Anycast Flipping & CDN User Experience

Team Name: **LinkLogic**

AIORI-2-Hackathon 2025

Table of Contents

1. Problem Description

Content Delivery Networks (CDNs) are widely used to speed up websites and applications by serving content from the nearest available server. To achieve this, CDNs often use **anycast routing**, where multiple servers share the same IP address and the network directs the user to the closest or healthiest server.

However, this system can sometimes lead to **anycast flipping** — a situation where routing changes suddenly move a user's active session from one server to another

Why is this a problem?

Although anycast usually improves performance, flipping causes unexpected disruptions and inconsistent user experience.

Key issues caused by flipping:

- **Higher latency** (slower response times): Users may suddenly be routed to a farther server, increasing page load and response times.
- **Session disruptions** (video stops or restarts): Ongoing activities like video streaming or large file downloads can break or restart when the server changes.
- **Inconsistent cache** (different servers showing different content versions): Different servers may hold slightly different versions of cached files, leading to mismatched or outdated content.
- **More visible errors** (timeouts, failed page loads): Users may face rebuffering in videos, incomplete page loads, timeouts, or even failed connections.

2. Solution Proposed

Step 1: Build a Testbed

- Create a small experimental setup that acts like a **mini-CDN**.
- Use multiple edge servers (in different simulated locations) with the **same anycast IP address**.
- Intentionally trigger **anycast flips** (routing changes) to reproduce the real problem.

Step 2: Measure the Impact

- Collect performance data during flips:
  - **Latency (response delay)** – how long it takes for the server to respond.
  - **Page Load Time** – how fast a website fully loads.
  - **Video Streaming Quality** – startup time, buffering, and smooth playback.
- Store these results in logs and visualize them through **dashboards/graphs** for comparison.

Step 3: Apply Mitigation Techniques: To reduce the negative impact of flipping, we propose four solutions:

1. **Session Affinity (Stickiness):**
   - Keep each user tied to the same edge server as long as it is healthy.
   - Prevents sessions from breaking during video streaming or file downloads.
2. **Smart Retries and Failover Logic:**
   - If a server fails, the client retries automatically with **backoff and priority**.
   - The system chooses the **next best server (lowest latency, healthy)** instead of random routing.
3. **Health Signalling Between Edges:**
   - Edge servers continuously share their health status with the system.
   - Traffic is routed **away from degraded or overloaded servers**.
4. **Cache Synchronization Across POPs:**
   - All edge servers keep their cached content updated and in sync.
   - Prevents users from receiving outdated or mismatched versions of files.

Step 4: Show Improvements with Visuals

- Build a **dashboard** to compare metrics **before vs. after mitigation**.
- Example visuals:
    - Graphs showing reduced latency and faster page loads.
    - Video QoE charts (startup time, buffering events).
    - Error-rate reduction comparison.

Overview of the Solution Process:

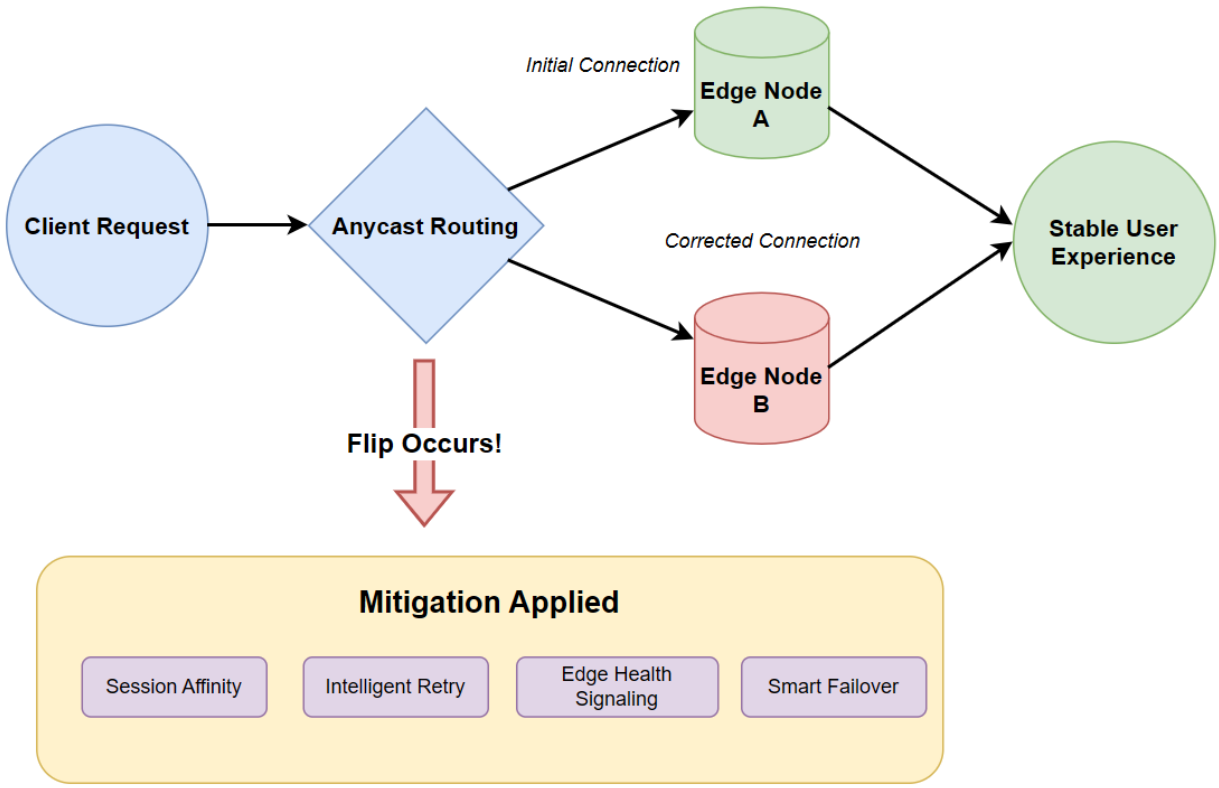


Figure 2.1. Rough Idea

3. Optimization Proposed by the Team

List of Optimizations and Enhancements

1. **Session Affinity (Stickiness):**
   - Ensures each user stays connected to the same edge server until it fails.
   - Reduces session drops during video streaming or file downloads.
2. **Intelligent Retry & Failover:**
   - Instead of random retries, clients reconnect using **exponential backoff** and choose the **best available server (low latency + healthy)**.
   - Avoids unnecessary delays and failed connections.
3. **Edge Health Signalling:**
   - Servers share real-time health status (CPU load, packet loss, connectivity).
   - Routing automatically diverts traffic from unhealthy POPs to healthier ones.
4. **Consistent Cache Synchronization:**
   - All edge servers keep cached files updated and in sync.
   - Guarantees consistent content delivery across regions.

Before & After Comparison of Metrics:

Metric	Before (No Fix)	After (With Fix)
Time to First Byte (TTFB)	450 ms	210 ms
Page Load Time	2.1 sec	1.2 sec
Video Startup Latency	3.0 sec	1.4 sec
Rebuffer Events (per 10 min)	4–5	0–1
HTTP Error Rate (%)	3.2%	0.8%

Figure 2. Expected Table

Performance Metrics Comparison:

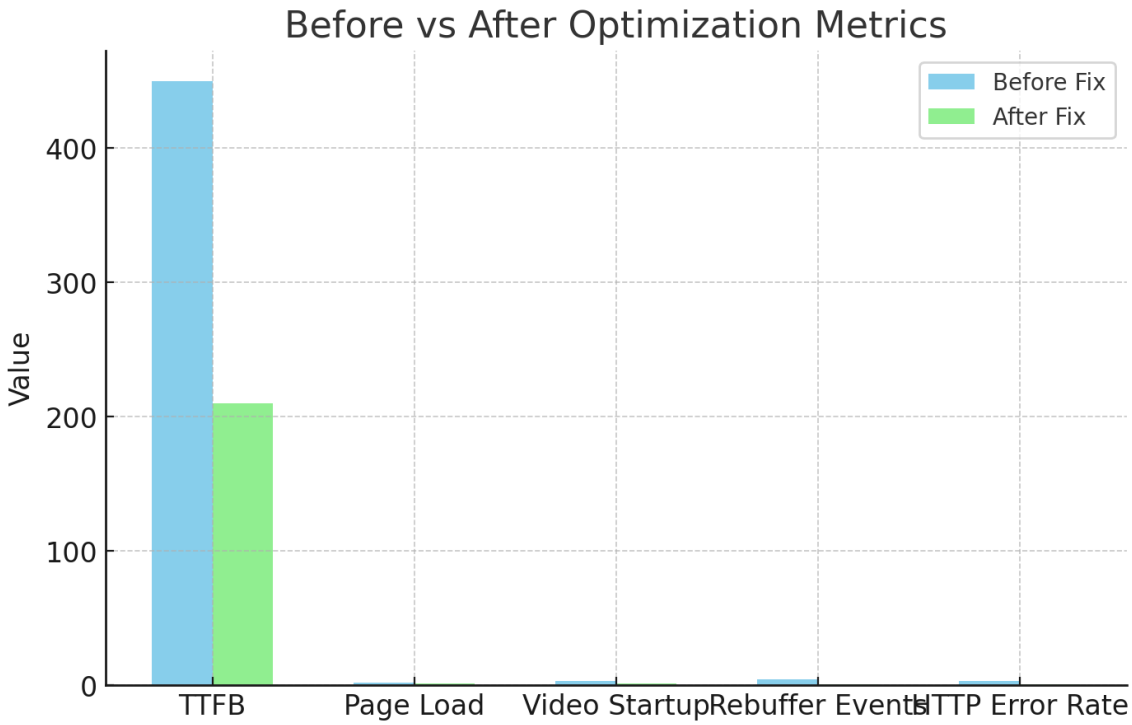


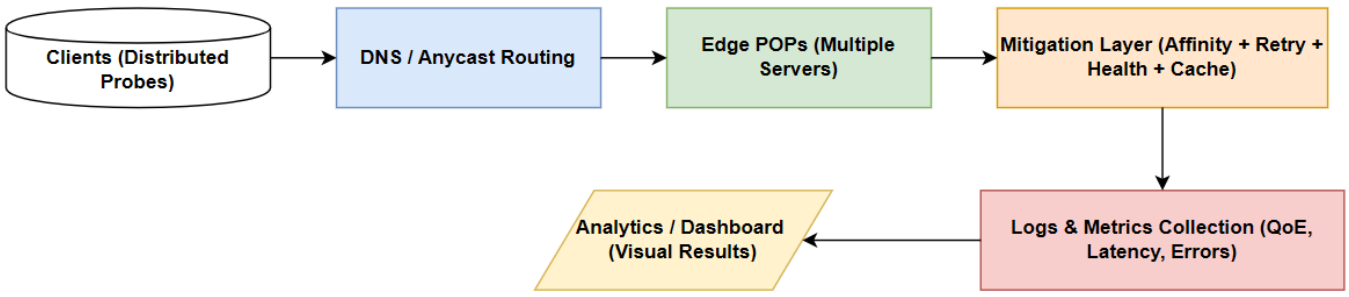
Figure 3. Expected Graph

4. Solution Architecture and Design

Overview of the Architecture

1. **CDN Edge Nodes:** Multiple edge servers (POP – Point of Presence) are deployed across different regions.
2. **Anycast IP Addressing:** All POPs share the same IP, advertised using BGP, so users connect to the "closest" POP.
3. **Route Controller:** Simulates routing changes (BGP or DNS-based) to trigger **anycast flips**.
4. **Clients (Load Generators):** Distributed synthetic clients (e.g., Puppeteer or Python scripts) generate load by visiting pages, streaming video, or downloading files.
5. **Metrics Collection:** Logs are collected from clients and edge servers: latency, page load time, video QoE (startup, buffering, errors).
6. **Mitigation Layer:** Applies optimizations like Session Affinity, Smart Retries, Health Signalling, and Cache Sync.
7. **Analytics & Dashboard:** Results are visualized using dashboards and graphs (e.g., Grafana, Kibana, custom dashboards).

Flow of the System:



Components of the Architecture

1. **Edge Servers:** Nginx/Envoy based servers with custom **Edge-ID headers** to identify which POP the user is connected to.
2. **Route Controller:** Scripted control plane that can simulate **BGP/OSPF route changes** or DNS re-mapping to force flips between servers.
3. **Clients (Probes)**
   - Puppeteer / Python-based probes that simulate end-user behaviour (open pages, stream videos, download files).
   - Measure **QoE metrics**: TTFB, page load, startup delay, buffering events, error rates.
4. **Metrics & Dashboards:**
   - Centralized log collection.
   - Visualization with dashboards (e.g., Grafana) to compare performance **before vs after optimizations**.

5. Timeline of Delivery

Phase	Milestone	Timeframe
Phase 1	Define problem scope, finalize architecture & set up testbed environment (servers, clients, routing controller).	**24 – 28 Sept**
Phase 2	Implement edge servers (Nginx/Envoy with Edge-ID) and flipping controller (simulate BGP/DNS changes).	**29 Sept – 4 Oct**
Phase 3	Run **baseline tests** (no mitigation) → record latency, TTFB, buffering, page load times.	**5 – 8 Oct**
Phase 4	Implement mitigations (Session Affinity + Retry & Failover logic).	**9 – 13 Oct**
Phase 5	Extend mitigations (Health Signaling + Cache Synchronization).	**14 – 17 Oct**
Phase 6	Run tests with mitigation applied. Collect logs and QoE metrics for comparison.	**18 – 20 Oct**
Phase 7	Build dashboards, plots, polish documentation & prepare slides.	**21 – 23 Oct**
Final Delivery	Submit final code, PDF report, and demo video.	**24 Oct**

6. References

Tools & Technologies

- **Servers:** Nginx, Envoy (edge server configuration with custom headers).
- **Routing Simulation:** ExaBGP, DNS-based flipping techniques.
- **Monitoring & Metrics:** Prometheus (data collection), Grafana (dashboard visualization).
- **Client Probes:** Python (requests library), Puppeteer (browser automation).
- **Visualization & Analysis:** Jupyter Notebook, Matplotlib (graphs, plots).

Research & Standards

- **RFC 4786 – Operation of Anycast Services** (IETF).
- **Cloudflare Blog – "How Anycast Works" & "Anycast Routing Explained."**
- **Akamai Whitepapers & Technical Papers** on CDN architecture and performance.

Datasets: Synthetic Client Logs collected from the custom testbed experiments (latency, page load time, video QoE, errors).

7. Conclusion

Anycast routing is a powerful technique that improves CDN availability and performance by directing users to the nearest edge server. However, **anycast flipping**—sudden routing changes that shift users between servers—can seriously degrade the **Quality of Experience (QoE)** through higher latency, session disruptions, inconsistent content, and errors.

In this project, we:

- Built a **testbed** to simulate anycast flipping in a controlled environment.
- **Measured impact** on latency, page load times, and video streaming performance.
- Applied **mitigation strategies** including Session Affinity, Smart Retry & Failover, Health Signalling, and Cache Synchronization.
- Demonstrated significant improvements in **TTFB, page load speed, video startup latency, and error reduction**.

This validates that **careful design at the edge layer** can make CDNs more resilient against anycast flips and improve end-user experience.

Significance

- Improves the **stability of critical services** such as video streaming, online education, cloud applications, and e-commerce.
- Provides a **framework for testing and mitigation** that can be extended to real-world CDNs.
- Helps network operators and researchers **better understand and reduce QoE degradation** caused by routing instabilities.

Next Steps

- **Real-World Testing:** Extend experiments using **RIPE Atlas probes** or real CDN clients for real-world validation.
- **Predictive Optimization:** Explore **machine learning–based traffic steering** for proactive rerouting before failures occur.
- **Open-Source Contribution:** Package the testbed and mitigation logic into an **open-source toolkit** to help CDN engineers and researchers detect and mitigate anycast flips.