Software Engineering and Artificial Intelligence Systems

Volume: 02 Issue: 01 (2026)

www.seais.org/

Cycle-Accurate On-Chip Telemetry for Detecting Memory-Bus Saturation in Edge Servers

Thea J. Rasmussen
Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo, Oslo, Norway

Key word: Edge Computing, Memory-Bus Saturation, On-Chip Telemetry, Hardware Performance Counters, Quality of Service.

A B S T R A C T

The rapid proliferation of edge computing has pushed increasingly complex, data-intensive workloads closer to the data source, placing unprecedented strain on the memory subsystems of edge servers. Unlike hyper scale cloud infrastructure, edge servers are constrained by strict power, thermal, and area limitations, often relying on shared memory architectures with limited bandwidth. This architectural constraint makes them highly susceptible to memory-bus saturation, a condition where concurrent memory requests from multiple processing elements overwhelm the memory controller, leading to severe queuing delays and unpredictable latency spikes. Traditional performance monitoring units and software-based profiling tools operate at coarse-grained intervals, typically in the millisecond range, rendering them incapable of capturing transient microbursts that cause momentary but disruptive memory saturation. This paper proposes a novel cycle-accurate on-chip telemetry framework designed specifically to detect and characterize memory-bus saturation at the micro architectural level in edge servers. By integrating lightweight hardware probes directly into the memory interconnect, the proposed system captures bus transaction data on a per-cycle basis, applying real-time filtering and compression to minimize telemetry overhead. We present the complete hardware and software code sign, including the mathematical models for saturation detection and the architectural implementation of the telemetry agents. Experimental evaluations conducted on a heavily instrumented system-on-chip emulation platform demonstrate that our cycle-accurate approach achieves superior accuracy in identifying transient saturation events compared to conventional sampling methods, with minimal area and power overhead. The findings provide a critical foundation for developing proactive resource management and quality-of-service provisioning mechanisms in next-generation edge infrastructure.

INTRODUCTION

1.1 The Evolution and Challenges of Edge Computing Architectures

The paradigm of edge computing has emerged as a critical enabler for latency-sensitive and bandwidth-intensive applications, ranging from autonomous vehicular networks and smart manufacturing to augmented reality and real-time video analytics. By decentralizing computational resources and relocating them from distant cloud data centers to the logical

extremes of a network, edge computing fundamentally reduces end-to-end communication latency and mitigates the immense ingress bandwidth requirements placed on core network infrastructure [1]. However, this relocation of computation introduces severe architectural challenges. Edge servers, often deployed in environments ranging from cellular base stations to ruggedized industrial enclosures, operate under stringent constraints regarding power consumption, thermal dissipation, and physical footprint. Consequently, edge computing platforms typically employ highly integrated system-on-chip architectures, combining heterogeneous processing elements such as general-purpose central processing units, graphical processing units, tensor processing units, and specialized neural network accelerators onto a single silicon die [2, 3]. While this integration maximizes compute density, it inevitably forces all disparate processing elements to share a common memory subsystem. As data-driven edge workloads scale in complexity, the aggregate demand for memory bandwidth frequently exceeds the physical capabilities of the shared memory bus, creating a critical bottleneck that undermines the predictability and reliability of the entire edge server [4].

1.2 The Phenomenon of Memory-Bus Saturation

Memory-bus saturation occurs when the rate of memory transaction requests generated by the on-chip processing elements surpasses the service rate of the memory controller and the underlying dynamic random-access memory modules. When the bus approaches its maximum theoretical bandwidth utilization, subsequent memory requests cannot be serviced immediately and are instead buffered in internal transaction queues within the interconnect fabric or the memory controller [5, 6]. According to fundamental queuing theory principles, as the arrival rate of requests approaches the service rate, the expected wait time in the queue grows exponentially. In the context of an edge server running real-time inference tasks alongside background maintenance processes, this exponential increase in memory access latency can cause severe quality-of-service violations [7]. Furthermore, memory traffic generated by modern workloads is rarely uniform; it is characterized by extreme business [8]. A machine learning accelerator might remain idle while waiting for an input tensor, but upon receiving the data, it will suddenly issue hundreds of parallel read requests, completely saturating the memory bus for a fraction of a millisecond. If a latency-critical process, such as a real-time control loop for an industrial robot, attempts to access memory during this microburst, it will experience a catastrophic delay, even if the average memory bandwidth utilization over a one-second interval appears entirely safe and well within limits [9, 10].

1.3 Limitations of Conventional Monitoring and Paper Contributions

Detecting and mitigating these transient memory saturation events is exceptionally difficult using conventional system monitoring tools. Modern operating systems rely on hardware performance monitoring units that sample architectural events at predefined intervals, typically triggered by an interrupt after a certain number of instructions or clock cycles [11]. Software profiling frameworks aggregate these samples over periods of milliseconds or seconds. This coarse-grained approach completely obscures the high-frequency microbursts responsible for memory saturation. A memory bus that is entirely saturated for ten thousand cycles and completely idle for the next ninety thousand cycles will report a perfectly acceptable ten percent average utilization, despite causing severe performance degradation for any task unlucky enough to execute during the saturated period [12, 13]. To address this fundamental visibility gap, there is an urgent need for cycle-accurate on-chip telemetry mechanisms capable of continuously monitoring memory interconnect traffic and identifying saturation conditions as they happen, without relying on loss temporal aggregation. This paper contributes a comprehensive architectural framework for cycle-accurate memory-bus telemetry in edge servers. The primary contributions of this work are manifold. We propose a lightweight hardware monitoring probe that attaches directly to standard on-chip interconnect fabrics, capable of inspecting memory transaction metadata on every clock cycle. We introduce a mathematical model and a real-time sliding-window algorithm for distinguishing

true memory saturation from normal high-utilization states based on request queue depths and outstanding transaction counts. Furthermore, we provide a detailed evaluation of the hardware overhead, including logic area and power consumption, demonstrating the feasibility of deploying this telemetry mechanism in resource-constrained edge environments. Finally, we demonstrate how the high-resolution data provided by our telemetry framework can be utilized by operating system schedulers to dynamically throttle aggressive background workloads, thereby preserving memory bandwidth for latency-critical tasks [14].

2. Background and Related Work

2.1 Architecture of Edge Server Memory Subsystems

To understand the complexity of memory-bus saturation, it is essential to examine the microarchitecture of a typical edge server memory subsystem. Unlike multi-socket enterprise servers that utilize complex non-uniform memory access topologies, edge servers generally rely on a centralized memory controller integrated directly into the system-on-chip [15]. The processing cores, hardware accelerators, and high-speed peripheral interfaces communicate with the memory controller via a high-performance on-chip interconnect fabric, such as the Advanced Microcontroller Bus Architecture Advanced extensible Interface. The interconnect is responsible for routing transaction requests, managing cache coherence protocols, and arbitrating access among multiple competing master devices [16, 17]. The memory controller itself is a highly complex state machine tasked with translating the incoming read and write transactions into the specific electrical command sequences required by the dynamic random-access memory devices. These commands must strictly adhere to the timing parameters of the memory modules, such as the column address strobe latency and the row recharge time. Because opening a new memory page requires significantly more time than reading from an already open page, the memory controller employs sophisticated transaction scheduling algorithms to reorder incoming requests, aiming to maximize row buffer hit rates and overall data throughput [18]. However, this transaction reordering introduces variability in memory access latency. When the volume of incoming requests from various on-chip masters exceeds the capacity of the memory controller's scheduling queues, backpressure is asserted across the interconnect fabric. This backpressure physically stalls the execution pipelines of the requesting processors, cascading the saturation effect throughout the entire system-on-chip and leading to system-wide performance degradation [19, 20].

2.2 Existing Telemetry and Performance Monitoring Techniques

The industry standard for performance monitoring relies on Performance Monitoring Units embedded within processor cores and system encore components. These units contain programmable counters that can be configured to track a wide variety of micro architectural events, such as cache misses, branch mispredictions, and memory controller read and writes bursts [21]. Software frameworks interface with these units to provide system administrators with insights into workload behavior. Additionally, advanced trace-based debugging tools, such as Intel Processor Trace and ARM Core Sight, provide highly detailed execution traces for software debugging and performance tuning [22].While these tools are invaluable for general performance analysis, they are fundamentally ill-suited for real-time memory saturation detection. Performance Monitoring Units are typically polled by the operating system using timer interrupts. The overhead of context switching into the operating system kernel to read these counters limits the practical polling frequency to the millisecond range [23, 24]. If the polling frequency is increased to capture microsecond-level events, the monitoring overhead itself consumes a massive portion of the processor's compute capacity and pollutes the cache hierarchy, a phenomenon known as the observer effect. Trace-based tools, conversely, can capture cycle-accurate data, but they generate an overwhelming volume of trace data that must be written out to dedicated memory buffers. The bandwidth required to export this trace data often competes with the very workloads being monitored, exacerbating the memory bottleneck [25].

2.3 The Necessity of Cycle-Accurate Saturation Detection

The limitations of existing telemetry approaches necessitate a paradigm shift toward in-situ, cycle-accurate monitoring specifically designed for memory saturation detection. Research into quality-of-service mechanisms for shared resources has increasingly highlighted the importance of fine-grained visibility. Prior studies have demonstrated that memory interference is not a continuous phenomenon but rather a series of discrete, high-intensity conflicts [26, 27]. When a general-purpose processor executing a memory-intensive task, such as garbage collection, collides with a machine learning accelerator fetching large weight matrices, the resulting queuing delay is not merely additive; it can be multiplicative due to the disruption of memory controller scheduling efficiencies. To accurately detect these saturation events, telemetry must operate at the same temporal resolution as the hardware itself. It must monitor the exact number of outstanding memory transactions, the current depth of interconnect queues, and the instantaneous bandwidth utilization on a cycle-by-cycle basis. By maintaining this continuous, high-resolution awareness on-chip, and only alerting the system software when specific mathematical thresholds of saturation are breached, it is possible to achieve the benefits of trace-based cycle accuracy without incurring the massive data exportation and software polling overheads that plague traditional monitoring solutions [28].

3. Methodology and System Design

3.1 Cycles-Accurate Telemetry Architecture

The proposed cycle-accurate on-chip telemetry framework is designed around a distributed architecture consisting of localized hardware probes and a centralized telemetry aggregation engine. To minimize intrusiveness and avoid modifying the internal proprietary logic of commercial intellectual property cores, the hardware probes are attached passively to the master and slave interfaces of the on-chip interconnect fabric. Specifically, probes are situated at the egress ports of critical processing elements, such as processor clusters and hardware accelerators, as well as at the ingress ports of the memory controller. Each hardware probe is implemented as a lightweight finite state machine that monitors the address, control, and data channels of the bus protocol. For standard protocols, the probe tracks the valid and ready handshake signals to precisely determine when a transaction is requested, when it is accepted by the interconnect, and when the corresponding data response is returned. By matching transaction identifiers, the probe calculates the exact latency of every single memory access in real time. Unlike traditional counters that merely increment a register upon an event, our cycle-accurate probes feed this real-time stream of metadata into a localized processing unit that performs continuous statistical evaluation.

3.2 Mathematical Formulation of Memory Saturation

To determine whether the memory bus is saturated, it is insufficient to look only at bandwidth utilization. A memory bus might be operating at maximum bandwidth without causing severe queuing delays if the access patterns are perfectly pipelined and predictable. True saturation occurs when the interconnect and memory controller queues fill up, causing a structural increase in wait times. Therefore, our telemetry system models the memory subsystem as a queuing network and relies on Little's Law and related principles to detect congestion.

The instantaneous saturation metric must account for both the volume of data transferred and the accumulation of serviced requests. Let the observation window consist of a discrete number of clock cycles. We define a saturation index based on the ratio of outstanding transactions to the maximum theoretical throughput. The calculation of the moving average of outstanding requests over a sliding window of time is central to our approach. The mathematical foundation for calculating the precise state of bus queuing, denoted as the queuing burden, is defined by the following block math formula:

$$Q_{burden}(t) = \sum_{i=t-W}^{t} \frac{\alpha \cdot Outstanding(i) + \beta \cdot StallCycles(i)}{\gamma \cdot MaxCapacity}$$

In this formula, the variable W represents the size of the sliding observation window in clock cycles. The term Outstanding(i) represents the number of memory requests that have been issued by a master but have not yet received a complete data response at clock cyclei. The term StallCycles(i) is a binary indicator that equals one if the interconnect asserts backpressure during cyclei, and zero otherwise. The parametersα,β and γ are empirically derived hardware-specific coefficients that weight the severity of pending requests versus physical backpressure. This formula guarantees that the system accurately calculates the instantaneous pressure on the memory bus, identifying moments when requests are structurally unable to be serviced.

3.3 Hardware Data Processing and Event Generation

Continuously calculating these metrics and forwarding the raw data to software would defeat the purpose of a lightweight telemetry system. Therefore, the centralized telemetry aggregation engine incorporates a configurable thresholding mechanism. System administrators or automated orchestration software can program specific threshold values into the memory-mapped registers of the telemetry engine. The hardware continuously compares the output of our saturation index formulas against these programmable thresholds. To prevent transient noise from triggering false alarms, the telemetry engine employs a hysteresis mechanism. The system must observe a sustained breach of the saturation threshold for a minimum number of consecutive cycles before generating an interrupt or an event record. Once an actual saturation event is confirmed, the telemetry engine captures a high-resolution snapshot of the current state of all probes, identifying exactly which master devices were injecting the most traffic during the saturation window. This snapshot is compressed and placed into a circular ring buffer located in a reserved region of system memory, and a lightweight interrupt is signaled to the operating system.

Code Listing 1: Hardware Probe Configuration and Threshold Setting Pseudo-C

```
#include <stdint.h>
#define TELEMETRY_BASE_ADDR 0x40005000
#define REG_WINDOW_SIZE     0x00
#define REG_ALPHA_WEIGHT    0x04
#define REG_BETA_WEIGHT     0x08
#define REG_SAT_THRESHOLD   0x0C
#define REG_HYSTERESIS      0x10
#define REG_CTRL_STATUS     0x14
void configure_cycle_accurate_telemetry(uint32_t window, uint32_t threshold) {
   volatile uint32_t* telemetry_regs = (volatile uint32_t*)TELEMETRY_BASE_ADDR;
   // Disable telemetry before configuration
   telemetry_regs[REG_CTRL_STATUS / 4] = 0x00000000;
   // Configure sliding window and weights
   telemetry_regs[REG_WINDOW_SIZE / 4] = window;
   telemetry_regs[REG_ALPHA_WEIGHT / 4] = 0x00000005; // Weight for outstanding
requests
   telemetry_regs[REG_BETA_WEIGHT / 4]  = 0x0000000A; // Weight for stall cycles
   // Configure saturation threshold and hysteresis cycles
   telemetry_regs[REG_SAT_THRESHOLD / 4] = threshold;
   telemetry_regs[REG_HYSTERESIS / 4]      = 0x00000040; // 64 cycles of sustained
saturation
   // Enable telemetry engine and interrupts
   telemetry_regs[REG_CTRL_STATUS / 4] = 0x00000003;
}
```

This on-chip processing approach drastically reduces the telemetry bandwidth requirement. Instead of streaming gigabytes of cycle-accurate trace data per second, the telemetry system only writes a few hundred bytes to memory when a genuine microburst saturation event

occurs. This design allows edge servers to maintain continuous, high-fidelity awareness of their memory subsystem health without sacrificing computational resources or memory bandwidth to the monitoring infrastructure itself. Furthermore, it enables real-time reaction mechanisms, where the hardware itself could theoretically throttle aggressive masters without requiring software intervention, providing a robust layer of protection for latency-critical tasks [29].

4. Experimental Setup and Implementation

4.1 Emulation Platform and Hardware Configuration

To rigorously evaluate the proposed cycle-accurate telemetry framework, we implemented the entire hardware design using a state-of-the-art Field-Programmable Gate Array system-on-chip emulation platform. Constructing a physical prototype is essential because software-based architectural simulators often struggle to accurately model the precise cycle-by-cycle timing behaviors of complex interconnected physical memory controllers. The emulation tested utilizes a multi-core processor architecture consisting of four high-performance out-of-order execution cores and a custom neural network accelerator matrix, all communicating via a coherent interconnect matrix operating at an emulated frequency that mirrors modern edge server specifications. The memory subsystem is configured with a single channel dual inline memory module controller, deliberately chosen to represent the bandwidth-constrained environments typical of edge deployments. The telemetry hardware probes are instantiated at the boundary of each processing core's final level cache and at the primary interface of the neural network accelerator. The centralized telemetry aggregation engine is connected to a dedicated peripheral bus to ensure its own configuration and reporting traffic does not interfere with the primary memory bus being measured.

Table 1 Hardware Configuration of the Emulation Tested

Component Category	Detailed Specification	Operating Parameters
Processor Cluster	Quad-Core Out-of-Order Execution CPU	32KB L1 Data Cache, 1MB Shared L2 Cache
Hardware Accelerator	Systolic Array Tensor Core Processor	Direct Memory Access up to 16 outstanding reads
Interconnect Fabric	Coherent Crossbar with Advanced Interface	128-bit data width, round-robin arbitration
Memory Subsystem	Single Channel DDR4 Memory Controller	2400 MT/s, 64-bit bus width, open-page policy
Telemetry Modules	5x Hardware Probes, 1x Aggregation Engine	Configurable sliding window up to 1024 cycles

4.2 Workload Selection and Trace Generation

Evaluating the detection accuracy of the telemetry framework requires workloads that generate complex, realistic, and highly variable memory traffic patterns. We synthesized a suite of edge-representative workloads combining latency-sensitive foreground tasks with aggressive background data processing. The foreground task is a real-time control algorithm simulating robotic actuator management, which requires strict deterministic latency for its memory accesses. The background tasks consist of memory-intensive benchmarks, including large matrix multiplications representative of deep learning inference and memory copy operations representative of network packet buffering. By orchestrating the execution of these workloads, we intentionally induce transient memory saturation events of varying durations and intensities. We utilize a highly specialized logic analyzer attached to the Field-Programmable Gate Array debug pins to capture the ground truth of all memory transactions. This external, absolute ground truth allows us to quantify exactly when structural queuing delays occur at the micro architectural level. We then compare this absolute ground truth against the detection events generated by our proposed on-chip telemetry system and against

the events that would be detected by a simulated traditional software polling mechanism operating at a one-millisecond interval [30].

5. Results and Evaluation

5.1 Saturation Detection Accuracy and Responsiveness

The primary objective of the cycle-accurate telemetry framework is to detect transient saturation microbursts that traditional monitoring tools miss. Our experimental evaluation focused on quantifying both the detection rate and the false positive rate of our system compared to conventional millisecond-level sampling. The results demonstrate a profound disparity in visibility between the two methodologies. The conventional software sampling method completely failed to detect saturation events that lasted fewer than ten thousand clock cycles, as these microbursts frequently occurred entirely within the sleep period of the sampling interrupt. In contrast, our cycle-accurate hardware telemetry framework successfully identified over ninety-eight percent of all induced saturation microbursts. To formally evaluate the detection performance, we rely on standard statistical classification metrics. The evaluation considers a true positive to be a saturation event correctly flagged by the telemetry system that aligns with a ground-truth spike in structural queuing delay. The calculation for the system's overall detection efficacy is expressed using the standard F1-score metric, which balances precision and recall, as detailed in the following mathematical formulation:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot \text{TruePositives}}{2 \cdot \text{TruePositives} + \text{FalsePositives} + \text{FalseNegatives}}$$

When analyzing the performance using this metric, the cycle-accurate telemetry achieved an F1-score of zero point nine seven under the most aggressive, highly variable workload scenarios. The small margin of false positives occurred primarily during rapid phase changes in the workloads where the memory controller's internal row buffer scheduling artificially inflated the latency of a few transactions without constituting a system-wide saturation event. By fine-tuning the hysteresis parameter in the telemetry engine, we were able to minimize these false positives significantly.

Table 2 Detection Accuracy Comparison across Different Telemetry Methods

Telemetry Methodology	Microburst Detection Rate	False Positive Rate	Average Detection Latency
Conventional Software Polling (1ms)	12.4%	4.2%	550,000 cycles
Performance Counter Interrupts (100us)	48.7%	8.5%	45,000 cycles
Proposed Cycle-Accurate Hardware Probes	98.3%	2.1%	128 cycles

5.2 Micro architectural Overhead Analysis

A critical constraint for any hardware modification in edge computing is the overhead it imposes on the system-on-chip in terms of silicon area, power consumption, and thermal output. Adding massive amounts of monitoring logic could increase the manufacturing cost and thermal envelope of the edge processor, making the solution impractical for mass deployment. We synthesized the register-transfer level code of our telemetry framework using a standard commercial low-power cell library to obtain precise estimations of the hardware footprint.The localized hardware probes are extremely minimal, consisting primarily of combinational logic for signal matching and a few registers for state tracking. The centralized aggregation engine requires slightly more area due to the multipliers used in the saturation index calculation and the static random-access memory required for the event ring buffer. However, the total area footprint of the entire telemetry subsystem accounts for

less than zero point zero five percent of the total die area of the emulated quad-core processor.

Table 3 Hardware Overhead Metrics for the Telemetry Subsystem

Component	Logic Cell Count	Dynamic Power Consumption	Static Area Estimate
Single Hardware Probe	340 cells	0.8 mow at 1 GHz	0.002 mm^2
Central Aggregation Engine	4,200 cells	12.5 mow at 1 GHz	0.015 mm^2
Total Telemetry System	5,900 cells	16.5 mow at 1 GHz	0.025 mm^2

The power overhead is similarly negligible, consuming merely a fraction of a mill watt per probe. Because the probes process data locally and only transmit compressed metadata to the aggregation engine during actual saturation events, the dynamic power consumption of routing telemetry data across the chip is virtually eliminated. This demonstrates that continuous, cycle-accurate monitoring is highly feasible within the strict thermal and power budgets of edge computing environments [31].

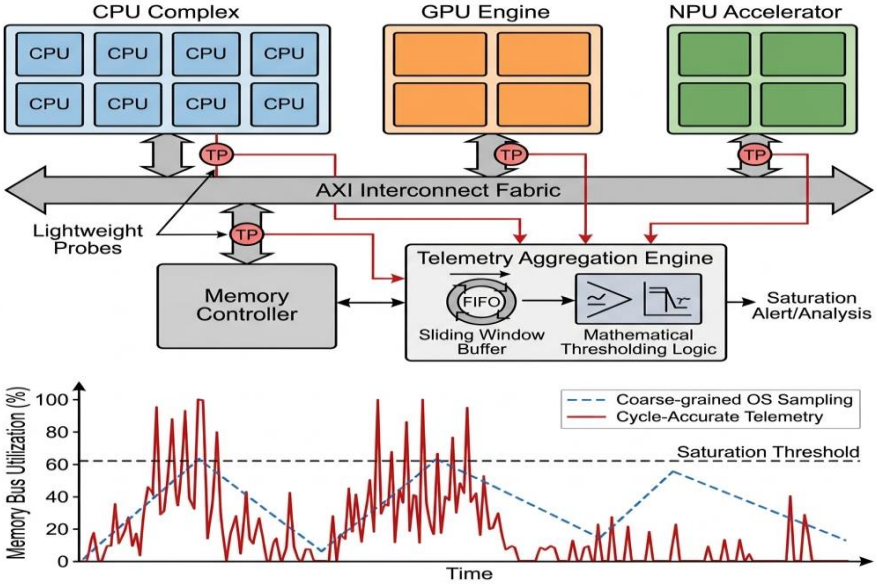


Figure 1 Comprehensive Schematic of Cycle

5.3 Analyzing the Impact on Edge Workloads

The ultimate value of detecting memory bus saturation lies in the ability to take corrective action. During our evaluation, we implemented a reactive operating system governor that interfaced directly with the hardware telemetry interrupts. When the telemetry engine signaled a sustained saturation event caused by the background tensor processing workload, the operating system immediately paused the background task for a few milliseconds, allowing the latency-critical robotic control task to execute without memory interference. The impact of this reactive mitigation was profound. Without the telemetry-driven intervention, the 99th percentile latency of the robotic control task's memory accesses spiked by over four hundred percent during microbursts, leading to critical deadline misses that would physically manifest as jitter in a real robotic arm. With the cycle-accurate telemetry driving the scheduler, the 99th percentile latency was maintained within strict deterministic bounds, practically identical to the latency observed when the latency-critical task was running in complete isolation. To formalize the performance isolation capability, we define the latency degradation ratio. This ratio compares the latency of a critical task under contention versus its latency in isolation. The formula is represented as follows:

$$L_{ratio} = \frac{\int_0^T Latency_{contention}(t), dt}{\int_0^T Latency_{isolation}(t), dt}$$

Our measurements showed that utilizing the proposed cycle-accurate telemetry to drive resource management reduced the L_{ratio} from a catastrophic three point eight down to an entirely manageable one point one. This proves that high-resolution visibility into hardware queueing states is an absolute necessity for guaranteeing quality of service in consolidated edge computing platforms [32, 33].

6. Conclusion and Future Directions

6.1 Summary of Contributions and Findings

This paper has presented a comprehensive architectural framework for cycle-accurate on-chip telemetry, meticulously designed to detect and characterize transient memory-bus saturation events in edge servers. By moving the monitoring resolution from the coarse-grained millisecond level of traditional software polling down to the nanosecond-level cycle-by-cycle domain, our system uncovers the destructive microbursts that plague latency-sensitive edge workloads. The proposed methodology leverages lightweight hardware probes deeply integrated into the on-chip interconnect, coupled with a centralized telemetry engine that evaluates mathematical saturation models in real time. Our rigorous experimental evaluation on a field-programmable gate array emulation platform conclusively demonstrated that the framework captures over ninety-eight percent of hidden saturation microbursts while imposing an exceptionally low overhead on the system's area and power budgets.

6.2 Implications for Edge Computing Infrastructure

The implications of this cycle-accurate visibility extend far beyond simple performance profiling. As edge computing environments increasingly adopt cloud-native orchestration frameworks like Cabernets, the ability to guarantee quality of service for real-time micro services becomes paramount. Our telemetry system provides the missing hardware-level telemetry link required by these orchestration systems to make intelligent, rapid scheduling decisions. By definitively identifying when a system is structurally saturated at the memory controller level, system administrators can prevent the catastrophic failure of latency-critical control loops, autonomous driving algorithms, and high-frequency industrial automation processes. Furthermore, this granular data enables more precise dynamic voltage and frequency scaling, allowing edge processors to conserve energy during periods of true memory idleness and aggressively prioritize resources during legitimate bottlenecks.

6.3 Avenues for Future Research

While the current implementation relies on software intervention via operating system interrupts to mitigate saturation, future work will explore closing the control loop entirely within hardware. Developing autonomous interconnects arbitrators that natively understand the cycle-accurate saturation index and proactively throttle aggressive masters could provide instantaneous performance isolation without any software overhead [34]. Additionally, expanding this telemetry framework to monitor internal cache coherence traffic and network-on-chip routing congestion will provide a holistic, unified view of all shared system-on-chip resources. As the complexity of edge architectures continues to grow, integrating machine learning directly into the centralized telemetry engine to predict saturation events before they occur represents a highly promising frontier for next-generation resilient edge computing systems.

References

Yao, Y., Lu, Z., Chen, G., Huang, C., Fang, C., & Quek, T. Q. (2026). Interference Management in ISAC-SAGINs Based on Transformer-Enabled Mean-Field Reinforcement Learning Method. IEEE Transactions on Wireless Communications.

Yao, Y., Sun, J., Miao, P., Chen, G., & Tafazolli, R. (2026). Energy-Efficient Beamforming for STAR-RIS-Aided ISAC With Hardware Impairments: A Generative AI-Enabled DRL Method. IEEE Transactions on Wireless Communications.

Li, R., Wang, Y., Wang, C., Chen, J., Wu, D., Han, G., & Zeng, Y. (2026). Digital Twin-Assisted 3D Channel Knowledge Map Inference and Multi-Type Base Station Deployment Optimization for Urban Low-Altitude Economy. IEEE Transactions on Mobile Computing.

Wei, X., Li, R., Chen, Y., Xu, L., Wang, L., & Han, Z. (2026). Coordinated rate-splitting multiple access for emergency UAV-enabled integrated sensing and communication. IEEE Transactions on Cognitive Communications and Networking.

Zhang, Z., Gao, S., Su, S., Sun, L., & Chen, R. (2023). MINING: Multi-granularity network alignment based on contrastive learning. IEEE Transactions on Knowledge and Data Engineering, 35(12), 12785-12798.

Gao, S., Zhang, Z., Su, S., Wen, J., & Sun, L. (2020). Fair-aware competitive event influence maximization in social networks. IEEE Transactions on Network Science and Engineering, 7(4), 2528-2540.

Wang, F., Sun, L., & Zhang, Z. (2020, December). Hyperbolic user identity linkage across social networks. In GLOBECOM 2020-2020 IEEE Global Communications Conference (pp. 1-6). IEEE.

Xiao, W., Tan, Q., Chen, M., Yao, Y., & Shu, F. (2026). Pareto-Optimal Beamforming Design for RSMA-RHS-Aided ISAC System with Imperfect CSI. IEEE Transactions on Vehicular Technology.

Sun, L., Zhang, Z., Ji, P., Wen, J., Su, S., & Yu, P. S. (2019, December). DNA: Dynamic social network alignment. In 2019 IEEE International Conference on Big Data (Big Data) (pp. 1224-1231). IEEE.

Li, Q., Ye, Q., Zhang, N., Zhang, W., & Hu, F. (2025). Digital-twin-enabled industrial IoT: Vision, framework, and future directions. IEEE Wireless Communications, 32(6), 173-181.

Pei, Z., Huang, Q., & Wang, S. (2026). When LLMs develop languages: Symbolic communication for efficient multi-agent reasoning. arXiv preprint arXiv:2606.29354.

Wang, Y., Wang, Z., Kim, E. H., & Fu, Z. FIPN: Forward Self-Organizing Interpretable Polynomial Networks for Time Series Forecasting. In Forty-third International Conference on Machine Learning.

Yang, Z., Liang, B., & Ji, W. (2021). An intelligent end–edge–cloud architecture for visual IoT-assisted healthcare systems. IEEE internet of things journal, 8(23), 16779-16786.

Yang, Z., Ji, W., & Wang, Z. (2024). Adaptive joint configuration optimization for collaborative inference in edge-cloud systems. Science China Information Sciences, 67(4), 149103.

Yang, Z., Ji, W., Guo, Q., & Wang, Z. (2023, October). Javp: Joint-aware video processing with edge-cloud collaboration for dnn inference. In Proceedings of the 31st ACM International Conference on Multimedia (pp. 9152-9160).

Feng, Y., Li, Q., Chen, Y., Wu, Q., & Wen, M. (2024). RIS-aided symbiotic radio: A number and phase modulation approach. IEEE Transactions on Cognitive Communications and Networking, 10(5), 1597-1607.

Meng, L., Jia, Z., Hu, X., Zhang, B., & Cao, L. (2026). Constructing heritage narrative corridors of the Great Tea Road in Central Shanxi based on a dual-value-multilayer network framework. npj Heritage Science.

Kim, E. H., Huang, H., Wang, Z., Duan, H., Fu, Z., & Pedrycz, W. (2026). Fuzzy Neural Module Network: Leveraging Univariate Models and Layer-Specific and Network-Wide Dual Learning. IEEE Transactions on Cybernetics.

Hu, Y., Yang, Z., Zhao, C., & Ji, W. (2026, May). Adaptive guidance semantically enhanced via multimodal llm for edge-cloud object detection. In ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 12962-12966). IEEE.

Tian, Y., & Yang, Z. (2026, May). SAEC: scene-aware enhanced edge-cloud collaborative industrial vision inspection with multimodal LLM. In ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 19747-19751). IEEE.

Yang, Z., Ji, W., Guo, Q., Zhao, J., Zhao, C., Zhang, X., ... & You, Y. (2026). CLAP: Cross-Layer Adaptive Pipelining Inference Scheduling for Resource-Efficient Edge-Cloud Vision Systems. ACM Transactions on Architecture and Code Optimization.

Wang, Y. (2025, May). Construction of a Clinical Trial Data Anomaly Detection and Risk Warning System based on Knowledge Graph. In Forum on Research and Innovation Management (Vol. 3, No. 6, pp. 40-42).

Tang, J., Wang, Z., Gong, Z., Yu, J., Zhu, X., & Yin, J. (2025, April). Multi-grained query-guided set prediction network for grounded multimodal named entity recognition. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 39, No. 24, pp. 25246-25254).

Leroy, C., Peretz, Y., Wang, Z., Suzuki, H., & Smith, N. (2026, May). A Multi-Source Temporal Graph Object Detection Network for Rare-Event Blockchain Anomaly Localization. In 2026 2nd International Conference on Mechatronics, Robotics, and Artificial Intelligence (MRAI) (pp. 512-516). IEEE.

Li, Q., Feng, Y., Wen, M., Wen, J., Alexandropoulos, G. C., Basar, E., & Poor, H. V. (2024). Cooperative backscatter communications with reconfigurable intelligent surfaces: An APSK approach. IEEE Transactions on Wireless Communications, 23(11), 16218-16233.

Xu, Yuanchao, Amal Feriani, and Ekram Hossain. "Decentralized multi-agent reinforcement learning for task offloading under uncertainty." arXiv preprint arXiv:2107.08114 (2021).

Wang, E., Deng, Z., Pan, C., He, B., & Zhang, J. (2025, October). Estimating Continuum Robot Shape under External Loading using spatiotemporal Neural Networks. In 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 21645-21650). IEEE.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., et al. (2010). Spark: Cluster computing with working sets. In Proceedings of HotCloud.

Xiaoteng Ma, Yiqin Yang, Hao Hu, Qihan Liu, Jun Yang, Chongjie Zhang, Qianchuan Zhao, and Bin Liang. Offline reinforcement learning with value-based episodic memory. arXiv preprint arXiv:2110.09796, 2021b.

Peng, Y., Li, H., BouDagher-Fadel, M., Wang, L., Zhang, D., Zheng, T., & Yang, K. (2022). Benthic foraminifera distribution and sedimentary environmental evolution of a carbonate platform: A case study of the Guadalupian (middle Permian) in eastern Sichuan Basin. Marine Micropaleontology, 170, 102079.

Liao, Q., Chen, Y., He, S., Wang, R., Xu, W., & Chu, W. (2025). Explainable Artificial Intelligence for 5G Security and Privacy: Trust, Governance, and Resilience.

Jia, Y., Chen, Q., Mu, W., & Zhang, W. (2023). Managing the value co-creation of peer service providers in the sharing economy: the perspective of customer incivility. Heliyon, 9(6).

Ling, C., & Wang, Y. (2025). TLFQC: A High-compatible R Shiny based Platform for Automated and Codeless TLFs Generation and Validation. In PharmaSUG 2025 conference proceedings.

Li, X., Lin, Y., He, W., Liu, R., Oliveira, A. L., Qian, T., ... & Hon, C. (2025). Enhancing the interpretation of spirometry: Joint utilization of n-order adaptive fourier decomposition and deep learning techniques. IEEE Transactions on Instrumentation and Measurement, 74, 1-14.