A screenshot of a computer program

AI-generated content may be incorrect.Luồng

**✅ 1. Xác định rõ chỉ số cần đo (Metrics)**

Bạn nên chọn các chỉ số phản ánh cả hiệu năng và độ tin cậy:

**📊 Hiệu năng:**

* **Throughput (TPS)**: số lượng giao dịch liên chuỗi/giây
* **Latency**: độ trễ từ SC → RC → DC
* **Load distribution**: mức độ phân phối tải trên các RC
* **Gas usage (on-chain)**: chi phí gas cho xác minh zk-proof

**🔐 Độ tin cậy / Khả năng phục hồi:**

* **Tỉ lệ thành công khi một RC bị down**
* **Tỉ lệ lỗi xác minh zk-proof**
* **Khả năng thích ứng của thuật toán cân bằng tải**

A screenshot of a computer

AI-generated content may be incorrect.

**✅ 1. Transaction Duration Breakdown per Step (MRC vs Single Relay)**

* **Mô tả:** Biểu đồ đường hoặc cột thể hiện **thời gian trung bình cho từng bước** trong pipeline (SC → Relay → DC).
* **Metrics đo:** tx\_SC, prover\_SC\_RC, relay\_commit, prover\_RC\_DC, tx\_DC.

**✅ 2. Total End-to-End Latency per Transaction**

* **Mô tả:** Biểu đồ đường thể hiện **tổng thời gian xử lý** của mỗi giao dịch qua SC1/SC2 vs MRC vs Single Relay.
* **Metrics đo:** Tổng thời gian từ B1 đến B7 của mỗi giao dịch.
* **Lưu ý:** Vẽ cho 50–100 giao dịch để nhìn xu hướng.

**✅ 3. Load Distribution Across Relay Chains**

* **Mô tả:** Biểu đồ cột thể hiện **tổng số giao dịch** được gửi tới từng relay trong MRC.
* **Metrics đo:** Relay ID → Số lượng giao dịch xử lý.
* **Ý nghĩa:** Cho thấy **thuật toán load balancing** hoạt động hiệu quả ra sao.

**✅ 4. Relay Chain Processing Time (per Relay)**

* **Mô tả:** Biểu đồ cột/đường thể hiện **thời gian xử lý trung bình tại từng relay**.
* **Metrics đo:** B4\_RC\_dong\_block + B5\_RC\_send\_ctx cho từng relay.
* **So sánh:** Giữa các relay hoặc giữa MRC và Single.

**✅ 5. Number of Transactions per Second (TPS)**

* **Mô tả:** Biểu đồ đường thể hiện **TPS theo thời gian** trong MRC vs Single.
* **Cách đo:** Ghi log timestamp của mỗi tx, tính TPS bằng sliding window.

**✅ 6. Variance / Boxplot of Transaction Duration**

* **Mô tả:** Biểu đồ boxplot thể hiện **sự dao động thời gian** giữa các bước hoặc toàn bộ tx.
* **So sánh:** SC1 vs SC2 vs Single.

**✅ 7. Relay Selection Efficiency**

* **Mô tả:** Nếu bạn dùng thuật toán chọn relay (Round-Robin, Least Load…), hãy biểu diễn:
  + **Số lượng tx mỗi relay**
  + **Tổng thời gian từng relay xử lý**
* **Mục tiêu:** Cho thấy thuật toán phân phối giao dịch có hiệu quả không.

**📌 Bonus: Bạn cũng có thể đo thêm:**

* % giao dịch trễ (>x giây)
* Tỷ lệ phân phối giao dịch giữa các SC → RC
* So sánh hiệu suất giữa thuật toán chọn relay khác nhau (RR vs LL)

CÁCH ĐO ADAPTIVE PROPERTIES

**1. securityLevel (Mức độ bảo mật)**

**Hàm: calculateAdaptiveSecurityLevel(relay)**

**Dựa vào:**

* failureRate: tỷ lệ thất bại tổng thể.
* consecutiveFailures: số lượng thất bại liên tiếp.
* recentFailures: số lần thất bại trong vòng 1 giờ qua.

**Cách tính:**

* failureRate = failureHistory.length / (successHistory.length + failureHistory.length)
* Áp dụng hệ số điểm dựa trên:
  + Ngưỡng:
    - <= 5% → tốt.
    - > 15% → trung bình.
    - > 30% → yếu.
  + Nếu có ≥ 3 lần thất bại liên tiếp, giảm thêm điểm.
  + Nếu có nhiều lỗi gần đây, giảm điểm nữa.
* **Mapping:**
  + score >= 0.8 → High
  + 0.5 <= score < 0.8 → Medium
  + < 0.5 → Low

**⚡ 2. processingSpeed (Tốc độ xử lý)**

**Hàm: calculateAdaptiveProcessingSpeed(relay)**

**Dựa vào:**

* avgLatency: độ trễ trung bình (ms).
* latencyVariance: độ biến thiên độ trễ.
* throughput: số lượng giao dịch thành công trong 1 phút gần nhất.

**Cách tính:**

* avgLatency = mean(latencyHistory)
* latencyVariance = std\_dev(latencyHistory)
* throughput = số lượng success trong 60 giây
* **Điểm hiệu suất (performanceScore) bị ảnh hưởng bởi:**
  + Latency (nhỏ thì tốt).
  + Latency variance (ổn định thì tốt).
  + Throughput cao → cộng điểm.
* **Mapping:**
  + score >= 0.8 → Fast
  + 0.5 <= score < 0.8 → Normal
  + < 0.5 → Slow

**⛽ 3. costTier (Chi phí giao dịch)**

**Hàm: calculateAdaptiveCostTier(relay)**

**Dựa vào:**

* avgGasPrice: giá gas trung bình.
* gasPriceTrend: xu hướng giá gas (tăng hay giảm).
* costEfficiency: hiệu quả chi phí (ngược với giá gas).

**Cách tính:**

* avgGasPrice = mean(gasPriceHistory)
* gasPriceTrend = (recent\_avg - older\_avg) / older\_avg
* Nếu giá gas đang giảm mạnh (> 10%) → cộng điểm.
* Nếu tăng > 10% → trừ điểm.
* **Mapping:**
  + score >= 0.8 → Economy
  + 0.5 <= score < 0.8 → Standard
  + < 0.5 → Premium

**🔥 4. maxGasLimit (Giới hạn gas tối đa)**

**Hàm: calculateAdaptiveMaxGasLimit(relay)**

**Dựa vào:**

* currentGasUsage: lượng gas đang sử dụng (ước lượng từ load).
* maxObservedGas: gas nhiều nhất từng ghi nhận.
* performanceScore: hiệu suất thấp thì giới hạn gas cũng thấp.

**Cách tính:**

* capacityUtilization = currentGasUsage / maxObservedGas
* Nếu:
  + <= 30% → tăng gas.
  + 50–80% → giữ nguyên.
  + > 80% → giảm gas.
* Nếu performance thấp (< 0.5) → giảm giới hạn gas.

**Kết quả:**

* maxGasLimit = clamp(1\_000\_000, 50\_000\_000, value)

**📊 5. Dữ liệu lịch sử (updateHistoricalData)**

* latency, gasPrice, load được cập nhật mỗi vòng theo thời gian.
* Lưu lại trong:
  + latencyHistory
  + gasPriceHistory
  + maxObservedGas, currentGasUsage

**✅ 6. Theo dõi giao dịch (trackTransactionResult)**

Khi một giao dịch thành công hay thất bại:

* Ghi thời gian vào successHistory hoặc failureHistory.
* Tăng consecutiveFailures nếu thất bại.
* Cập nhật lastFailureTime.
* Dọn các record cũ (> 24h).

**Chi tiết SECURITY ADAPTIVE**

**🧠 Bối cảnh chung:**

securityLevel được dùng để đánh giá độ tin cậy và an toàn của một relay chain dựa trên dữ liệu lịch sử về giao dịch (thành công/thất bại).

**🔢 1. Tính failureRate – tỷ lệ thất bại**

js

CopyEdit

const totalAttempts = metrics.successHistory.length + metrics.failureHistory.length;

metrics.failureRate = metrics.failureHistory.length / totalAttempts;

**Ý nghĩa:** nếu có 100 giao dịch, 10 thất bại → failureRate = 0.1 (10%).

⚠️ Nếu tổng số lần thử (totalAttempts) < MIN\_SAMPLES (mặc định là 5), ta **trả về mặc định "Medium"** vì chưa đủ dữ liệu:

js

CopyEdit

if (totalAttempts < ADAPTIVE\_CONFIG.LEARNING.MIN\_SAMPLES) {

return 'Medium';

}

**🧮 2. Khởi tạo điểm securityScore = 1.0**

js

CopyEdit

let securityScore = 1.0;

Chúng ta sẽ giảm dần điểm này theo từng yếu tố rủi ro.

**🧷 3. Giảm điểm theo failureRate**

js

CopyEdit

if (metrics.failureRate <= 0.05) {

securityScore \*= 1.0; // Giữ nguyên

} else if (metrics.failureRate <= 0.15) {

securityScore \*= 0.8; // Giảm nhẹ

} else {

securityScore \*= 0.5; // Giảm mạnh

}

| **failureRate (%)** | **Hành động** | **securityScore** |
| --- | --- | --- |
| ≤ 5% | Giữ nguyên | ×1.0 |
| ≤ 15% | Giảm nhẹ | ×0.8 |
| > 15% | Giảm mạnh | ×0.5 |

**🔁 4. Giảm điểm theo số lần thất bại liên tiếp (consecutiveFailures)**

js

CopyEdit

if (metrics.consecutiveFailures >= 3) {

securityScore \*= 0.3;

}

Nếu relay gặp **3 lần thất bại liên tục trở lên**, đây có thể là dấu hiệu relay không ổn định → giảm mạnh bảo mật.

**⏱ 5. Giảm điểm nếu có lỗi gần đây (trong vòng 1 giờ)**

js

CopyEdit

const oneHourAgo = Date.now() - 3600000;

const recentFailures = metrics.failureHistory.filter(time => time > oneHourAgo).length;

if (recentFailures > 0) {

securityScore \*= (1 - recentFailures \* 0.1);

}

* Mỗi lần thất bại gần đây làm giảm 10% điểm.
* Nếu có 2 thất bại trong 1h gần nhất → điểm nhân với 0.8.

**✅ 6. Ràng buộc điểm nằm trong khoảng [0.1, 1.0]**

js

CopyEdit

metrics.securityScore = Math.max(0.1, Math.min(1.0, securityScore));

**📊 7. Mapping securityScore → securityLevel**

js

CopyEdit

if (metrics.securityScore >= 0.8) return 'High';

if (metrics.securityScore >= 0.5) return 'Medium';

return 'Low';

| **securityScore** | **securityLevel** |
| --- | --- |
| ≥ 0.8 | High |
| ≥ 0.5 và < 0.8 | Medium |
| < 0.5 | Low |

**📌 Ví dụ minh hoạ**

**🔹 Trường hợp 1: An toàn cao**

* 20 giao dịch thành công, 0 thất bại
* failureRate = 0, consecutiveFailures = 0, không có lỗi gần đây  
  → securityScore = 1.0 → High

**🔹 Trường hợp 2: Trung bình**

* 15 success, 3 failure → failureRate ≈ 0.17
* consecutiveFailures = 1
* 1 lỗi gần đây  
  → securityScore = 0.5–0.6 → Medium

**🔹 Trường hợp 3: Nguy hiểm**

* 5 success, 5 failure → failureRate = 0.5
* consecutiveFailures = 4
* 2 lỗi gần đây  
  → securityScore ≈ 0.3 \* (1 - 0.2) ≈ 0.24 → Low