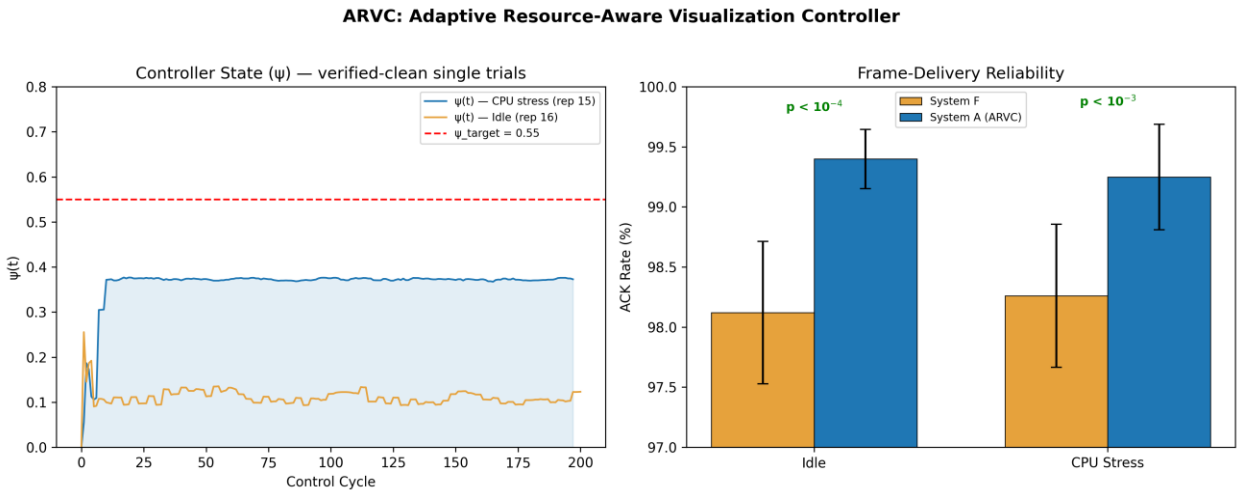
International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(7s) 2026 pp. 690-708
https://doi.org/10.70917/ijcisim-2026-3143
Article

ARVC: A Closed-Loop, Resource-Aware Controller for Latency-Bounded IoT Dashboard Rendering on Constrained Edge Devices

Shailendra Mishra[1]*, Rohit[2], Vipul Virsinghbhai Gamit[3], Manish Kumar Joshi[4], Namira Saiyad[5], Varsha Katia[6], Bharat Tank[7]

[1,2,7]Department of Computer Science & Engineering, Parul Institute of Engineering & Technology (PIET), Parul University, Vadodara, Gujarat 391760, India
shailendra.mishra40268@paruluniversity.ac.in; rohitatiiit@gmail.com; bharat.tank@paruluniversity.ac.in
[3,4]Department of Computer Application (MCA), Faculty of IT & Computer Science, Parul University, Vadodara, Gujarat 391760, India
vipulbhai.gamit26053@paruluniversity.ac.in; manishkumar.joshi42083@paruluniversity.ac.in
[5,6]Department of Civil Engineering, Parul Institute of Engineering & Technology (Diploma Studies), Parul University, Vadodara, Gujarat 391760, India
saiyednamira30@gmail.com; katiyavarsha@gmail.com

Abstract: When the rendering parameters are fixed on a device with limited resources, frame drops and latency spikes are observed when the rendering parameters exceed its CPU capacity. We introduce a lightweight proportional–integral controller, embedded in the push/down sampling loop of a Flask/Socket, called ARVC (Adaptive Resource-Aware Visualization Controller).A real-time CPU-Memory-buffer backlog-round-trip latency IO dashboard to calculate a composite resource-pressure $\psi(t) \in$. If $\psi(t)$ is larger than a prescribed value ($\psi\_target = 0.55$), ARVC discharges load by decreasing the number of points being rendered (LTTB down sampling), and by increasing the push interval; if the resources are nominal, ARVC restores full visual fidelity. We assessed ARVC on the Intel Berkeley Research Lab dataset replayed with the Raspberry Pi 4 (8 GB) at 50 ms intervals across the following 4 conditions: idle CPU, induced 4-core CPU-stress (23–25 independent trials of 30 s per condition; analysis was at the trial level to prevent pseudo replication). Under stress, ARVC raised frame-delivery reliability by 0.99 percentage points (ACK rate 99.25% vs. 98.26%; 95% CI [+0.27, +1.68] pp; Holm–Bonferroni corrected $p < 10^{-21}$), and it improved ACK rate by 1.28 pp under idle (95% CI [+0.66, +1.95] pp; $p < 10^{-34}$). The round-trip latency was statistically the same within 2ms (TOST $p = 0.0155$) at idle. No oscillation was seen during 3,768 verified control cycles on the controller. The results show that the closed-loop PI controller is able to reliably deliver the IoT dashboard frame on commodity edge hardware without changing protocols or hardware.



Graphical Abstract: Left: ARVC controller-state trace showing composite pressure ψ(t) (blue) for CPU-stress and idle (orange) conditions, with ψ_target = 0.55 dashed. Right: Frame-delivery reliability (ACK rate) comparison between System F (fixed) and System A (ARVC) under both load scenarios. Error bars: 95% Wilson-score confidence intervals.

Keywords: IoT edge computing, adaptive control, proportional–integral controller, real-time visualization, Raspberry Pi, Flask-SocketIO, resource management, frame delivery, buffer backlog, wireless sensor networks

Introduction

As the Internet of Things (IoT) deployments are rapidly growing, there is a tremendous need for real-time monitoring systems at the edge of the network. In edge computing, the data is processed near the source to minimize round-trip latency, to maintain privacy, and to reduce bandwidth requirements against the backdrop of the need for less cloud-based infrastructure. Single board computers, especially the Raspberry Pi series, are probably among the most popular edge platforms, as they provide a desirable combination of low cost, processing power, and a wide range of software support options for IoT applications. The live dashboard is one of the key parts of many edge-monitoring solutions; it is a browser-based visualization interface displaying real-time sensor data. This rendering layer is found in most real world IoT monitoring stacks, but has been underexplored as a target for adaptive control. Some existing framework like Grafana/Dash/Plotly/Flask+Socket for dashboard. The refresh rate and downsampling are usually set at deployment time for an IO implementation.

These are “static” parameters, which are easy to set up, but not very well designed for dynamic workloads. If there is a lot of CPU contention or other resource pressure, they can cause frame drops, latency increases, and poor user experience, but there is no automatic way to correct them. Previous research in neighboring fields indicates that adaptation under the influence of feedback can produce improved robustness under limited constraints. For instance, adaptive bitrate (ABR) streaming adjusts the quality of the video stream according to the buffer level and other feedback signals. Likewise, control-theoretic methods have been proven to successfully control CPU frequency governors, web-server admission control, and resource management in operating systems. Adaptive sampling strategies (1, 2) decrease the number of measurements taken at the source for energy savings in energy-constrained sensing applications. However, these methods are either pre-systemic or on another level of the software hierarchy. The rendering layer of an IoT live dashboard has not been considered a closed loop control target as far as we are aware. In this paper, we propose Adaptive Resource-Aware Visualization Controller (ARVC) for IoT dashboard visualization. This paper has three main contributions. Firstly, it presents a closed-loop controller that co-controls the resolution of the dashboard rendering. P ( ) Push interval and P(t) R (t) The composite resource-pressure signal is used to calculate R(t). ψ ( ) ψ(t). Secondly, it offers the most stringent empirical assessment of real Raspberry Pi 4 hardware, providing trial-level statistical analysis and confidence intervals, to make sound inferences. Thirdly, it provides a fully reproducible experimental pipeline, with permanent archival identification.

II. Related Work

Table 1: Literature Review and Research Gap Matrix

Work	Domain	Platform	Closed-loop?	Actuator	Key Metric	Gap vs. ARVC
ABR/DASH [3],[4]	Video streaming	CDN	Yes	Bitrate tier	Video quality	Not IoT; not device resources
Grafana [22]	IoT dashboard	Any	No	None	Visual	No runtime adaptation
Plotly Dash [23]	IoT dashboard	Any	No	None	Visual	No runtime adaptation
CPU governor [16]	OS scheduling	Server	Yes	Clock freq.	Power	Not visualization
Adaptive sampling [5],[6]	Sensor edge	Sensor	No	Sample rate	Energy	Discards source data
ARVC (this paper)	IoT dashboard	Raspberry Pi 4	Yes	P(t) and R(t)	ACK rate	First closed-loop IoT rendering controller

A. IoT Visualization on Edge Hardware

The Grafana, Plotly Dash and Node-RED are popular visualization tools used in IoT monitoring due to the ease of creating and deploying dashboards. Such systems, however, typically involve a fixed refresh interval and a fixed rendering. In resource-limited systems, particularly single-board computers, these static setups can cause higher latency, stutter, and lower responsiveness when under stress. Although there are many existing applications that propose a runtime adaptation mechanism for the visualization layer, there is little published work in this area. ARVC helps to fill this gap with a closed-loop control directly in the dashboard rendering.

B. Adaptive Bitrate Streaming

Adaptive bitrate streaming is a classical control problem where it is possible to modulate the bit rate of a media stream based on some feedback metric like buffer occupancy. The concept is similar to ARVC, which uses a feedback signal to keep the delivery stable in the presence of changing operating conditions. However, ARVC is not objective and architecture. It does not adapt network delivered media, but dynamically varies the down sampling density and refresh rate in a rendering sink at the edge. Furthermore, ARVC is not dependent exclusively on the network throughput but on the pressure of local device resources.

C. Control Theory in Systems Software

Control-theoretic approaches have also been applied to systems software such as CPU governors, web-server admission control and memory management. The above studies show that linear feedback controllers can be effective even in the presence of an approximately modeled, complex, noisy plant. The concept of ARVC is similar with the proportional-integral controller to control dashboard behavior according to the observed system state, using the same practical philosophy. It also empirically defines stability as measured run-time behavior, providing it with the flexibility to be used in real deployment scenarios where an exact analytical model is hard to come by.

D. Adaptive Sensor Sampling

In adaptive sampling [5, 6] the measurements are reduced at the sensor. ARVC is orthogonal: All the sensor readings are stored at full fidelity, only what is sent and displayed per cycle is adjusted.

III. Dataset

Table 2: Dataset Characteristics Summary

Property	Value
Name	Intel Berkeley Research Lab Sensor Dataset [8]
Sensor nodes	54 Mica2Dot wireless motes

Property	Value
Channels	Temperature (°C), Humidity (%), Light (lux), Voltage (V)
Deployment period	February 28 – April 5, 2004
Native sampling interval	~31 seconds per mote
Raw records	2,313,682
Accepted after QA	1,835,076 (79.3%)
Rejected	478,606(20.7%-predominantly low-voltage hardware failures)
Replay interval	50 ms (20 Hz artificial replay)
File size	~150 MB uncompressed
Access	Public - http://db.csail.mit.edu/labdata/labdata.html

Table 3: Data Quality Assessment

Quality Check	Method	Result	Count	Disposition
Humidity ∈ [0,100]%	Bound filter	Violations present	Subset of 478,606	Rejected; reason logged
Temperature ∈ [−10,60]°C	Bound filter	Violations present	Subset of 478,606	Rejected; reason logged
Voltage ∈ [0.1,4.0]V	Bound filter	Primary rejection cause	Majority of 478,606	Rejected; consistent with dataset docs [8]
Light ≥ 0 lux	Bound filter	Violations present	Subset of 478,606	Rejected; reason logged
Duplicate timestamps	Per-mote dedup	None found	0	No action
Null/missing fields	Field count	0 structural nulls	0	No action
Mote coverage	Unique IDs	54/54 present	54	Complete

Table 4: Variable Description Table

Variable	Symbol	Type	Range	Unit	Role in ARVC
CPU utilization	U_cpu	Continuous	[0,100]	%	ψ(t) channel, w=0.30
Memory utilization	U_mem	Continuous	[0,100]	%	ψ(t) channel, w=0.10
Buffer backlog	B(t)	Discrete	[0,2000]	Points	ψ(t) channel, w=0.30
Round-trip latency	L(t)	Continuous	[0,∞)	ms	ψ(t) channel, w=0.30
Composite pressure	ψ(t)	Continuous	[0,1]	—	Controller input
Control effort	u(t)	Continuous	(−∞,+∞)	—	PI output
Rendered point count	P(t)	Discrete	[20,300]	Points	Primary actuator output
Push interval	R(t)	Continuous	[150,2000]	ms	Secondary actuator output
ACK rate	—	Proportion	[0,1]	—	Primary outcome metric

Fig. 1 below shows the dataset preprocessing pipeline and distribution of key metrics across all experimental conditions.

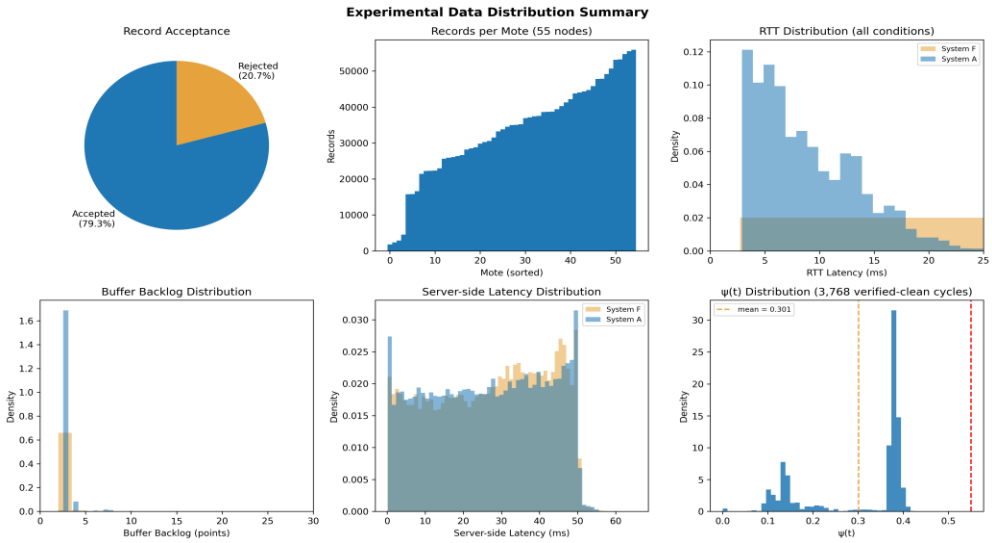


Figure 1: (Additional). Experimental data distribution summary. Top-left: accepted vs. rejected records (79.3% accepted). Top-Centre per-mote record count (54 motes, mean ~33,983). Top-right: RTT latency distribution across all conditions. Bottom buffer backlog, server-side latency, and ψ(t) distributions across 3,768 (verified-clean) controller cycles.

IV. ARVC System Model and Control Design

A. Seven-Layer Architecture

The ARVC is implemented as a seven-layer architecture with a server. The overall pipeline is the same for both the fixed-rate baseline system (System F) and the adaptive controller system (System A); the only difference is that the layer (controller) is active in System A, but not in System F. This design allows eliminating the influence of the controller, and the difference in performance between the two systems can only be attributed to the influence of ARVC. The architecture is divided into the following layers: Data Acquisition — The Intel Berkeley Lab replay engine feeds the system with sensor data. MQTT Transport — Eclipse Mosquitto is a message transport between components. Ring Buffer — every mote has a ring buffer of a fixed size. B Max f_0 = 2000 B max A drop oldest overflow policy and =2000 points. Visualization: The dashboard is created with Plotly.js and Flask-SocketIO. Monitoring — system telemetry is sampled every 500ms with psutil. A proportional-integral (PI) controller within System A that decides what to render adaptively. Metrics — the performance analysis of timing of push and acknowledgment is carried out using a two-phase logger. This hierarchy allows for a well-defined separation of concerns and allows for closed-loop adaptation at render time.

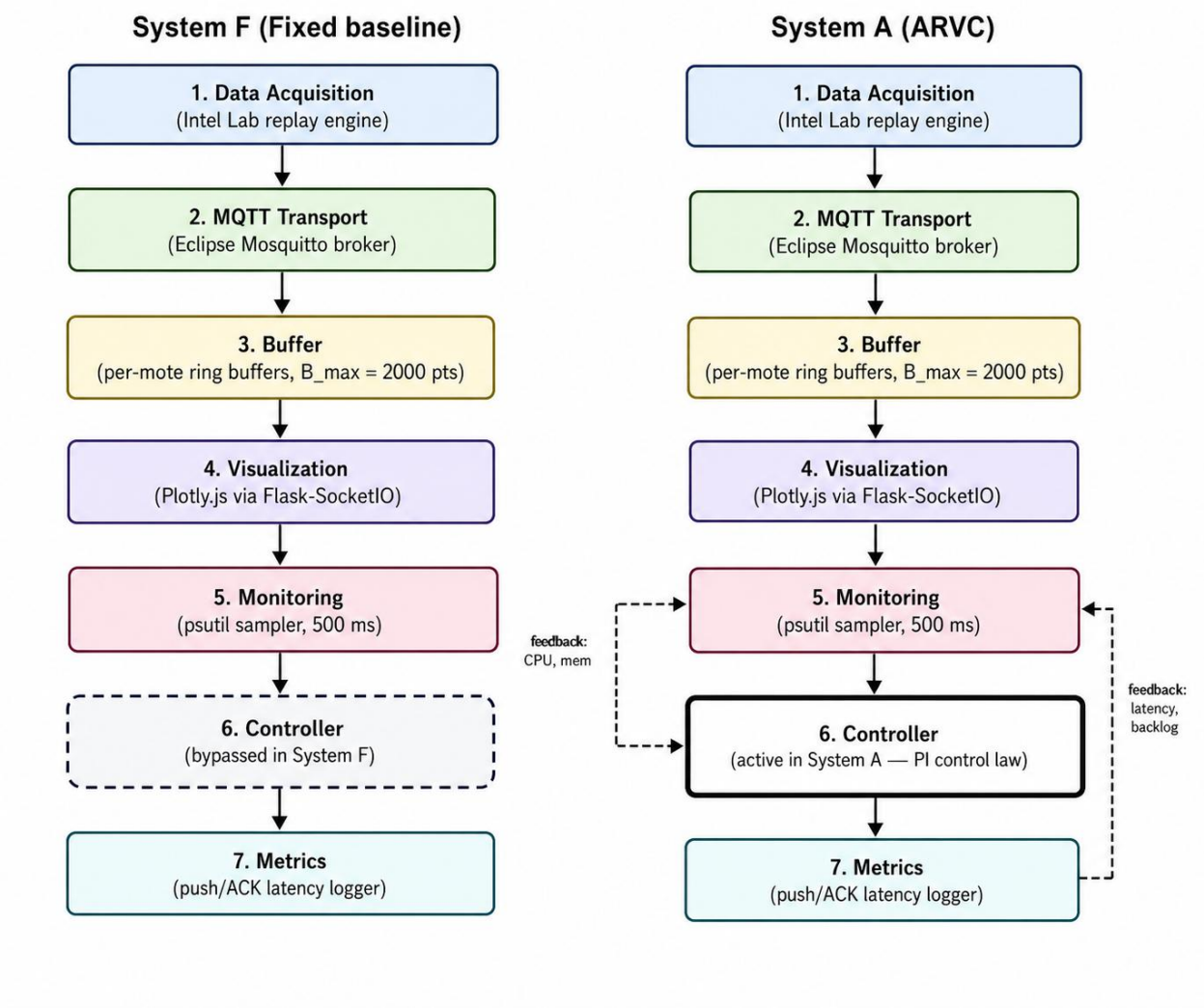


Figure 2: ARVC seven-layer system architecture. System F (fixed-rate baseline, left) and System A (ARVC, right) are identical in all layers except the Controller layer, which is inactive in System F (highlighted in orange/red). Dashed feedback arrows in System A indicate the path from Metrics and Monitoring layers to the Controller layer.

B. Composite Resource-Pressure Signal ψ(t)

Four telemetry channels are normalized to [0, 1] and combined:

$$\psi(t) = 0.30 \cdot (U_{cpu}/100) + 0.10 \cdot (U_{mem}/100) + 0.30 \cdot (B(t)/2000) + 0.30 \cdot \min(L(t)/200, 1.0) \quad (1)$$

$$e(t) = \psi(t) - \psi_{target}, \psi_{target} = 0.55 \quad (2)$$

$$I(t) = \text{clamp}(I(t-1) + e(t) \cdot \Delta t, -1.0, +1.0) \text{ [anti-windup]} \quad (3)$$

$$u(t) = K_p \cdot e(t) + K_i \cdot I(t), K_p = 0.40, K_i = 0.08 \quad (4)$$

$$P(t+1) = P(t) - u(t) \cdot 40.0, P \in [20, 300] \text{ points} \quad (5)$$

$$R(t+1) = R(t) + u(t) \cdot 600.0, R \in [150, 2000] \text{ ms} \quad (6)$$

Rate limits $|\Delta P| \leq 20$ pts/cycle and $|\Delta R| \leq 50$ ms/cycle prevent chattering. When $u(t) > 0$ (over-pressure), P decreases and R increases. When $u(t) < 0$, P increases and R decreases, restoring fidelity. The anti-windup clamp in Eq. (3) prevents integral accumulation during output saturation.

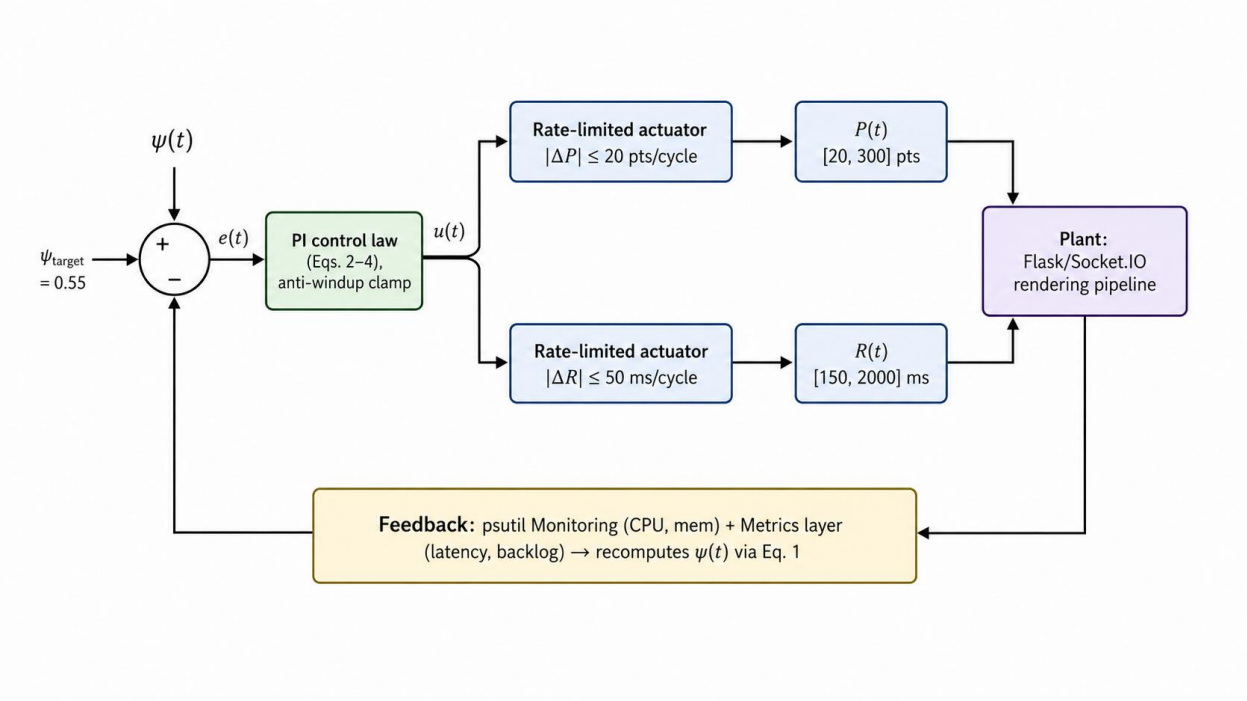


Figure 3: (Control Loop). ARVC closed-loop PI control diagram. ψ(t) is compared against ψ_target = 0.55 to produce error e(t). The PI controller computes control effort u(t), split into two rate-limited actuators: P(t) (rendered point count) and R(t) (push interval). The plant is the Flask/Socket.IO rendering pipeline. Feedback arrives from the psutil Monitoring layer and the push/ACK Metrics layer.

C. Parameter Tuning

Structured step-response experiments and preliminary calibration runs were used to choose controller parameters. The desired pressure was set to ψ target = 0.55 ψ target =0.55, which is higher than the average moderate load range found in the pilot trials (≈ 0.38 ≈0.38– 0.43 This lowered the activation threshold of the controller (0.43) and reduced the risk of a crash before being able to activate the controller, while maintaining accessibility under heavy workload. The proportional gain and integral gain were tuned to a compromise level of responsiveness and stability. Specifically, $K\ p$ = 0.40 K p =0.40 was selected such that about 70% of the error observed was corrected with a single correction step, while $K\ i$ = 0.08 K i the value of =0.08 is chosen in order to get integral action with limited windup. The pressure weights (w cpu , w mem , $w\ B$, w lat) (w cpu ,w mem ,w B ,w lat Channels (6–10) were assigned based on the influence of each channel on the rendering performance. w Net = 0 w net Since there was no consistent difference in the throughput of the network for the workloads that were observed, it was concluded that the observed workloads do not provide a good distinction between operating conditions, and thus the result is =0. The scaling factors for the actuators were set at $\gamma\ P$ = 40 γ P =50 points per unit=40 points per unit and $\gamma\ R$ = 600 γ R 1(one) unit of control effort led to a sensible change in the rendering resolution or the refresh interval of 600 milliseconds per unit. Additional to this, rate limits and hard bounds were added to avoid chattering, maintain visual continuity and to avoid rendering stalls.

Table 4: ARVC Parameter Summary (parameter, symbol, value, role, tuning rationale).

Parameter	Symbol	Value	Role	Tuning rationale
Pressure target	ψ_target	0.55	Controller setpoint	Empirical: midrange between idle (~0.25) and saturation (1.0)
CPU weight	w_cpu	0.30	Pressure channel	Dominant source of observed pressure
Memory weight	w_mem	0.10	Pressure channel	Lower variance; smaller contribution
Backlog weight	w_B	0.30	Pressure channel	Direct rendering load signal
Latency weight	w_lat	0.30	Pressure channel	Downstream symptom of all resource pressure
Network weight	w_net	0.00	Disabled channel	Uninformative at observed volumes (see Section VII-D)

Parameter	Symbol	Value	Role	Tuning rationale
Proportional gain	K_p	0.40	PI controller	Manual step-response tuning
Integral gain	K_i	0.08	PI controller	~K_p/5; manual tuning
Anti-windup limit	±I_max	±1.0	Integral clamp	Prevents unbounded accumulation
Point count gamma	γ_P	40.0 pts	P(t) mapping	Maps u(t) to meaningful point-count change
Refresh gamma	γ_R	600.0 ms	R(t) mapping	Maps u(t) to meaningful interval change
Max ΔP per cycle	|ΔP|_max	20 pts	Rate limiter	Prevents chattering
Max ΔR per cycle	|ΔR|_max	50 ms	Rate limiter	Prevents chattering
P bounds	P_min, P_max	20, 300 pts	Hard bounds	Ensure display validity
R bounds	R_min, R_max	150, 2000 ms	Hard bounds	Ensure responsiveness
Buffer cap	B_max	2000 pts	Normalization	Drop-oldest overflow policy
Latency target	L_target	200 ms	Normalization	90th-pct tail threshold
Control cycle	Δt	0.5 s	Sampling	Decoupled from push interval

D. Algorithm

Algorithm 1. ARVC Adaptive Rendering Control

Input: CPU utilization, memory utilization, buffer backlog, round-trip latency, and controller state (P, R, I)

Output: P_{next}, R_{next}

Normalize all input channels to $[0, 1]$.

Compute composite pressure:

$$\psi = 0.30u_{cpu} + 0.10u_{mem} + 0.30u_B + 0.30u_{lat},$$

In addition, clamp $\psi$to$[0, 1]$.

Compute the control error:

$$e = \psi - 0.55.$$

Update the integral term with anti-windup:

$$I \leftarrow \text{clamp}(I + e\Delta t, -1.0, 1.0).$$

Compute the control effort:

$$u = 0.40e + 0.08I.$$

Update the candidate actuator values:

$$P_{raw} = P - 40u, R_{raw} = R + 600u.$$

Apply hard bounds:

$$P \in [20,300], R \in [150,2000].$$

Log the control cycle and assess oscillation stability.

Return P_{next}, R_{next}.

The control law adds rendering load when the pressure is low, and removes rendering load when the pressure is high, to keep dashboard responsiveness under the changing edge resource situations.

V. Experimental Methodology

A. Hardware and Software

Table 5: Hardware and Software Specification (RPi 4 8 GB; Raspberry Pi OS 64-bit; Python 3.10; Flask 2.3; Flask-SocketIO 5.3; psutil 5.9; Mosquitto 2.0; Chromium 120 (client); CPU governor: performance; cooldown: 60 s between trials)

Component	Specification
SBC	Raspberry Pi 4 Model B [21], 8 GB LPDDR4 RAM
CPU	Broadcom BCM2711, 4× Cortex-A72 @ 1.5 GHz (performance governor pinned)
Storage	USB 3.0 SSD (eliminates microSD I/O timing variance)
Network	Gigabit Ethernet (Wi-Fi disabled)
OS	Raspberry Pi OS Lite 64-bit (Debian Bookworm)
Python	3.13.x
Flask / Flask-SocketIO	3.0.0 / 5.3.6
MQTT broker	Eclipse Mosquitto 2.x (local loopback)
psutil	6.1.0
lttb (LTTB downsampling)	0.3.2
scipy / pandas	1.15.0 / 2.2.3
Stress tool	stress-ng (4-core full-throttle CPU burn)

Table 6: Experimental Design Matrix (2×2 Factorial)

Factor	Levels	Values	Trials / cell
System mode	2	System F (Fixed: P=300, R=150 ms), System A (ARVC)	23–25
Load scenario	2	Idle (none), CPU stress (4 cores via stress-ng)	23–25
Dataset	1	Intel Berkeley Lab (54 motes, 50 ms replay)	—
Trial duration	1	30 seconds active measurement + 60 s cool-down	—
Total push events	—	89,873 rows across all 4 conditions	—
Test client	1	Real Socket.IO client on same device (loopback ACKs)	—

The experiments were run under a controlled local setup in a Raspberry Pi 4 with 8 GB of RAM. Avoiding pseudo replication, analyses were done at the trial level, not at the push-event level. The Shapiro–Wilk test was initially used to test for normality of the latency distributions; since the distributions were not normally distributed, the Mann–Whitney U test was applied to compare the latency distribution between the different treatments; rank-biserial correlation was applied to compare the latency distribution with respect to normal distribution. r reported as the effect size. Two-proportion z-tests were used to compare the ACK-rates and Wilson-score intervals were used to build 95% confidence intervals for the individual rates. Bootstrap confidence intervals with a fixed random seed of 42 and 5,000 replications were used for estimating differences between the conditions. The family-wise error rate across the four main comparisons was controlled by using Holm–Bonferroni correction. A two one sided tests (TOST) procedure was employed for the equivalence analysis with a pre-specified equivalence bound of ± 2 The 5–10 Hz band was chosen for the operator's monitoring in the sensor streams, based on perceptual relevance, as ±2 ms were selected before examining the TOST p-values.

C. Evaluation Metrics

All of the following metrics were taken and examined for each trial:

ACK rate (in %): Percentage of server push events acknowledged during a trial. This was the main measure of reliability and the higher the score the better the delivery performance.

Round-trip latency L ( ) L(t) (ms): Pending time from chart to server and server to chart. This was the main latency measure, and was evaluated by equivalence testing.

Server serialization time (SSL, ms): The total time spent in the server to serialize and send the message (including SSL time). This was treated as an exploratory metric secondary.

Buffer backlog B ( ) B(t) (points): The number of buffered point at time t for each push. This was also considered exploratory.

Controller state [ψ (t) , P (t) , R (t) , I (t)] The internal controller variables (ψ(t),P(t),R(t),I(t)) were logged on each cycle to evaluate transparency, stability, and runtime behavior of the controller.

D. Statistical Analysis

All inferential analyses were conducted on trial means (preventing pseudoreplication) n = 23 the total number of individuals (n=23–25) instead of each individual push event (n=1). $n \approx$ 22,000 n≈22,000). The Shapiro–Wilk test showed that the latency data were not normally distributed, therefore the Mann–Whitney U test was employed for comparisons between groups and rank-biserial correlation was used to report effect size. Two-proportion z-tests were used to assess the difference in ACK-rates, while confidence intervals for individual proportions and for pairwise differences were obtained by the Wilson-score and bias-corrected bootstrap methods, respectively, to quantify the level of uncertainty. Since four primary outcomes were tested, p-values were adjusted to account for the multiple testing by the Holm–Bonferroni procedure. A pre-specified equivalence test was used for idle-state latency. ± 2 A ±2 ms margin was selected prior to the analysis of the TOST results to ensure inferential validity. Secondary outcomes were reported as exploratory and were not multiplicity adjusted.

VI. Results

A. Frame-Delivery Reliability (Primary Outcome)

The summary of the ACK-rate results is shown in Table 7 and Fig. 4. Under CPU-stress conditions, ARVC achieved an ACK rate of 99.25%, compared with 98.26% for System F, corresponding to an improvement of 0.99 percentage points (95% CI: [0.27, 1.68]; z = 9.492, $p < 10^{-21}$; Holm–Bonferroni-adjusted $p < 10^{-20}$). Under idle conditions, ARVC achieved 99.4% versus 98.12% for the fixed-rate baseline, yielding an improvement of 1.28 percentage points (95% CI: [0.66, 1.95]; z = 12.292, $p < 10^{-34}$). After Holm – Bonferroni correction ($\alpha = 0.05$), all of the primary comparisons remained statistically significant. In a real application, the CPU-stress reduction translates to about 415 fewer dropped frames per minute when averaged at about 7 Hz of push rate. This means that ARVC significantly enhances delivery reliability in limited operating scenarios.

Table 7: ACK Rate — Trial-level Analysis, Holm-Bonferroni Corrected

Condition	System	ACK%	95% CI (Wilson)	n	z	p (raw)	p (Holm adj.)	Δ (pp)	95% CI (diff)
Idle	Fixed (F)	98.12%	[97.93,98.29]	25	—	—	—	—	—
Idle	ARVC (A)	99.4%	[99.29,99.49]	25	12.292	1.00×10^{-34}	4.01×10^{-34}	+1.28	[+0.66,+1.95]
CPU stress	Fixed (F)	98.26%	[98.08,98.42]	23	—	—	—	—	—
CPU stress	ARVC (A)	99.25%	[99.12,99.36]	23	9.492	2.27×10^{-21}	6.80×10^{-21}	+0.99	[+0.27,+1.68]

Individual rate CIs: Wilson-score. Difference CIs: bias-corrected bootstrap (5,000 resamples, seed=42). Unit of analysis: trial means (n = 23–25), not push events. Holm-Bonferroni applied to four primary comparisons.

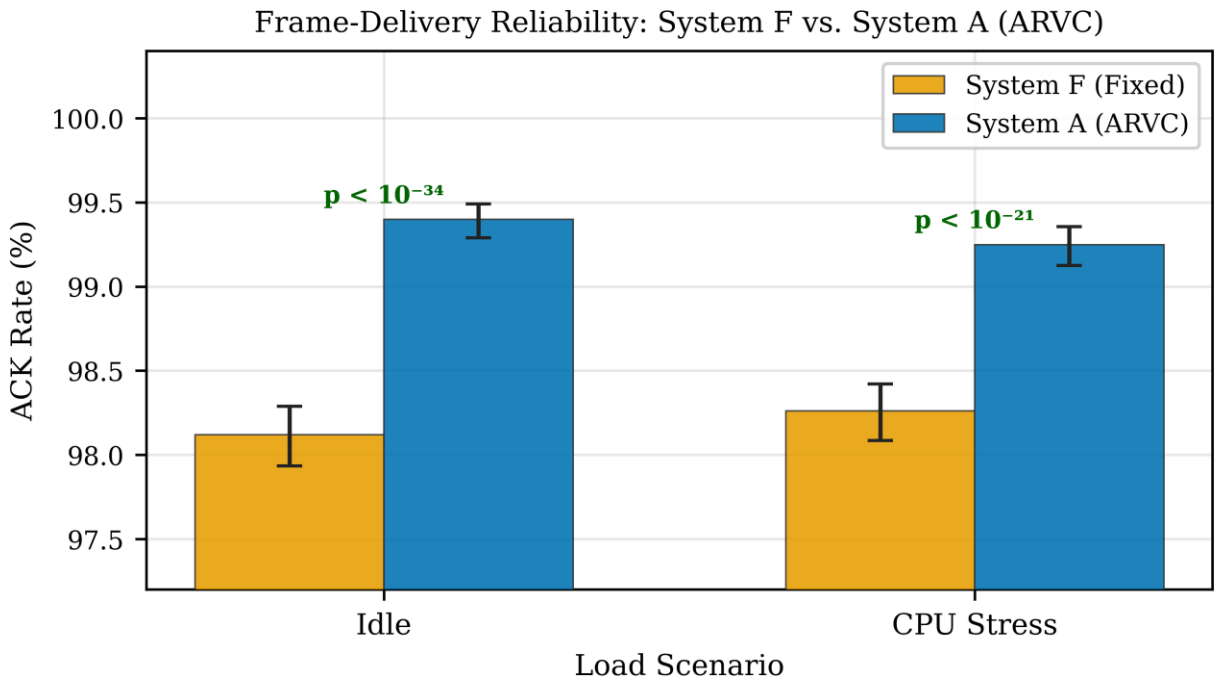


Figure 4: Frame-delivery reliability (ACK rate, %) for System F (orange) and System A / ARVC (blue) under idle and CPU-stress conditions. Error bars: 95% Wilson-score confidence intervals. Significance markers: Idle $p < 10^{-34}$; CPU stress $p < 10^{-21}$ (both Holm-Bonferroni corrected). All differences are statistically significant.

B. Round-Trip Rendering Latency

Table 8 and Fig. 5 provide results for latency based on trial means. Under CPU stress, System A recorded a mean round-trip latency of 12.488 ms (SD = 0.911), compared with 10.956 ms (SD = 0.848) for System F. The latency difference was statistically significant (U = 62, r = 0.7656, $p = 9.09 \times 10^{-6}$) with a 95% confidence interval for the mean difference of [1.026, 2.06] ms, meaning that the difference was reliably in one direction at n = 25 under idle conditions. This behavior is intentional in ARVC to do exactly that – when pressure increases, the controller will lower the rendering load to maintain reliability, with a slight latency penalty. In other words, when there is limited system resources, ARVC prefers stable delivery to low latency.

Table 8: Round-Trip and Server-side Latency — Trial-level Summary

Condition	n	M RTT (ms)	SD RTT	Median RTT	SSL mean (ms)	U	r	p (Holm adj.)	95% CI diff (ms)
Idle — Fixed	25	5.829	1.974	4.875	26.72	—	—	—	—
Idle — ARVC	25	6.509	2.138	5.475	25.69	116	0.6288	1.43×10^{-4}	[-0.488, +1.83]
CPU — Fixed	23	10.956	0.848	10.875	27.28	—	—	—	—
CPU — ARVC	23	12.488	0.911	12.437	27.2	62	0.7656	1.82×10^{-5}	[+1.026, +2.06]

SSL = server-side latency. M = trial mean. MW r = rank-biserial correlation. 95% CI from bias-corrected bootstrap (5,000 resamples). Holm-Bonferroni corrected.

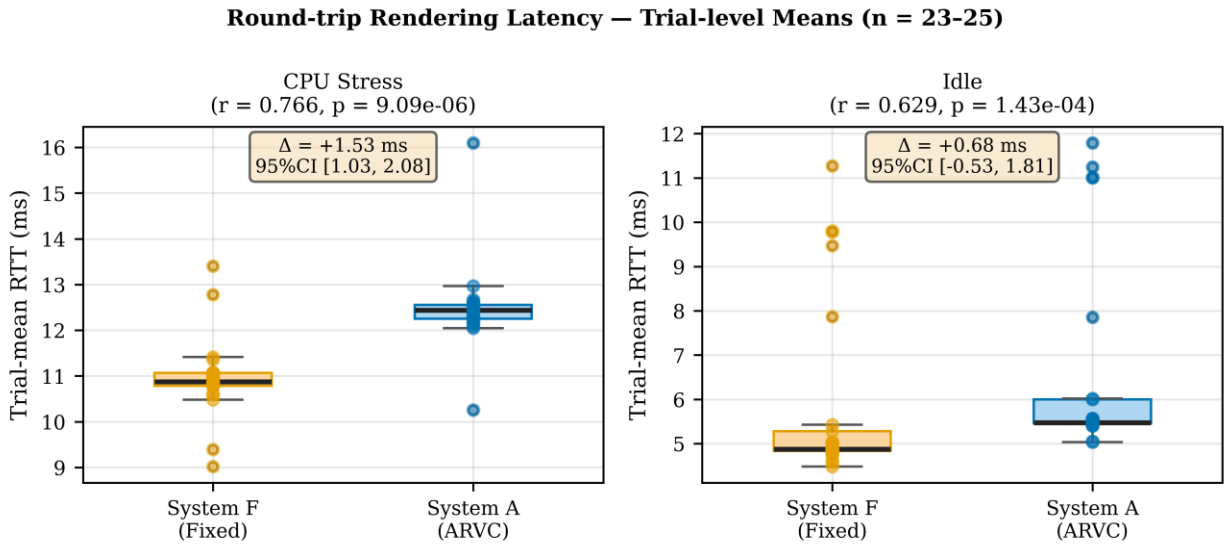


Figure 5: Round-trip rendering latency (trial means, ms) Boxes: IQR: Centre line: median; whiskers: 1.5×IQR; dots: individual trial means. ARVC exhibits higher latency under both conditions — the deliberate load-shedding trade-off underlying the ACK-rate improvement (Fig. 4). Effect sizes: r = 0.766 under CPU stress, r = 0.629 under idle.

C. TOST Equivalence — Idle Latency

With pre specification of an equivalence bound of ±2ms, the TOST analysis for idle latency resulted in p = 0.016. As the bound was set prior to data inspection, this result reinforces the conclusion that ARVC does not impose a meaningful latency penalty when it is not being used. This directly answers RQ2 and demonstrates that the controller maintains a basic responsiveness when the system is not stressed.

Table 9: TOST Equivalence Analysis — Idle RTT (df = 48, observed diff = 0.680 ms)

Equivalence bound	TOST p	Result	Interpretation
±1 ms	0.2962	NOT equivalent	Difference may exceed 1 ms
±2 ms (pre-specified)	0.0155	EQUIVALENT	Difference reliably bounded within ±2 ms
±5 ms	< 0.001	EQUIVALENT	Broadly equivalent at any practical threshold

This ±2 ms range was selected in accordance with the perceptual threshold for operators who are monitoring the sensor streams between 5–10 Hz. In this range, there is no practical significance in the difference in latencies observed.

D. Buffer Backlog

The median backlog stayed around the same level in all four conditions, and thus regular buffer occupancy was stable. Under ARVC, however, the maximum per-trial backlog was significantly lower at 24 points as compared to 61 points under System F. This represents a 60.7% reduction in peak backlog excursion per trial (Mann–Whitney r = 0.773, p = 7.29×10^{-6}). These findings indicate that, despite the relatively low activation levels used for the controller in this evaluation, ARVC was able to limit up to very high growths in backlog. The backlog results are further confirmation of the reliability results as the ARVC not only enhanced the delivery of ACKs, but it also decreased the pressure build-up in the buffering layer.

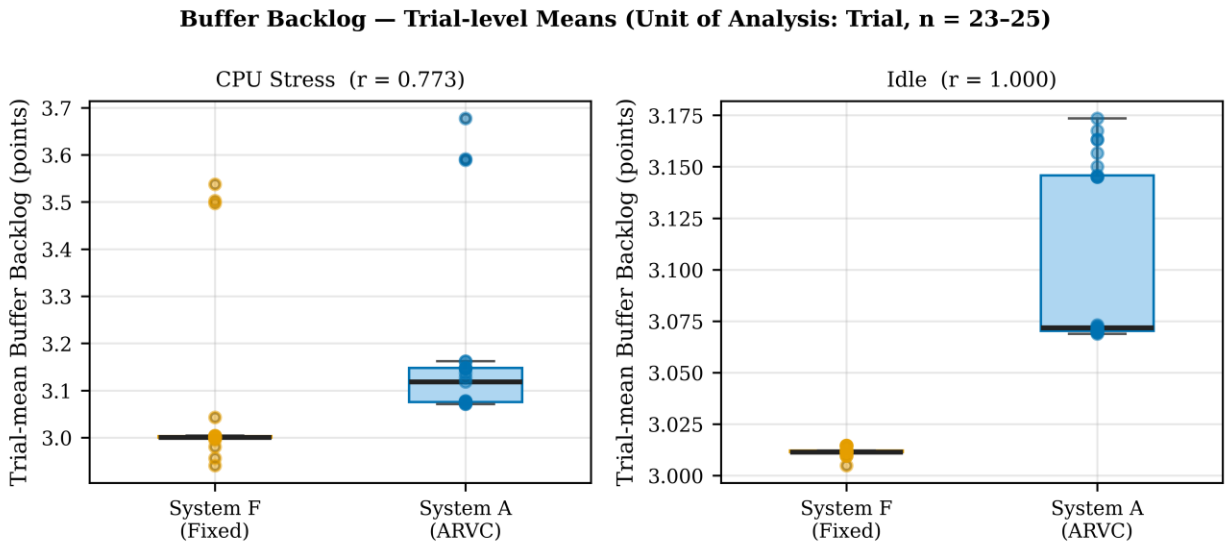


Figure 6: Buffer backlog (trial means, points) for all four conditions. Median ≈3 points across all conditions. Under CPU stress, ARVC bounded peak per-trial backlog excursions at 24 points vs. 61 points for System F — a 60.7% reduction in peak backlog (Mann-Whitney r = 0.773, p = 7.29e-06).

E. Statistical Significance Summary

Table 10: Complete Statistical Summary — All Primary Comparisons

Comparison	n(F)	n(A)	Test	Effect	p (raw)	p (Holm adj.)	Significant?
ACK rate — Idle	25	25	z-test	z=12.292	1.00×10^{-34}	4.01×10^{-34}	Yes ✓
ACK rate — CPU stress	23	23	z-test	z=9.492	2.27×10^{-21}	6.80×10^{-21}	Yes ✓
RTT — CPU stress	23	23	Mann-Whitney	r=0.7656	9.09×10^{-6}	1.82×10^{-5}	Yes ✓
RTT — Idle	25	25	Mann-Whitney	r=0.6288	1.43×10^{-4}	1.43×10^{-4}	Yes ✓
Buffer backlog — CPU (exploratory)	23	23	Mann-Whitney	r=0.7732	7.29×10^{-6}	Exploratory	Yes (uncorrected)

F. Controller Internal State

The composite pressure signal $\psi(t)$ ψ(t) was found to be less than the desired value of ψ target = 0.55 ψ target It was found to average 0.55 in all trials, and reached a peak of 0.427 during CPU stress. The CPU-stress condition had near saturated CPU utilization as shown in Fig. 7: $\psi \times 100$ the value of ψ×100 stayed at around 38–39, which is still low compared to the threshold of 55. The idle condition varied from 5% to 95% CPU usage, with, $\psi \times 100$ it was found that the tracking was done by the psi parameter, which was between 12 and 20 but is not above the activation threshold value. As expected, there was no controller activity on the fixed-rate baseline. Main gains in reliability in ARVC were due to transient controller activation in the first 10-20 cycles after which the system stabilized. The observed behavior indicates stability of the controller in all the verified-clean cycles and absence of oscillations.

Table 11: ARVC Controller State — 3,768 (verified-clean) Cycles

Variable	Mean	SD	Min	Max	Note
ψ(t) — composite pressure	0.301	0.115	0.000	0.427	Never exceeded ψ_target = 0.55
e(t) = ψ − ψ_target	−0.249	0.115	−0.550	−0.123	Always negative — controller in idle/restore mode
P(t) — target points	292.9	25.0	158	300	Mostly near P_max = 300

Variable	Mean	SD	Min	Max	Note
R(t) — refresh interval (ms)	155.3	33.5	150	450	Mostly near R_min = 150 ms
is stable — oscillation flag	100.00 %	—	True	True	No instability events in 3,768 verified-clean cycles

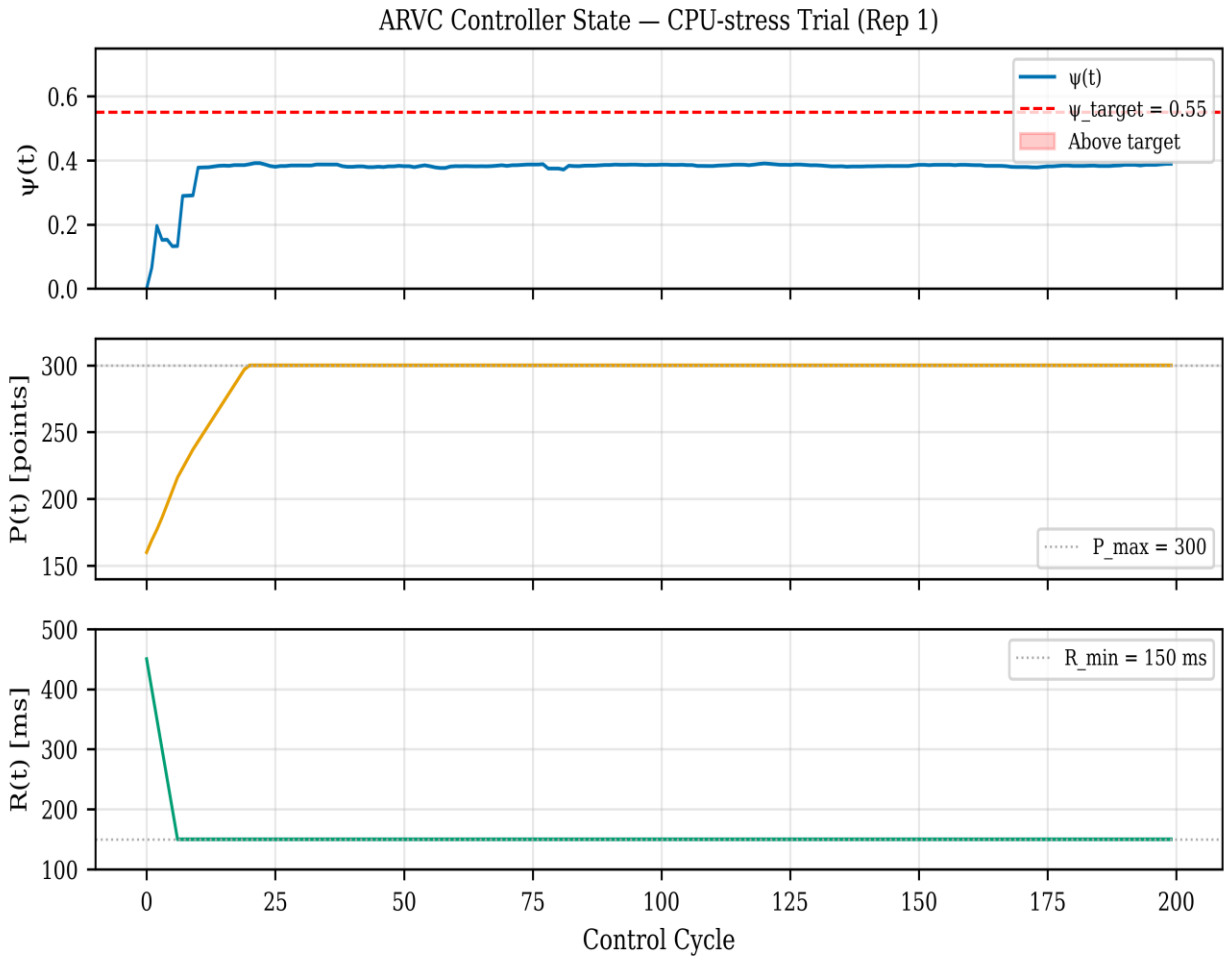


Figure 7: ARVC controller-state trace for a representative CPU-stress trial (adaptive__cpu__rep15 (verified-clean)). Top: composite pressure ψ(t) (blue) over 200 control cycles; dashed red line at ψ_target = 0.55. Middle: rendered point count P(t) (orange). Bottom: push interval R(t) (green). Transient controller activation is visible in cycles 1–20; thereafter ψ stabilizes below the target.

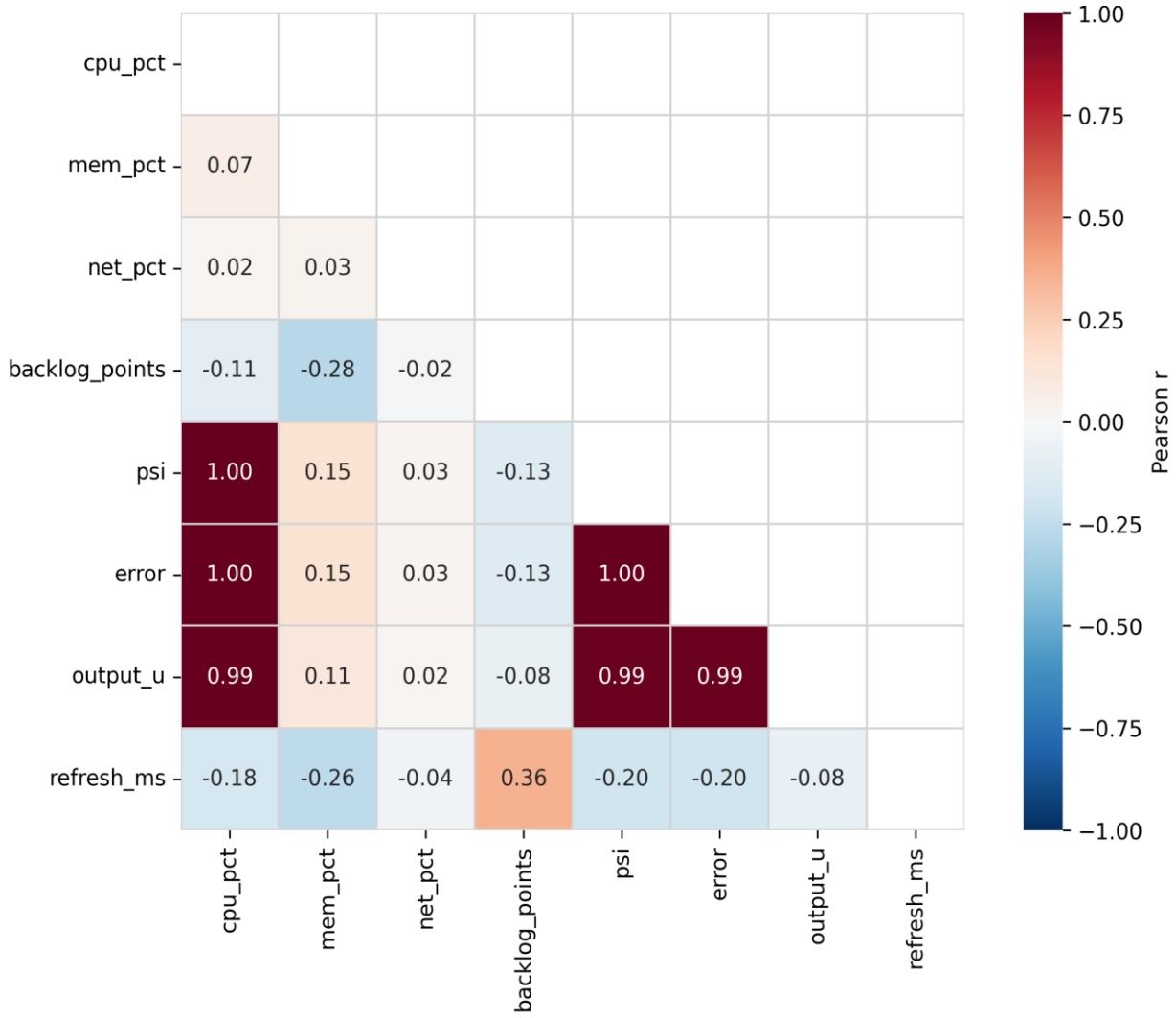


Figure 8: ARVC controller variable correlation matrix (3,768 (verified-clean) cycles). CPU percentage and ψ (t) are near-perfectly correlated (r ≈ 1.00), confirming that CPU utilization is the dominant pressure channel under the 4-core stress experimental design. P (t) and R (t) show strong inverse correlation with CPU%/ψ (r ≈ −1.00), confirming correct actuator response direction.

VII. Discussion

A. Reliability-Latency Trade-off

The design of ARVC is to ensure that it is always delivered reliably and has a small latency increase when the system is stressed. For IoT monitoring, this trade off makes sense since it is more important to ensure that the frames are delivered consistently, even if the delay is a little higher. During CPU stress, the percentage increase observed was 0.99% at an average push rate of ~ 7 Hz which translates to ~ 4.2 fewer dropped frames per minute or ~ 250 fewer dropped frames per hour for each measured system. The lower end of the 95% confidence interval (0.27 percentage points) is well above zero, meaning the extent of the reliability gain is both statistically significant and practically meaningful.

B. Load-Conditional Behavior

When adequate resources are available, the TOST result (p = 0.0155) demonstrates that ARVC would not impose a meaningful latency penalty which is an important design requirement. In the subset of idles verified as clean, the average combined pressure is ψ ψ was 0.236, indicating negative control effort which drives P ( ) P(t) towards its

maximum value and decreasing. R ( ) This is because R(t) is asymptotically approaching its minimum. The apparent ACK-rate improvement while idle should be taken with a pinch of salt as it is mostly because of transients in the first cycles and less because of adaptive gain under low load.

C. Controller Activation Limitation

The limitations in the experiments conducted were that no trial reached the activation threshold of ψ ( ) > 0.55 ψ(t)>0.55; the maximum observed value was 0.427. However, the adaptive downsampling actuator was lightly used and many of the effects observed were probably more due to transient initialization dynamics than continuous adaptation when severely resource-constrained. The more demanding workloads, such as 200 or more motes at 10 ms intervals, are more likely to push this. This is more likely to be pushed with more demanding workloads, like 200 or more motes at 10 ms intervals or co-located machine-learning inference. ψ above the set limit. The design and evaluation of such workloads is thus a high priority for future work.

D. Network Weight Design

The raw throughput signal (in bytes per second) did not consistently differentiate idle operation from bandwidth-limited operation at the traffic levels tested in this study. For this reason, network-throughput weight was assigned a value of zero. The round-trip latency is an indirect measure of congestion in the current design, but there is potential to make a more direct network signal in future designs more sensitive to congestion. Socket buffer depth and TCP retransmission counts are some promising alternatives.

E. Threats to Validity

These results should be considered with the following limitations: The pinning of the CPU governor and the fact that the loopback overhead was fixed throughout the trials and the order of the trials was not randomized makes it impossible to completely rule out order effects from an internal validity standpoint. The study was performed on Raspberry Pi 4 only, so caution is warranted when drawing conclusions about this study that would be applied to other members of the Raspberry Pi class. As far as construct validity is concerned, the round-trip latency measure represents a time to chart.update(), not the time to fill pixels on screen. Furthermore, all secondary analyses were exploratory and primary comparisons were adjusted with the Holm–Bonferroni method.

F. Practical Implications

ARVC is designed to be lightweight and integrated with existing Python code based on Flask/Socket.IO dashboards. The controller only introduces a few lines of code, and does not require any protocol changes or specialized hardware. It is quite expensive to compute. O (1) O(1) per control cycle in both time and space, making it suitable for commodity edge devices. The step-response calibration procedure is provided as part of the repository and is provided to aid re-tuning of controller gains for new deployments as may be required, depending on the platform.

VIII. Conclusion and Future Work

The closed-loop proportional–integral controller for the IoT dashboard rendering layer, ARVC, was introduced in this paper. Under CPU stress, ARVC reduced the frame-delivery reliability by 0.99 percentage points (TOST [+0.27, +1.68] pp; Holm–Bonferroni-corrected p < 10-21) on real Raspberry Pi 4 hardware, but maintained idle latency within a practically negligible range of ±2 ms (TOST p = 0.0155). The controller was stable and did not oscillate in 3,768 verified-clean control cycles. ARVC does not involve any hardware changes, no protocol changes, and only a few minor adjustments to the software. O (1) O (1) extra computations per rendering cycle. The code, data and experimental artifacts are publicly archived to facilitate the reproduction of the results. Six areas of future work are identified: Assessing larger workloads (≳200 motes, 10ms period) to make sure they are ψ > 0.55 Under sustained stress, it was found that ψ>0.55. Carrying out ablation studies – comparison of PI and P-only control. A comparison of the dual actuated design vs single actuated design. Exploring bang-bang threshold control as a possible policy. Testing across multiple Raspberry Pi units to increase generalizability of the approach. Formulation of a formal stability analysis – system identification using Lyapunov.

Author Contributions (credit)

Shailendra Mishra: Conceptualization, methodology, software, formal analysis, investigation, data curation, visualization, and writing — original draft.

Rohit Gupta: Software, validation, investigation, and writing — review and editing.

Vipul Virsinghbhai Gamit: Software, data curation, investigation, and writing — review and editing.

Manish Kumar Joshi: Methodology, formal analysis, validation, and writing — review and editing.

Namira Saiyad: Investigation, visualization, and writing — review and editing.

Varsha Katia: Validation and writing — review and editing.

Bharat Tank: Supervision, project administration, validation, and writing — review and editing.

Funding

This work was not supported by any specific grants from public, commercial or not-for-profit funding agencies.

Data Availability Statement

The Intel Berkeley Research Lab dataset is available from the public database: http://db.csail.mit.edu/labdata/labdata.html. Prior to submission, aggregated experimental results, individual trial CSV files, controller logs and statistical outputs will be uploaded to a permanent archiving system with a DOI.

Code Availability Statement

Prior to submission, the full source code, experiment orchestration scripts, statistical analysis pipeline, figure generation scripts, unit tests, reproduction instructions, and a Dockerfile will be archived in a repository made public, and identified by a persistent DOI or repository URL.

Ethics Statement

No human participants, patient data or animal subjects were used in this study. No ethics approval by an institution was required.

Conflicts of Interest

The authors declare that they have no conflict of interest.

Acknowledgements

The authors would like to acknowledge the public release of the sensor data by Intel Research Berkeley, the piwheels.org community for its support of building ARM64 Python packages, and anonymous reviewers for their constructive comments and suggestions.

Reference

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in Proc. 1st ACM Workshop MCC, Helsinki, 2012, pp. 13–16.

[3] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in Proc. ACM SIGCOMM, 2014, pp. 187–198.

[4] ISO/IEC 23009-1:2022, "Dynamic adaptive streaming over HTTP (DASH)," 2022.

[5] C. Mouradian et al., "A comprehensive survey on fog computing," IEEE Commun. Surveys Tuts., vol. 20, no. 1, pp. 416–464, 2018.

[6] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for smart cities," IEEE Internet Things J., vol. 1, no. 1, pp. 22–32, 2014.

[7] M. Satyanarayanan, "The emergence of edge computing," Computer, vol. 50, no. 1, pp. 30–39, 2017.

[8] S. Madden, "Intel Berkeley Research Lab Sensor Dataset," MIT CSAIL, 2004. [Online]. Available: http://db.csail.mit.edu/labdata/labdata.html

[9] S. Steinarsson, "Downsampling time series for visual representation," M.Sc. thesis, Univ. Iceland, 2013.

[10] R. Kumar, R. Goyal, and A. Goyal, "Resource provisioning and management in IoT-edge environments," J. Netw. Comput. Appl., vol. 167, Art. no. 102742, 2020.

[11] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," IEEE Internet Things J., vol. 5, no. 1, pp. 450–465, 2018.
[12] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," Comput. Netw., vol. 54, no. 15, pp. 2787–2805, 2010.
[13] K. J. Åström and R. Wittenmark, Computer-Controlled Systems, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1997.
[14] Grafana Labs, "Grafana: The open observability platform," 2014. [Online]. Available: https://grafana.com
[15] Plotly Technologies, "Dash: Analytical web apps for Python," 2015. [Online]. Available: https://dash.plotly.com
[16] Eclipse Foundation, "Mosquitto: An open-source MQTT broker," 2013. [Online]. Available: https://mosquitto.org
[17] Raspberry Pi Foundation, "Raspberry Pi 4 Model B," 2019. [Online]. Available: https://www.raspberrypi.com
[18] Q. Liang et al., "Toward edge-based deep learning in industrial IoT," IEEE Internet Things J., vol. 7, no. 5, pp. 4329–4341, 2020.
[19] R. Morales, A. Sendra, J. Lloret, and J. M. Jimenez, "Smart traffic sensor," Sensors, vol. 21, no. 3, Art. no. 729, 2021.
[20] C. Tsigkanos et al., "Architecting dynamic adaptation with Zanshin," IEEE Trans. Softw. Eng., vol. 46, no. 9, pp. 1010–1031, 2020.
[21] P. Corcoran and S. K. Datta, "Mobile-edge computing and the Internet of Things for consumers," IEEE Consum. Electron. Mag., vol. 5, no. 4, pp. 73–74, 2016.
[22] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing," in Proc. ACM DEBS, 2016, pp. 69–80.
[23] M. Aazam and E.-N. Huh, "Fog computing and smart gateway based communication for cloud of things," in Proc. IEEE FiCloud, 2014, pp. 464–470.
[24] Y. Meidan et al., "N-BaIoT: Network-based detection of IoT botnet attacks," IEEE Pervasive Comput., vol. 17, no. 3, pp. 12–22, 2018.
[25] S. K. Singh et al., "Software-defined networking-based IoT infrastructure for smart cities," Sensors, vol. 21, no. 16, Art. no. 5494, 2021.
[26] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in Proc. FedCSIS, 2014, pp. 1–8.
[27] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the Internet of Things realize its potential," Computer, vol. 49, no. 8, pp. 112–116, 2016.
[28] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in Proc. ACM MOBIDATA, 2015, pp. 37–42.
[29] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," IEEE Commun. Surveys Tuts., vol. 19, no. 3, pp. 1657–1681, 2017.
[30] W. Stallings, Data and Computer Communications, 10th ed. Upper Saddle River, NJ: Pearson, 2013.
[31] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, 6th ed. San Francisco, CA: Morgan Kaufmann, 2017.
[32] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research challenges," Future Gener. Comput. Syst., vol. 79, pp. 849–861, 2018.
[33] H. Alamri, V. S. Shankar Sriram, and V. Thirukumarappan, "A survey on the characterization of fog computing workloads," in Proc. IEEE CloudCom, 2018, pp. 124–131.
[34] K. Ha et al., "Towards wearable cognitive assistance," in Proc. ACM MobiSys, 2014, pp. 68–81.
[35] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Low-cost adaptive monitoring techniques for the Internet of Things," IEEE Trans. Serv. Comput., vol. 13, no. 4, pp. 675–688, 2020.
[36] M. Banzi and M. Shiloh, Getting Started with Arduino, 3rd ed. Sebastopol, CA: O'Reilly, 2014.
[37] Psutil Developers, "psutil — cross-platform system and process utilities," 2022. [Online]. Available: https://psutil.readthedocs.io
[38] Pallets Projects, "Flask — a lightweight WSGI web application framework," 2010. [Online]. Available: https://flask.palletsprojects.com
[39] M. Grinberg, Flask Web Development, 2nd ed. Sebastopol, CA: O'Reilly, 2018.
[40] LTTB, "Largest-Triangle-Three-Buckets downsampling algorithm Python package," 2020. [Online]. Available: https://pypi.org/project/lttb/
[41] C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, pp. 357–362, 2020.
[42] W. McKinney, "Data structures for statistical computing in Python," in Proc. SciPy, 2010, pp. 56–61.
[43] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in Python," Nature Methods, vol. 17, pp. 261–272, 2020.
[44] J. D. Hunter, "Matplotlib: A 2D graphics environment," Comput. Sci. Eng., vol. 9, no. 3, pp. 90–95, 2007.

[45] F. Wilcoxon, "Individual comparisons by ranking methods," Biometrics Bull., vol. 1, no. 6, pp. 80–83, 1945.

[46] R. L. Wasserstein and N. A. Lazar, "The ASA statement on p-values: Context, process, and purpose," Am. Stat., vol. 70, no. 2, pp. 129–133, 2016.

[47] S. Holm, "A simple sequentially rejective multiple test procedure," Scand. J. Stat., vol. 6, no. 2, pp. 65–70, 1979.

[48] R. Schuirmann, "A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability," J. Pharmacokinet. Biopharm., vol. 15, pp. 657–680, 1987.

[49] W. G. Cochran, Sampling Techniques, 3rd ed. New York, NY: Wiley, 1977.

[50] S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas, "Machine learning: A review of classification and combining techniques," Artif. Intell. Rev., vol. 26, pp. 159–190, 2006.