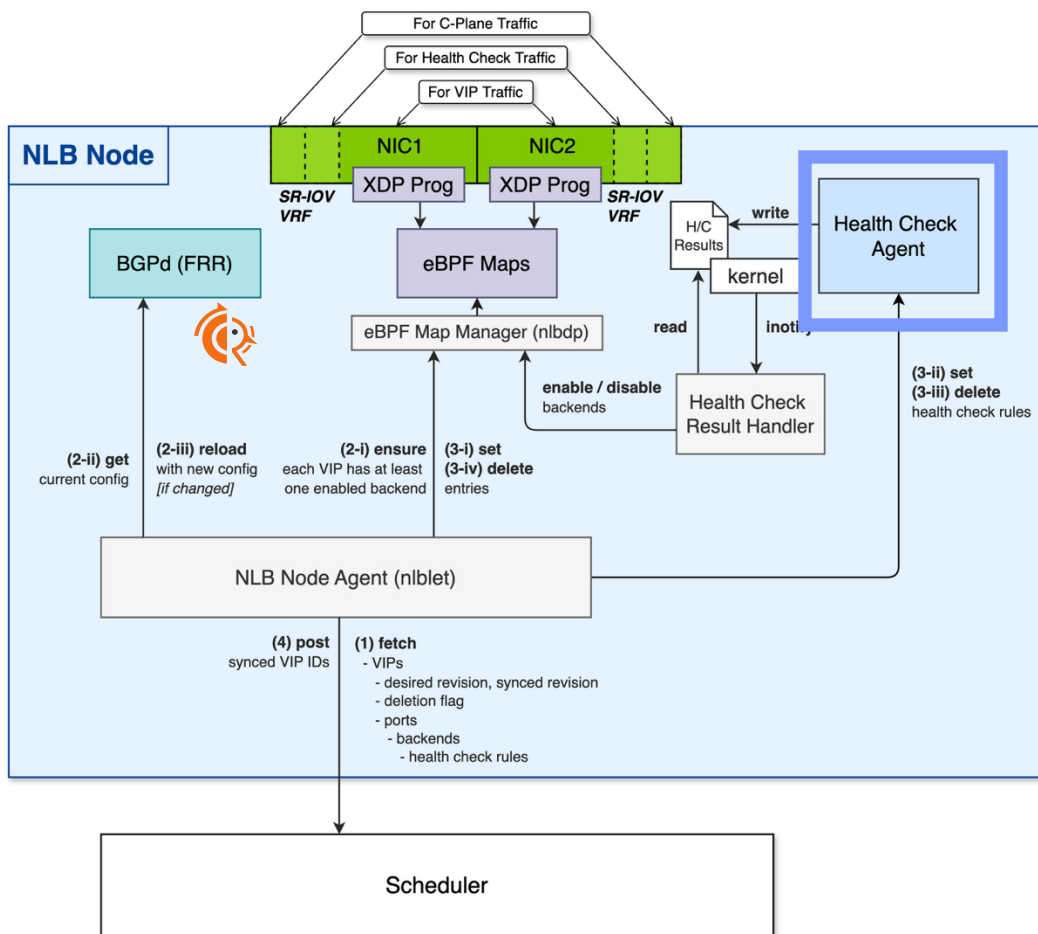
Runs checks continuously at short intervals for each VIP backend,

requiring high performance.

(We implemented the agent in Rust.)



- A separate Handler process **syncs the eBPF maps**, triggered by the file update.



Network LB Node: Reconciliation

NLB Node Agent (nlblet)

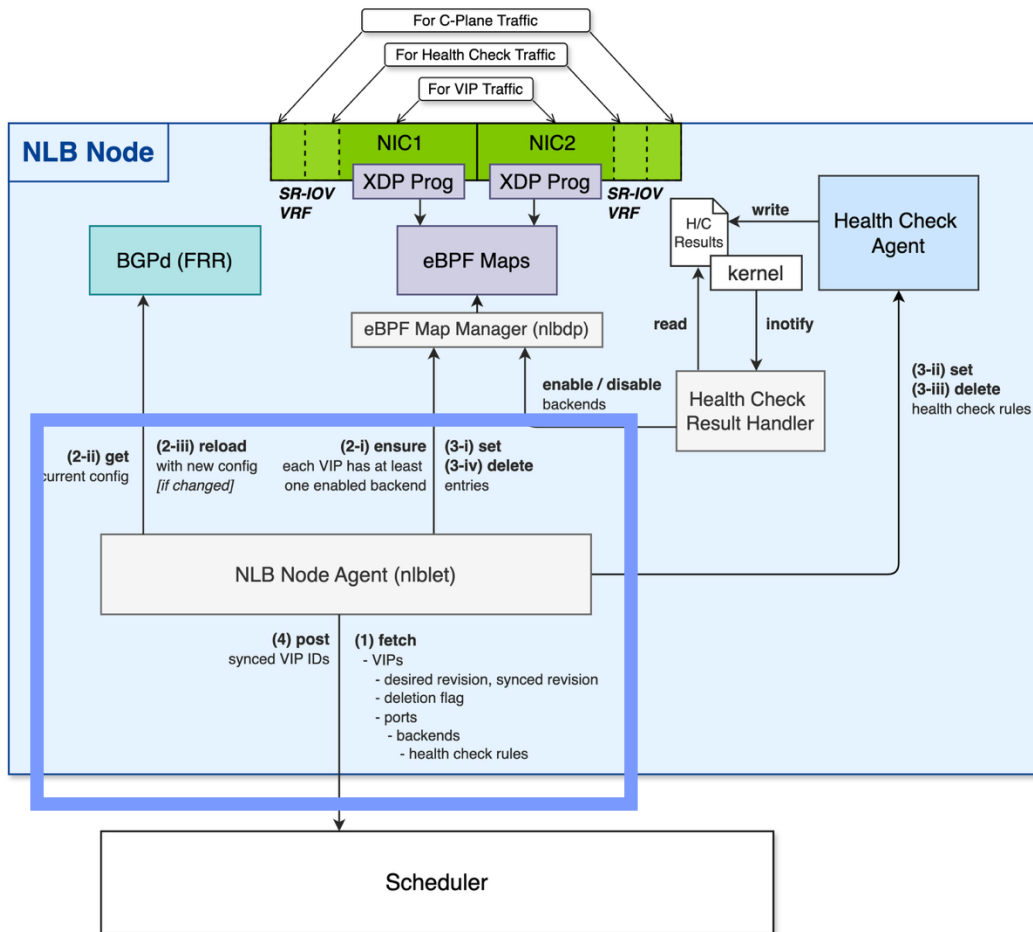
(Named after kubelet in Kubernetes.)

The 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