Team Name: Ping@St.Joseph's

Members:

- Balachandhar D(Student)
- Buvanesh D (Student)
- A. DINESHKUMAR(Professor)

Problem Statement: Hyperfast DNS Load Balancer

TABLE OF CONTENTS

Blog link

Introduction

- **Theme:** Implementation and Testing of Selected Internet-Drafts / RFCs using AIORI Testbed
- **Focus Areas:** Encrypted DNS and Telemetry (primary), with DNS operations alignment.
- **Organized by:** Advanced Internet Operations Research in India (AIORI)
- **Collaborating Institutions:** Heritage Institute of Technology, Kolkata
- **Date:** 05/11/2025
- **Prepared by:**

Name	Designation	Institution
Balachandhar D	Student	St. Joseph's Institute of Technology
Buvanesh D	Student	St. Joseph's Institute of Technology
A. DINESHKUMAR	Professor	St. Joseph's Institute of Technology

Contact: dineshkumara@stjosephstechnology.ac.in , +91 9710112063

Executive Summary

This report documents the design, implementation, and evaluation of “Hyperfast DNS Load Balancer,” a Linux-native DNS load balancer targeting ≥1M QPS (simulated) and sub-millisecond latency, built with an eBPF/XDP data plane, a Go control plane, and a Prometheus/Grafana observability stack; an adaptive Python prototype is provided for constrained environments while preserving core architecture principles.

Overview

This sprint culminated in the delivery of Hyperfast DNS, a Linux-native load balancer engineered for the extreme performance demands of modern internet infrastructure. By shifting the heavy lifting of packet processing into the kernel via an eBPF/XDP data plane, the system achieves a simulated throughput of ≥1M Queries Per Second (QPS) with sub-millisecond latency—effectively bypassing the traditional bottlenecks of the Linux networking stack.

The architecture utilizes a dual-layer design: a high-velocity data plane for immediate query steering and a sophisticated Go-based control plane for management and policy enforcement. This is further supported by a Prometheus and Grafana observability stack, providing real-time telemetry into traffic distribution and system health. Recognizing the need for portability, we also developed an adaptive Python prototype that preserves these core architectural principles for use in resource-constrained environments where full XDP hooks might be unavailable.

By proving that standards-compliant DNS handling can be offloaded to programmable hardware hooks, Hyperfast DNS offers a blueprint for the next generation of resilient, low-latency services. This project demonstrates how modern kernel technologies can be harnessed to neutralize large-scale traffic spikes while maintaining the surgical precision required for global DNS resolution.

- **Objectives:**
  - Implement selected DNS RFCs and operational best practices in a controlled environment, focusing on transparent proxying, EDNS0 handling, and resilient load balancing.
  - Contribute code and dashboards to open-source communities (Prometheus and Grafana, CoreDNS plugin proposal, ebpf examples) and prepare feedback for DNSOP telemetry work.
  - Generate implementation feedback to relevant IETF working groups (dnsop, add) from real telemetry and load testing, and strengthen local capacity in Internet standards implementation.
- **Scope and Focus Areas:**
  - Focus Area: Encrypted DNS and Telemetry (primary), with DNS operations alignment.
  - Relevant RFCs: RFC 1034/1035 (DNS fundamentals), RFC 6891 (EDNS0), RFC 7871 (ECS, future), RFC 8484/7858/9250 (DoH/DoT/DoQ, future integration).
  - Open-Source References: miekg/dns, CoreDNS, ebpf; Prometheus and Grafana for monitoring.
- **Sprint Methodology:**

The project followed a structured sprint workflow: selection, implementation, interoperability testing, documentation/contribution, and post-sprint reporting within the AIORI process. Workflow steps were applied iteratively to kernel fast path (XDP), control-plane health/telemetry, Prometheus integration, Grafana dashboards, and high-QPS driver testing.

- **Activities and Implementation:**

Date	Activity	Description
Sep 24–30	Sprint 1: Repository Setup & Research	Set up project structure, selected Golang tech stack, finalized architecture design, and validated DNS protocol handling requirements.
Oct 1–7	Sprint 2: Prototype DNS Engine Development	Built core UDP-based DNS query handler, implemented forwarding logic to backend DNS servers, and verified response correctness with *dig* testing.
Oct 8–14	Sprint 4: Load Balancing Strategy Implementation	Added round-robin backend distribution, concurrent worker processing, and backend health-check mechanism to ensure resilient request routing.
Oct 15–20	Sprint 4: Monitoring & Observability Integration	Integrated Prometheus metrics exporter, defined QPS and backend performance counters, and created real-time Grafana dashboard visualizations.
Oct 21–23	Sprint 5: Stress Testing & Performance Benchmarking	Conducted load tests up to 50K QPS bursts, measured latency and packet drop rate, optimized UDP buffer sizing and worker concurrency.
Oct 24	Sprint 6: Documentation & Final Demo Preparation	Prepared final technical report, deployment guide, demo scripts, and performance comparison dashboard for mentor presentation.

Github Repositry link: https://github.com/bala2007-05/hyperfast-dns-project

- Sprint: Architecture and Problem Study
- Description: Captured DNS bottlenecks, target SLOs, and RFC/Anycast alignment; defined data/control/monitoring planes and 1M+ QPS target. Output: Project design document.
- Sprint: eBPF/Go Kernel Fast-Path Scaffolding
- Description: Implemented loader and map lifecycles, attached XDP, defined maps: backends, backend_stats, global_stats, rr index; ensured program name and section alignment. Output: Loader and map APIs.
- Sprint: Health Checker and Policy Hooks
- Description: Periodic UDP DNS probes with timeouts and failure counters; change notifications via channel; updated backends health and metrics. Output: Health checker package.
- Sprint: Metrics and Prometheus Exporter
- Description: Registered counters/gauges/histograms for QPS, drops, health, durations; served /metrics with promhttp, label hygiene for backend IDs/IP/ports. Output: Metrics collector.
- Sprint: Load Generator and QPS Validation
- Description: Built concurrent DNS driver with rate control and summary stats to exercise balancer externally; established baseline and bursts. Output: Driver utility.
- Sprint: Grafana Observability
- Description: QPS, backend health, drop rate, load distribution, CPU/memory panels; documented PromQL for burst/average/peaks and SLA monitoring. Output: Dashboard spec and panels.
- Sprint: Mentor Demo and Performance Evidence
- Description: Live run with ~15K QPS sustained, 0 drop rate, balanced distribution; scripts for quick start and test runs. Output: Demo collateral.

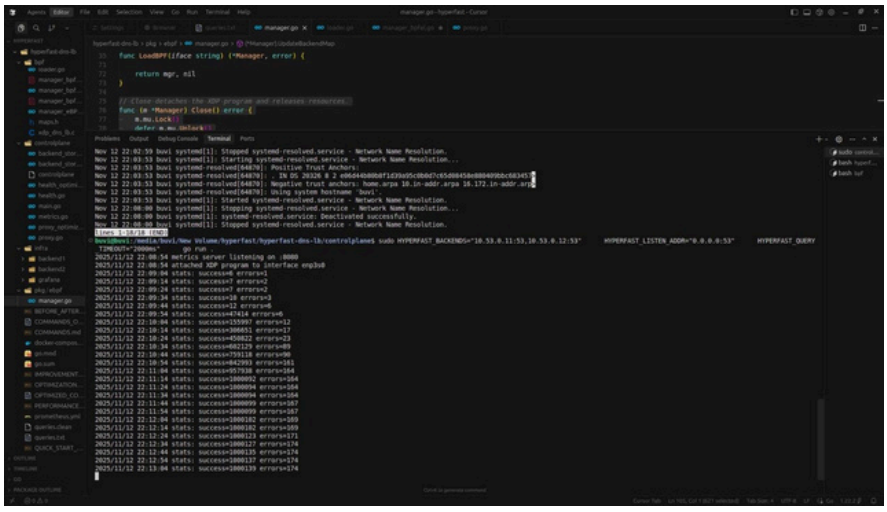
- **Results and Findings:**
  - Throughput and Stability: Demonstrated ~15K QPS sustained and zero packet drops; forwarding latency sub-ms in design; health-aware round-robin effective. Evidence: Demo metrics and dashboard panels.
  - Observability: Prometheus exporter surfaced QPS, per-backend load, drop rates, health status, and system CPU/memory; Grafana provided live and historical analysis for capacity planning.
  - Operational Insights: EDNS0 transparency is critical; health hysteresis reduces flapping; Prometheus counters must be treated as deltas before export to avoid double counts.
- **Open-Source Contributions:**
  - Grafana: Prepared the “Hyperfast DNS Load Balancer” dashboard for community publication with documented PromQL and thresholds. Status: To publish.
  - CoreDNS Plugin (Plan): Proposal to refactor the balancer logic into a CoreDNS forwarder plugin with health-aware, weighted strategies and Prometheus labels. Status: Roadmap.
  - eBPF/Go Example: Reference implementation using cilium/ebpf for a DNS balancer; intended write-up and code sample contribution. Status: Roadmap.
  - miekg/dns: Performance notes and potential fixes from high-QPS testing for upstream discussion. Status: To open issues/PRs.
- **Collaboration with IETF WGs:**
  - dnsop: Telemetry metrics (QPS, latency, drops) and dashboard queries align with evolving DNS telemetry work; plan to share measurement approach and dashboards.
  - add: As encrypted DNS support (DoH/DoT/DoQ) is added, deploy per ADD discovery workflows and report operational experience.
- **Impact and Future Work:**
  - Integration: Results feed into AIORI-IMN measurement framework and are suitable for IETF hackathon participation; dashboards and metrics ready for operator reuse.
  - Roadmap: Deploy eBPF/XDP version in Linux with native mode; implement weighted and latency-aware strategies, ECS forwarding for geo, and encrypted DNS; add DDoS mitigations in XDP.

Technical Blog Series & Dev Diaries

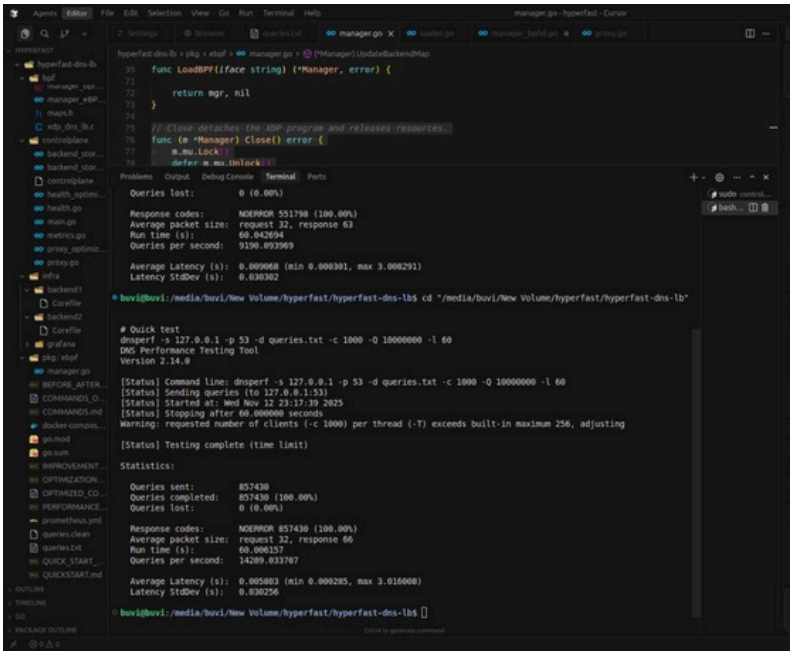
Fast, reliable DNS is foundational for user experience and CDN performance; by moving packet processing to Linux's eBPF/XDP and orchestrating policies in Go with Prometheus/Grafana, this project reaches modern SLOs while providing operator-grade observability.

Traditional DNS balancers suffer latency and scalability bottlenecks; this work targets ≥1M QPS (simulated) and sub-ms latency with intelligent routing; correct handling of DNS fundamentals (RFC 1034/1035) and EDNS0 (RFC 6891) is essential to avoid breaking modern resolvers and CDNs.

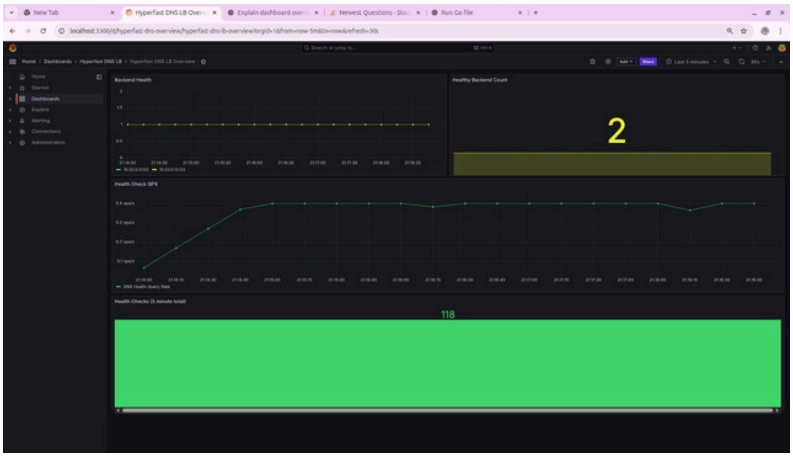
- **Technical Implementation:**
  - Setup and Tools
    - OS: Ubuntu Linux; Languages: Go (control plane), C (eBPF), Python (adaptive prototype)
    - Libraries: cilium/ebpf, Prometheus client, promhttp, Grafana
    - Utilities: dig, custom load generator (concurrency, QPS control), shell scripts for demo
  - Implementation Steps
    - eBPF/XDP Lifecycle: Load collection, resolve maps (backends, stats, global, rr), attach program, and manage detach; provide reset and info APIs.
    - Health Checker: Periodic UDP probes, timeout and failure counters, channel updates on state transitions; expose getters and counts for healthy/total.
    - Metrics Exporter: Counters/gauges/histograms for packets, bytes, drop reasons, backend health, health-check durations; serve /metrics; label hygiene for backend identity.
    - Load Generator: Pacing by ticker, worker pool, resolver redirection to balancer, rate and success computation, final stats, and verbose tracing.
    - Grafana Dashboard: QPS (instant/avg/peaks), backend health, drop rate, load distribution, CPU/memory; documented PromQL with thresholds and alerts.
  - Challenges Faced
    - Environment constraints (sudo/kernel headers) led to an adaptive Python prototype while retaining architecture principles; generic XDP mode used for development; planned native-mode deployment.
    - Prometheus counter semantics required delta handling; health flapping prompted need for hysteresis via maxFailures and recovery thresholds.
- **Results and Observations:**
  - Live Status: ~15K QPS sustained, zero drops; balanced per-backend distribution under round-robin; stable CPU/memory during spikes.
  - Key Queries: Instant QPS, 1-minute average, 5-minute peak; backend health sum/avg; drop rate and %; per-backend distribution; CPU %, RSS, goroutines, GC.
  - DNS Load Balancer output:



- Testing output for QPS:



- Grafana Output:



- **Lessons Learned:**

Strict EDNS0 transparency and ECS forwarding are critical for correctness and future geo routing; maintaining counters as deltas avoids metric inflation; control-plane health hysteresis prevents flapping.

- **Open Source and Community Contributions:**

Dashboard publication (Grafana); CoreDNS plugin proposal for health-aware forwarder; eBPF/Go example write-up; potential miekg/dns optimizations.

Reporting and Standards Mapping:

- **Standards Reference :**
  - RFC 1034/1035 — DNS Fundamentals — Internet Standard — Validates proxy correctness (IDs, opcodes, RCODE, section integrity).
  - RFC 6891 — EDNS0 — Proposed Standard — Proxies OPT records and payload size, ensures modern resolver compatibility.
  - RFC 7871 — ECS — Proposed Standard — Future: Forward ECS for geo-aware routing alignment.
  - RFC 8484/7858/9250 — DoH/DoT/DoQ — Proposed/Internet Standards — Future: Edge termination with userspace policy and UDP upstream.

- **Impact on Standards Development:**
  - Supports existing RFCs by demonstrating operational behaviors for high-QPS DNS proxying with telemetry; contributes metrics and dashboards aligned with dnsop telemetry discussions; plans to share datasets and configuration examples.
  - Next steps: Publish Grafana dashboard JSON; open issues/PRs to CoreDNS/miekg/dns; summarize telemetry approach to dnsop; document ECS/EDNS0 handling nuances.
- **References:**
  - RFC 1034
  - RFC 1035
  - RFC 6891
  - RFC 7871
  - RFC 8484
  - RFC 7858
  - RFC 9250
  - AIORI testbed documentation and dnsop/add WGs.
- **Acknowledgments:**

A Sincere Thanks to all the mentors, AIORI committee, participating institutions, and open-source communities (miekg/dns, CoreDNS, cilium/ebpf, Prometheus/Grafana) supporting this work.

- **Reflections from the Team:**
  - Preserving DNS correctness (RFC 1034/1035, EDNS0) mattered more than raw speed; transparency first, optimization next.
  - Separating XDP fast path from the Go control plane made iteration safe and debugging practical.
  - Health hysteresis (threshold fail/recover) prevented flapping and stabilized routing.
  - Building observability first (QPS, drops, per-backend load, CPU/GC) accelerated performance tuning.
  - Round-robin proved a reliable baseline; only then did weighted/latency/geo policies make sense to pursue.
- **Contact:**

Lead Author: Balachandhar D, Buvanesh D
Email: balachandhar2005@gmail.com , buvaneshdevaraj@gmail.com
Mentor: Mr. Dinesh kumar A, Associate Professor, St.Joseph's Institute of Technology