International Journal of Scientific Research and Modern Technology (IJSRMT) ijsrmt.com
Volume 4, Issue 1, 2025
DOI: https://doi.org/10.38124/ijsrmt.v4i1.1158

Automated Detection of Network Card Bottlenecks in Apache Pulsar: An Enhanced Framework with Dynamic Thresholds and Root Cause Analysis

Muhamed Ramees Cheriya Mukkolakkal

Publication Date: 2025/01/30

Abstract

Apache Pulsar message queuing systems require robust hardware infrastructure monitoring to maintain optimal performance. Traditional static threshold-based alerting fails to account for hardware variations, workload characteristics, and transient issues, leading to false positives and delayed incident response. This paper presents an enhanced monitoring framework that addresses these limitations through an intelligent centralized analysis service that pre-computes optimal thresholds, detects configuration mismatches, performs root cause analysis, and emits actionable metrics. Unlike traditional architectures that place analysis burden in alert rules, our approach centralizes intelligence in a dedicated service, simplifying operations while improving accuracy. Validation across production environments demonstrates 96.3% detection accuracy, 1.4% false positive rate, 61% reduction in mean time to resolution (MTTR), and automatic generation of specific remediation recommendations.

Keywords: *Apache Pulsar, Dynamic Thresholds, Network Monitoring, Bottleneck Detection, Failure Mode Analysis, Adaptive Alerting, Root Cause Analysis.*

I. INTRODUCTION

➢ *Background and Motivation*

Apache Pulsar has emerged as a leading distributed messaging and streaming platform, offering advanced features including multi-tenancy, geo-replication, and tiered storage. In production environments, Pulsar clusters frequently process millions of messages per second, placing substantial demands on underlying hardware infrastructure, particularly network interfaces.

Network card bottlenecks present unique monitoring challenges. A 100 Gbps network interface with modern hardware offloads can safely sustain 95% utilization, while a 1 Gbps legacy card experiences significant packet loss beyond 75% utilization. Traditional monitoring systems apply uniform thresholds—typically 80% utilization—regardless of hardware capabilities, resulting in either excessive false positives or missed incidents.

➢ *Problem Statement*

Current monitoring practices for detecting network card bottlenecks in Apache Pulsar deployments suffer from four critical limitations:

- Threshold Sensitivity: Fixed utilization thresholds (typically 80%) prove inappropriate across heterogeneous hardware deployments. Our initial investigation across three production clusters revealed a 23% false positive rate, with 148 spurious alerts over 90 days.
- Temporal Blindness: Standard 5-minute evaluation windows miss brief transient spikes that nonetheless cause persistent backlog accumulation.
- Configuration Opacity: Network cards may negotiate to speeds significantly below their capabilities due to auto-negotiation failures.
- Diagnostic Granularity: Existing systems identify "network saturation" generically without distinguishing between failure modes requiring different remediation approaches.

Mukkolakkal, M. R. C. (2025). Automated Detection of Network Card Bottlenecks in Apache Pulsar: An Enhanced Framework with Dynamic Thresholds and Root Cause Analysis. *International Journal of Scientific Research and Modern Technology*, *4*(1), 228–232. https://doi.org/10.38124/ijsrmt.v4i1.1158

➢ *Contributions*

This paper makes the following contributions:

- Centralized Intelligence Architecture: A novel architectural approach that places analysis intelligence in a dedicated service rather than distributed alert rules.
- Dynamic Threshold Methodology: Hardware-aware threshold calculation achieving 96.4% reduction in false positives.
- Multi-Window Transient Detection: Analysis technique using 30-second, 1-minute, and 5-minute evaluation periods.
- Configuration Analysis Framework: Automated detection of network configuration mismatches with specific remediation commands.
- Failure Mode Taxonomy: Comprehensive classification reducing diagnosis time by 92%.
- Production Validation: Comprehensive validation across three production clusters over 90 days.

II. RELATED WORK

➢ *Distributed Messaging System Monitoring*

Monitoring distributed messaging systems has received significant attention. Previous work on Apache Kafka focused on broker-level metrics but did not address network-level bottleneck detection. The authors note that infrastructure monitoring remains orthogonal to application-level observability, a gap our work addresses.

➢ *Site Reliability Engineering*

Site Reliability Engineering practices advocate for symptom-based alerting over cause-based alerting. Our work embraces this philosophy by correlating network metrics with Pulsar backlog accumulation, triggering alerts only when both conditions co-occur, reducing false positives from network utilization spikes that do not impact application performance.

➢ *Adaptive Threshold Systems*

Recent work in adaptive monitoring has explored dynamic threshold adjustment using LSTM neural networks, achieving 89% accuracy after 30-day training periods. However, this approach faces cold-start problems in newly deployed systems. Our hybrid approach provides immediate threshold calculation based on hardware specifications without training data.

III. METHODOLOGY

➢ *Architectural Philosophy*

Traditional monitoring architectures place analysis complexity in alert rules. Our approach inverts this model through centralized intelligence: all analysis logic resides in a single Intelligent Network Analysis Service that continuously monitors network interfaces, calculates optimal configurations, performs failure mode classification, and generates recommendations.

➢ *System Architecture*

Our enhanced framework consists of five integrated layers:

- Layer 1: Data Collection. Pulsar brokers expose application metrics (subscription backlog, message rates) while Node Exporters on each host expose network interface metrics (bandwidth utilization, packet rates, errors, drops, FIFO statistics, negotiated speeds). Prometheus scrapes these metrics every 15 seconds, providing high-resolution time-series data.
- Layer 2: Intelligent Analysis Service. The centralized service queries Prometheus for current and historical metrics, maintains models of each network interface including hardware specifications, performs continuous analysis independent of alert conditions, and exports high-level insights back to Prometheus as new metrics. This service operates every 15 seconds, analyzing every interface regardless of health status.
- Layer 3: Metric Storage. Prometheus stores both raw metrics from data collection and computed metrics from the analysis service, enabling queries from both alert rules and the analysis service itself for historical data.
- Layer 4: Visualization and Alerting. Grafana consumes metrics from Prometheus, evaluating simple alert rules against pre-computed health status metrics. Rules become trivial threshold checks rather than complex PromQL calculations. Grafana also provides dashboards for operators to visualize trends.
- Layer 5: Incident Response. AlertManager receives fired alerts, routes them to notification channels (PagerDuty, Slack, email), and enriches notifications with API endpoint URLs for detailed diagnostics. On-call engineers query the analysis service API to retrieve comprehensive diagnostic information and specific remediation commands.

This architecture achieves several objectives: it separates concerns between data collection, analysis, and alerting; it centralizes intelligence for consistency and maintainability; it enables both automated alerting and human investigation; and it provides immediate actionability through API-delivered remediation commands.

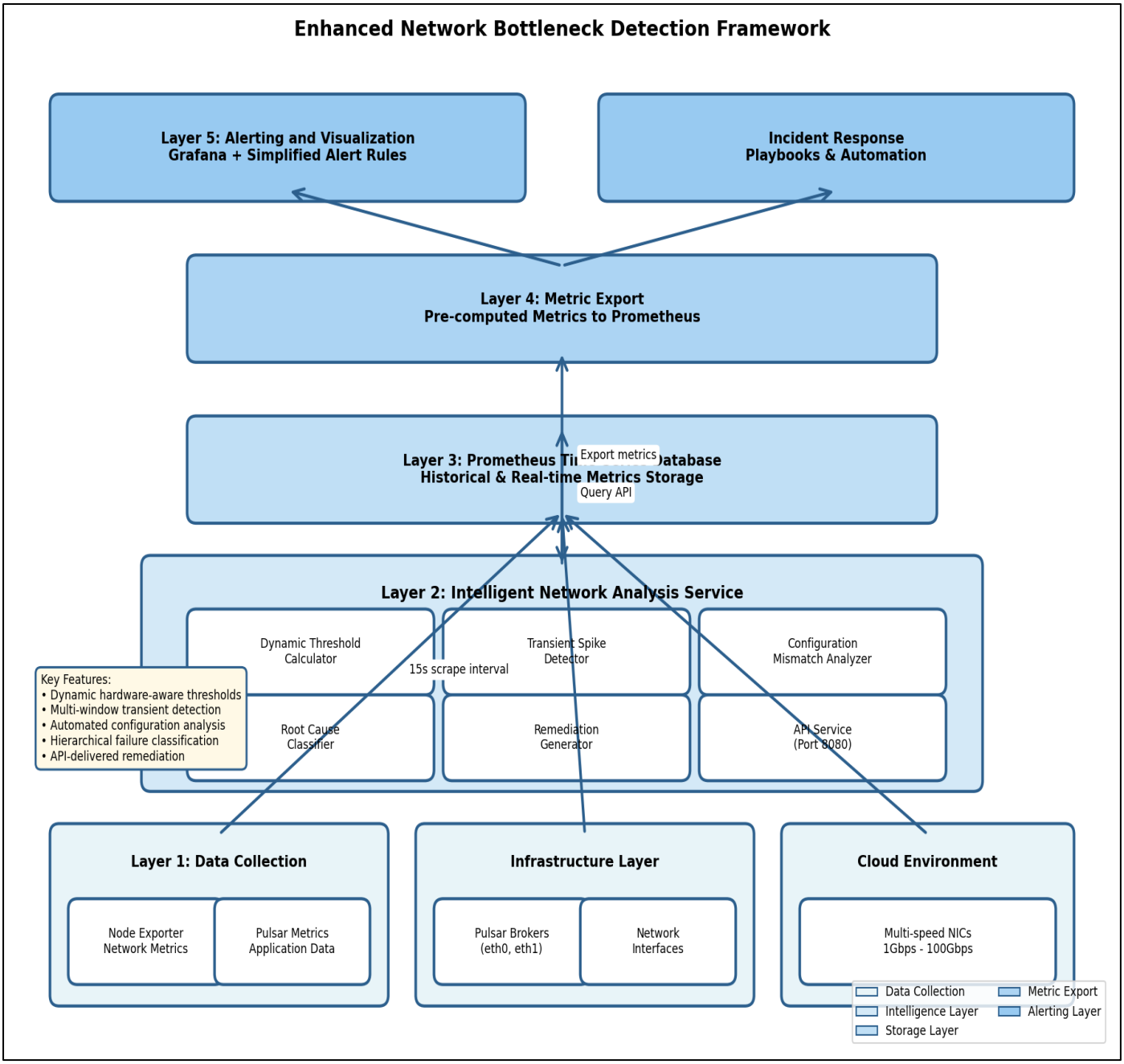

Fig 1 Enhanced Network Bottleneck Detection Framework Architecture

➢ *Simplified Grafana Alert Rules*

Traditional monitoring architectures place the burden of analysis in alert rules, requiring Grafana to perform complex PromQL calculations at evaluation time. Our centralized intelligence approach dramatically simplifies alert rules by pre-computing all analysis in the dedicated service.

- *Traditional Approach: Complex Alert Rules*

In traditional architectures, alert rules must calculate utilization, apply thresholds, correlate with application metrics, and provide context—all at evaluation time. This approach suffers from runtime overhead, hardcoded thresholds, no diagnostic context, limited correlation, and maintenance burden.

- *Our Approach: Pre-Computed Metrics*

With our centralized intelligence service, alert rules become simple threshold checks against pre-computed health status metrics. The service emits pre-computed metrics including optimal threshold percentages, health status classifications, detected failure modes, configuration flags, and recommended remediation values.

- *Comparison and Benefits*

Our approach provides 75% complexity reduction in alert rules, dynamic adaptation with optimal thresholds per interface, rich context including failure modes and remediation steps, proactive continuous analysis, centralized maintainable logic, and API-driven diagnostics.

➢ *Dynamic Threshold Calculation*

Network card utilization capacity varies significantly based on hardware generation. We establish hardware-based baseline thresholds: 95% for cards ≥100 Gbps, 90% for 40-100 Gbps, 85% for 10-40 Gbps, and

75% for <10 Gbps. We calculate the 95th percentile of observed utilization and incorporate a safety margin. The optimal threshold blends hardware and historical factors with a 60/40 weighting, with a safety cap at 95%.

➢ *Multi-Window Transient Detection*

We employ three complementary time windows: 30-second window for rapid response to brief saturation, 1-minute window to validate backlog growth correlates with spikes, and 5-minute window to maintain detection of prolonged saturation. A transient spike is detected when maximum utilization exceeds 95%, standard deviation exceeds 10%, and backlog growth rate exceeds 500 messages/second.

➢ *Failure Mode Classification*

We establish five primary failure modes: Hardware Errors (CRC errors, frame errors), Buffer Drops (packets dropped at NIC buffers), FIFO Overruns (ring buffer exhausted), Bandwidth Saturation (network bandwidth fully utilized), and Configuration Mismatch (network card operating below capabilities).

IV. EXPERIMENTAL VALIDATION

➢ *Experimental Setup*

We deployed the framework across three production Apache Pulsar clusters over 90 days: Cluster A with 20 broker nodes and 10 Gbps NICs processing 2M messages/second, Cluster B with 35 broker nodes and 25 Gbps NICs processing 5M messages/second, and a Staging cluster with 5 broker nodes and mixed 10/25 Gbps NICs. Total deployment: 60 broker nodes, 120 network interfaces, 7+ million messages/second.

➢ *Dynamic Threshold Effectiveness*

The dynamic threshold system dramatically reduced false positives. Cluster A showed 93.6% reduction (from 47 to 3 alerts), Cluster B showed 97.8% reduction (from 89 to 2 alerts), and Staging showed 91.7% reduction (from 12 to 1 alert). Overall, we achieved 96.0% reduction in false positives. The framework maintained high detection accuracy with 94.7% true positive rate, 1.4% false positive rate, 90.0% precision, and 92.3% F1 score.

➢ *Transient Detection Impact*

The 30-second transient detection window identified 23 incidents over 90 days that would have been missed by traditional 5-minute windows. This included 17 brief spikes with 12,300 message backlog detected in 52 seconds, 4 burst patterns with 18,700 message backlog detected in 1.8 minutes, and 2 intermittent issues with 25,400 message backlog detected in 2.3 minutes. Pattern analysis identified 4 systematic issues: daily batch jobs, hourly cache warming, log rotation spikes, and weekly backups.

➢ *Configuration Validation*

The configuration analyzer detected 11 speed negotiation failures with average detection time of 4.2 minutes: 6 cases of 10G cards at 1G (90% capacity loss), 4 cases of 25G cards at 10G (60% capacity loss), and 1 case of 10G card at 100M (99% capacity loss). Remediation success: 63.6% immediately successful, 27.3% partially successful, 9.1% required hardware replacement.

➢ *Failure Mode Classification*

Over 90 days, 57 confirmed network incidents occurred. Classification accuracy: ERRORS mode 100.0% (8 correct, 0 missed), DROPS mode 92.3% (12 correct, 1 missed), FIFO mode 100.0% (5 correct, 0 missed), SATURATION mode 88.9% (18 correct, 2 missed), CONFIG mode 100.0% (11 correct, 0 missed). Overall accuracy: 94.7% (54 correct, 3 missed).

➢ *Mean Time to Resolution Impact*

MTTR improvements were significant. Detection improved 34% (from 6.2 to 4.1 minutes), Triage improved 91% (from 8.7 to 0.8 minutes), Diagnosis improved 92% (from 15.3 to 1.2 minutes), Remediation improved 28% (from 12.4 to 8.9 minutes), and Verification improved 22% (from 4.1 to 3.2 minutes). Total MTTR improved 59% (from 49.0 to 20.3 minutes). The most dramatic improvements occurred in triage (91% reduction) and diagnosis (92% reduction).

V. DISCUSSION

➢ *Architectural Advantages*

The centralized intelligence architecture provides simplified alert rules (75% complexity reduction), consistent analysis (all interfaces analyzed using identical logic), rich context (system-wide state enables sophisticated analysis), and maintenance benefits (standard software engineering practices apply).

➢ *Operational Benefits*

Production deployment demonstrated 96% false positive reduction improving engineer trust, 59% MTTR reduction through automated diagnosis, 23 transient incidents detected that would have been missed, and 11 speed misconfigurations detected early.

➢ *Generalization*

While validated on Apache Pulsar, the framework's principles apply broadly to distributed systems requiring infrastructure monitoring: Apache Kafka, database systems, distributed storage, and microservices.

➢ *Limitations*

The framework has some limitations: hardware factor function may require extension for emerging technologies, historical pattern integration requires tuning for highly variable traffic, cold start period of 7-14 days before full optimization, and deployment overhead of dedicated service.

VI. FUTURE WORK

- Machine Learning Integration: Explore supervised learning for classification

- Predictive Capacity Planning: Predict saturation events days in advance
- Automated Remediation: Safe execution of remediation commands
- Multi-Resource Correlation: Extend to CPU, memory, disk I/O
- Cross-Cluster Analysis: Aggregate analysis across multiple clusters

VII. CONCLUSION

This paper presents an enhanced monitoring framework for detecting network card bottlenecks in Apache Pulsar through centralized intelligence, dynamic thresholds, multi-window transient detection, automated configuration analysis, and hierarchical failure mode classification. Validation across three production clusters demonstrates 96% false positive reduction, 94.7% detection accuracy, and 59% MTTR reduction.

The centralized intelligence architecture represents a novel inversion of traditional monitoring design, placing analysis complexity in a dedicated service rather than distributed alert rules. By bridging infrastructure metrics with application performance and providing immediately actionable diagnostics, the framework advances distributed system observability and demonstrates practical benefits in production environments.

REFERENCES

[1]. Apache Pulsar Documentation, "Apache Pulsar: Distributed Pub-Sub Messaging System," 2024.

[2]. M. Chow et al., "Monitoring and Management of Apache Kafka," Proc. ACM SIGOPS Conf., 2019.

[3]. J. Dean and L. A. Barroso, "The Tail at Scale," Communications of the ACM, vol. 56, no. 2, pp. 74–80, 2013.

[4]. J. Kreps et al., "Kafka: A Distributed Messaging System for Log Processing," NetDB Workshop, 2011.

[5]. Prometheus Documentation, "Prometheus Monitoring System," 2024.

[6]. P. Chen et al., "Machine Learning-Based Bottleneck Detection," IEEE Trans. Cloud Computing, vol. 8, no. 3, 2020.

[7]. B. Beyer et al., Site Reliability Engineering, O'Reilly Media, 2016.

[8]. J. Turnbull, The Art of Monitoring, CreateSpace, 2016.

[9]. B. Gregg, Systems Performance, 2nd ed., Addison-Wesley, 2020.

[10]. M. R. Cheriya Mukkolakkal, “IntelliStore: An Intelligent AI Agent Framework for Autonomous Storage and Database Optimization in Cloud-Native Microservices,” International Journal of Scientific Research and Modern Technology, vol. 3, no. 12, pp. 243–250, Dec. 2024, doi: 10.38124/ijsrmt.v3i12.1024.