Verified Bidirectional IoT Retrofitting of Legacy Industrial Temperature Controllers Through Modbus Write Readback

Rachmad Andri Atmoko[1], Firman Pratama Dewantara[1], Bayu Sutawijaya[1*], I Dewa Made Widia[1], Salnan Ratih Asriningtias[1], M. Fajar Adiatmoko[2], Akas Bagus Setiawan[3]

[1]*Faculty of Vocational Studies, Universitas Brawijaya, Malang, Indonesia*
[2]*PT. Mokko Otomasi Indonesia*
[3]*Department of Information Technology, Politeknik Negeri Jember, Jember, Indonesia*
*Corresponding author: Bayu Sutawijaya, e-mail: bayu_sutawijaya@ub.ac.id

Article Info: Received: 12-05-2026 Revised: 24-07-2026 Published: 11-08-2026

Abstract
Legacy industrial PID temperature controllers, such as the Autonics TN series (TNH/TNL), are widely deployed to run process cycles in small and medium-scale industrial operations but generally lack native IoT connectivity. While retrofitting such controllers for cloud-based monitoring (the read direction) is comparatively well established, writing process configuration back to the controller reliably, and verifying that the write actually took effect, remains a harder and less-explored problem. This study addresses that gap by proposing and evaluating a register-level write-then-read-back verification protocol for bidirectional deployment of process patterns (PTTN: pattern number, step count, per-step setpoint and duration, and end-state behavior) to a TNH controller over a direct RS-485 serial path (Jalur A), one of two architecturally distinct control paths in the system under study; the other, an ESP32-mediated cloud path (Jalur B), is implemented and compile-verified but not yet validated on physical hardware. Using a Modbus RTU simulator and 100 synthetic patterns ranging from 1 to 20 steps, the protocol achieved 100.00% Pattern Deployment Accuracy (SD = 0.00%), with p50, p90, and p99 end-to-end latencies of 511, 543, and 578 ms, respectively, that did not scale meaningfully with pattern complexity. A subsequent fault-injection experiment (N = 45 trials: 15 baseline plus 30 induced-fault) found that under transient device unresponsiveness, all interrupted trials were reported as client-side failures even though independent read-back confirmed the controller state had been correctly updated, revealing a false-failure risk that could cause naive retry logic to redundantly or unsafely re-send already-successful commands. These simulator-based results establish the correctness and reliability characteristics of the verified deployment logic and motivate, without yet empirically validating, the necessity of the cloud-capable Jalur B path, whose hardware validation remains future work.

Keywords: industrial IoT retrofit, Modbus RTU, bidirectional control verification, fault injection, PID temperature controller

1. Introduction

Many small and medium-scale industrial operations rely on industrial PID temperature controllers, such as the Autonics TN series (TNH/TNL), to run process cycles ranging from injection-molding barrel heating to chemical reactor control, furnace and heat-treatment cycles, and drying or curing ovens. Adding IoT connectivity specifically to PID loop controllers of this kind, rather than to the surrounding line as a whole, has been demonstrated for information-delivery purposes [1], and retrofitting programmable logic and process controllers for Industry 4.0/5.0 scenarios more broadly is an active research thread in its own right [2,3]. Digitally retrofitting already-installed equipment of this kind, rather than replacing it, is itself an established Industry 4.0 strategy, documented both as a general industrial modernization approach [4] and as a systematic digital-retrofitting practice for legacy manufacturing systems [5,6], particularly relevant to resource-constrained small and medium enterprises in developing economies [7]. These controllers are mature, reliable, and already installed; they are not, however, designed for network or cloud connectivity, a characteristic of the product category rather than a defect of any specific installation.

This paper addresses the harder, less-explored half of the retrofit problem: **bidirectional** control, meaning deploying process configuration (PTTN: pattern number, step count, per-step setpoint and duration, end-state behavior) *to* the controller from a web-based SCADA application, rather than merely reading telemetry *from* it. Retrofitting legacy controllers with cloud-based *monitoring* (the read direction) is comparatively well established; writing configuration back to the controller reliably, and verifying that the write actually took effect, is structurally harder and is this paper's focus. Two architecturally distinct paths exist in the system under study:

© 2026 The Authors. Published by Fakultas Teknik, Universitas Wisnuwardhana Malang
This is an open access article under the CC BY NC SA license. (https://creativecommons.org/licenses/by-nc-sa/4.0/)

- **Direct-serial ("Jalur A")**: the Laravel-based SCADA backend writes directly to the controller over RS-485 via a USB-serial adapter, using a Python Modbus bridge process. This path is structurally tied to the physical host running the backend: it cannot reach a controller unless the serial adapter is wired into that same machine.
- **ESP32-mediated ("Jalur B")**: the same backend publishes configuration over MQTT to a cloud broker (already deployed at a public IP address in production), which an ESP32 device, physically co-located with the controller, subscribes to and would, if capable, write onward to the controller over its own local RS-485 connection.

Only the direct-serial path is empirically evaluated with simulator-based accuracy in this paper. The ESP32-mediated write path has been designed and implemented, including firmware functions mirroring the direct-serial register sequence exactly, and compile-verified against the target toolchain, but has not yet been validated against physical hardware; we report its design and implementation status but explicitly do not claim empirical performance results for it (see Section 3.3).

1.1 Contributions

1. A register-level, write-then-read-back verification protocol for bidirectional process-pattern deployment to a legacy Modbus RTU controller, with an explicit accuracy metric (**Pattern Deployment Accuracy**) rather than an assumption that "command sent" implies "command applied."
2. A controlled fault-injection methodology and an empirical reliability characterization revealing a **false-failure** condition, in which the deployment client reports failure while the controller state is in fact correctly updated, with direct implications for retry-safety in industrial retrofit software.
3. A documented architectural constraint, proven from the Modbus bridge implementation itself rather than assumed, that motivates why cloud-capable bidirectional control requires an edge-resident write capability (Jalur B) rather than extending the direct-serial path.

1.2 Related Work

Recent work commonly places a field-side gateway between installed equipment and upper IIoT services. Such gateways translate among Modbus, MQTT, and OPC UA, but their design objectives differ across platforms [8–14]. Boareto et al. [8] demonstrate an IIoT integration case for legacy field devices. Ai et al. [9] separate gateway workloads into isolated instances to support heterogeneous protocols. The implementations in [10,11] emphasize access or acquisition timing, whereas [12] studies two-way Modbus TCP/RTU conversion through a queuing model. Performance-oriented examinations of Modbus links and machine-software bridges are reported in [13,14]. None of these studies measures whether a complete configuration written to a controller is later observed unchanged through an independent register read. Hercog et al. [15] review ESP32 design patterns, and Ungurean [16] presents an STM32 gateway for data-centric IIoT middleware. The present ESP32-S3 node combines production telemetry safeguards, including RTC support, SD buffering with SHA-256 checking, dual-core task separation, and OTA updates, with the proposed verified configuration workflow.

Security research on legacy Modbus and SCADA installations addresses a neighboring, but different, issue. Message-authentication mechanisms [17,18], device-preserving SCADA defenses [19], broader PLC-security analyses [20], and proxy-mediated protection [21] seek to prevent or detect hostile interference. This study instead examines benign communication disruption. In both cases, software must not treat an unverified write response as conclusive evidence of the controller state.

Connectivity can also be added above the fieldbus layer. The approaches in [22,23] respectively expose legacy controllers through an OPC UA agent and connect legacy assets to an Asset Administration Shell. They share the replacement-avoidance objective, but neither uses the controller-side Modbus write/readback workflow evaluated here.

To our knowledge, no prior work combines (a) retrofit of a legacy, non-IoT-native Modbus temperature controller and (b) a verified write-then-read-back bidirectional control protocol with an explicit deployment-accuracy metric and empirical reliability characterization in one evaluated system. This combination, rather than any one element alone, defines the novelty position of this paper. The claim is based on a targeted literature search, not a systematic review; its scope is clarified in Section 3.3.

2. Research Method

2.1 System Architecture and Verified Deployment Protocol

The Autonics TN-series (TNH/TNL) controller exposes its process state and configuration over Modbus RTU (RS-485). Table 1 documents the register ranges actually used by this system, extracted directly from the production codebase (TnRegisterMap.php) rather than assumed from the datasheet alone.

Table 1. TNH register map used by this system.

Register class	Function code	Address range (offset)	Content
Coil	FC01 read / FC05 write	0–2	RUN/STOP, Auto-Tune execute, Reset Alarm
Holding: operation	FC03 read / FC06 write	0–7	Run/stop mirror, auto-tune, auto/manual mode, manual MV, set value (SV), operation mode, 2-DOF mode
Holding: pattern config	FC03 read / FC16 write	200–208 (9 registers)	time_unit, start_condition, wait_width, wait_time, pattern_number, repetitions, end_state, pid_group, step_quantity
Holding: pattern steps	FC03 read / FC16 write	209–248 (40 registers)	20 steps of [target_sv, duration]
Input: monitoring	FC04 read only	1000–1023	Present value (PV), setpoint mirror, status flags, active pattern/step, elapsed/remaining time

A deployed pattern (PTTN) is, physically, a piecewise-constant setpoint trajectory: each step holds a target setpoint (TS) for a target duration (TM) before advancing according to its end-action, and a step with TM = 0 acts as an instantaneous transition rather than a held plateau. Figure 1 shows the setpoint profile of a 4-step recipe (SYN-TEST-004-4STEP: TS0 = 90 °C/TM0 = 0 s, TS1 = 90 °C/TM1 = 300 s, TS2 = 128 °C/TM2 = 0 s, TS3 = 128 °C/TM3 = 180 s) that was deployed and independently read back through the same write-then-read-back protocol as Experiment A, confirming Pattern Deployment Accuracy of 17/17 = 100% for this specific pattern; the figure is plotted from the verified register values, not merely the input recipe, grounding the register-level description above in the physical process it represents. These setpoints were chosen only to exercise the protocol across two distinct plateaus within the realistic industrial process-temperature range used elsewhere in this study (Section 2.2); they are not drawn from, and are not intended to evoke, any single named industrial process. Steps 1 and 2 share the same target setpoint (90 °C), as do Steps 3 and 4 (128 °C), so the deployed profile is physically a two-stage come-up (ambient to 90 °C, then 90 °C to 128 °C) with a hold at each stage, rather than the overshoot-then-drop shape an earlier version of this pattern used. Because the TNH pattern registers store only target setpoint and hold duration per step, not transition rate, the ramp segments connecting the plateaus in Figure 1 are not register data; they are drawn schematically using an assumed illustrative come-up/cool-down rate (3 °C/min from an approximate ambient starting temperature of 38 to 40 °C) purely so the profile's shape is physically plausible, and this rate is explicitly not a measurement.

We emphasize that the two zero-duration steps in this pattern (TM0 = TM2 = 0) are a **protocol-testing artifact**, chosen specifically to exercise a register-level transition across a step boundary, and are not a validated process hold time. In an actual production deployment, every step's hold duration (TM) must instead be derived from a process- and product-specific validation study appropriate to the application before being implemented, consistent with standard process-authority practice for the industry in question; the pattern used here demonstrates deployment-protocol correctness only, not a validated production recipe. Operating a deployed process above its intended reference temperature risks exceeding the validated safe range for that recipe rather than improving the outcome the process is designed for.

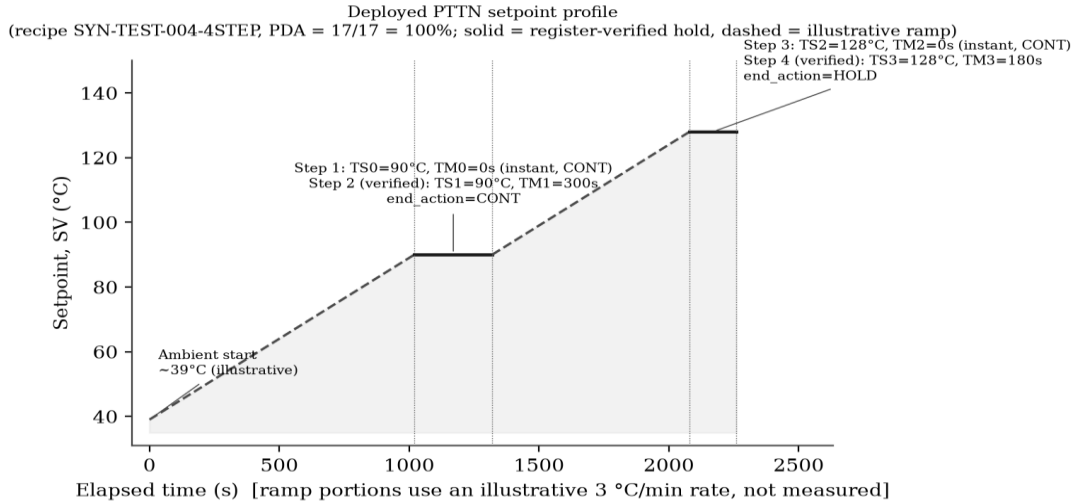

Figure 1. Temperature-vs-time setpoint profile for a deployed four-step process pattern.

Solid segments are the register-verified holds (Steps 2 and 4); dashed segments are illustrative ramps (not register data) shown only so the transitions read as a continuous physical trajectory rather than instantaneous jumps. Steps 1 and 3 (TM = 0) are protocol-testing transition points, not validated process holds.

Two Control Paths.

Direct-serial (Jalur A). The SCADA backend (Laravel) writes to the controller via a Python Modbus bridge process (modbus_bridge.py, using pymodbus) over a USB-RS485 adapter. Pattern deployment proceeds as: (1) write pattern_number as a single register (FC06); (2) write the 9-register pattern-config block (FC16); (3) write the 40-register step block (FC16); (4) independently read back both blocks (FC03) and compare against the values that were sent. This path is fully implemented and, critically, is architecturally *host-local*: the Modbus client library used has no TCP transport, so it can only reach a controller physically wired to the same machine the backend process runs on. Because the production MQTT broker for this system is confirmed to be reachable at a public IP address (that is, the backend is not physically co-located with every factory), this path cannot serve as the control mechanism for a genuinely cloud-hosted, multi-site deployment: a structural limitation, not an implementation gap.

ESP32-mediated (Jalur B). The backend instead publishes pattern configuration over MQTT to the same public-broker infrastructure already used for telemetry. MQTT broker architectures of this kind are widely reported for industrial IoT telemetry and control at varying scales of throughput and reliability requirement, including EMQX-based deployments [24], alternative industrial pub/sub middleware such as Zenoh [25], architectural enhancements for high-throughput peer-to-peer MQTT messaging [26], and MQTT-based real-time anomaly-detection architectures [27]. An ESP32-S3 “IoT Logger” device, physically co-located with the controller and already proven in production for telemetry through its RTC, SD+SHA-256 buffering, dual-core FreeRTOS task separation, and OTA update capability, subscribes and, in the design evaluated here, would write the same register sequence onward to the controller over its own local RS-485 connection, then read back to verify, mirroring the direct-serial protocol exactly. Firmware implementing this write-and-verify sequence (mbWriteSingleRegister, mbWriteMultipleRegisters, and a coordinating mbDeployPatternToTNH function) has been written and compile-verified against the target ESP32-S3 toolchain; **it has not been deployed to or tested against physical hardware**, and no empirical results for this path are reported in this paper (see Section 3.3).

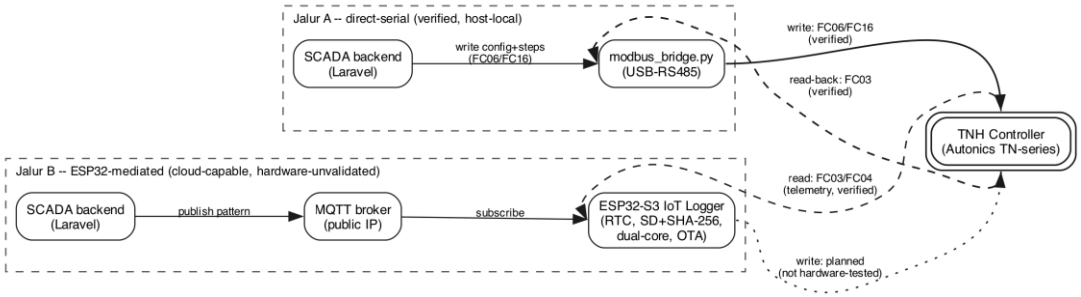

Figure 2. Summarizes both paths together, including the verification status of each hop

Both control paths to the TNH controller. Solid and dashed edges are verified in this study or in production; the dotted edge (ESP32 to TNH write) is implemented and compile-verified but not hardware-tested, and carries no empirical claim in this paper.

2.2 Experiment A: Pattern Deployment Accuracy

Because physical access to a TNH controller was not available during this study, Experiments A and B (Sections 2.2 and 2.3) were conducted against a purpose-built Modbus RTU slave simulator (tnh_simulator.py, using pymodbus) that implements the exact register map in Table 1, connected to the production Laravel codebase via a virtual serial port pair (socat) in place of a physical USB-RS485 cable. This substitution is explicit throughout: it validates the *software-side mechanism* (the correctness of the write sequence, the read-back comparison logic, and the client’s behavior under communication faults) but does not, and cannot, validate the physical RS-485 electrical/timing behavior of a real controller. All results in Section 3 are labeled synthetic accordingly.

100 synthetic recipes were generated with controlled variation: step count cycling through {1, 5, 10, 20} (25 trials per bucket), target setpoint drawn from a realistic industrial process-temperature range (60 to 135 °C), duration, pattern end-state, and start-condition varied per trial. Each recipe was deployed via the direct-serial write sequence and immediately read back. For each trial we recorded, per field, whether the read-back value matched the value sent (accounting for the existing implementation’s known integer-scaling convention, for

example a setpoint below 200 is stored times 10), and the wall-clock latency of the write and read-back phases separately.

Pattern Deployment Accuracy is defined as follows. Let $\boldsymbol{x}_i$ be the expected register vector for trial i,

$$\boldsymbol{x}_i = [x_{i1}, x_{i2}, \dots, x_{iM_i}]$$

and let $\hat{\boldsymbol{x}}_i$ be the independently read-back vector,

$$\hat{\boldsymbol{x}}_i = [\hat{x}_{i1}, \hat{x}_{i2}, \dots, \hat{x}_{iM_i}]$$

where M_i is the number of verifiable fields in trial i. Field-level agreement is encoded by the match indicator:

$$m_{ij} = \begin{cases} 1, & \hat{x}_{ij} = x_{ij} \\ 0, & \hat{x}_{ij} \neq x_{ij} \end{cases}$$

Pattern Deployment Accuracy for trial i is then, using the match indicator from Eq. 3,

$$\mathrm{PDA}_i = \frac{\sum_{j=1}^{M_i} m_{ij}}{M_i} \times 100\%$$

For a set of N deployment trials, mean PDA and its sample standard deviation are:

$$\overline{\mathrm{PDA}} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{PDA}_i$$

$$s_{\mathrm{PDA}} = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (\mathrm{PDA}_i - \overline{\mathrm{PDA}})^2}$$

Deployment latency was measured as the elapsed wall-clock time from the beginning of the first write call to the completion of the final verification read:

$$T_i = t_{i,\mathrm{end}} - t_{i,\mathrm{start}} = T_{i,\mathrm{write}} + T_{i,\mathrm{readback}}$$

Percentile latency is reported as:

$$T_p = Q_p(\{T_1, T_2, \dots, T_N\})$$

The protocol therefore treats confirmation as an observation of device state, not as an implication of transmission. This is analogous to telemetry frameworks that obtain evidence beyond a sensor's own status report [28]. Safe update commitment at the edge-cloud boundary [29] and end-to-edge actuation architectures [30] address the broader separation between issuing a command and establishing its effect. Pattern Deployment Accuracy (Eq. 4) operationalizes that distinction for individual controller registers.

2.3 Experiment B: Fault Injection Reliability

The configuration and step blocks are written in separate FC16 operations. Because the protocol supplies no transaction spanning both operations, a disturbance between them can leave mixed configuration state, a risk considered in [12,13]. Recovery behavior also matters for operational technology networks [31]. Our procedure applies a deliberate, bounded interruption to one simulator transaction and observes the resulting client report and register state. It follows the experimental rationale of resilience and chaos studies of digital twins [32], critical infrastructure [33], and production environments [34], but remains limited to a single device transaction. We used 15 trials in each condition: baseline; a 1-s simulator pause immediately before the step-block write (SIGSTOP); and the same pause lasting 5 s. The simulator was resumed with SIGCONT, and a separate readback established the final register state after every trial.

Let C_i denote the client-reported outcome of trial i, where $C_i = 1$ means the client reports success and $C_i = 0$ means the client reports failure. Let D_i denote the independently verified device state, where $D_i = 1$ means all target registers match the intended pattern and $D_i = 0$ means at least one target register is incorrect. Each trial is classified as:

$$\mathrm{TS}_i = \mathbb{1}(C_i = 1 \wedge D_i = 1)$$
$$\mathrm{TF}_i = \mathbb{1}(C_i = 0 \wedge D_i = 0)$$
$$\mathrm{FF}_i = \mathbb{1}(C_i = 0 \wedge D_i = 1)$$
$$\mathrm{FS}_i = \mathbb{1}(C_i = 1 \wedge D_i = 0)$$

Here, TS denotes a correct success report, TF a correct failure report, FF a failure report despite a correct device state, and FS a success report despite an incorrect device state. The symbol $\mathbb{1}(\cdot)$ is the indicator. The false-failure proportion for a condition is:

$$R_{\mathrm{FF}} = \frac{\sum_{i=1}^{N_c} \mathrm{FF}_i}{N_c} \times 100\%$$

We note explicitly that process-level suspension (SIGSTOP) delays byte processing without discarding the operating system's serial buffer; it therefore models a *device-busy/unresponsive* fault, not a *physical link*

severance in which bytes are genuinely lost. This distinction is material to interpreting the results (Section 3.2) and is not, to our knowledge, addressed in the Modbus-reliability literature we reviewed.

3. Results and Discussion

3.1 Results

Pattern Deployment Accuracy Experiment.

All 100 observations produced complete agreement between the expected and read-back register vectors. Thus, Pattern Deployment Accuracy averaged **100.00%**; its sample SD was **0.00%**, and both the minimum and maximum were **100.00%** (Table 2; Figure 3a)

Table 2. Pattern Deployment Accuracy and latency by step count (N = 25 per bucket).

Step count	Trials	Mean accuracy	Latency p50 (ms)	Latency p90 (ms)
1	25	100.00%	515	538
5	25	100.00%	512	544
10	25	100.00%	510	537
20	25	100.00%	511	549

Overall latency across all 100 trials: **p50 = 511 ms, p90 = 543 ms, p99 = 578 ms, max = 626 ms**. Notably, latency did not scale meaningfully with pattern complexity (a 20-step pattern showed no measurable latency penalty over a 1-step pattern; Figure 3b), indicating that per-transaction overhead (connection setup, Modbus framing, and the subprocess-invocation overhead of the current bridge implementation) dominates over payload size in this range.

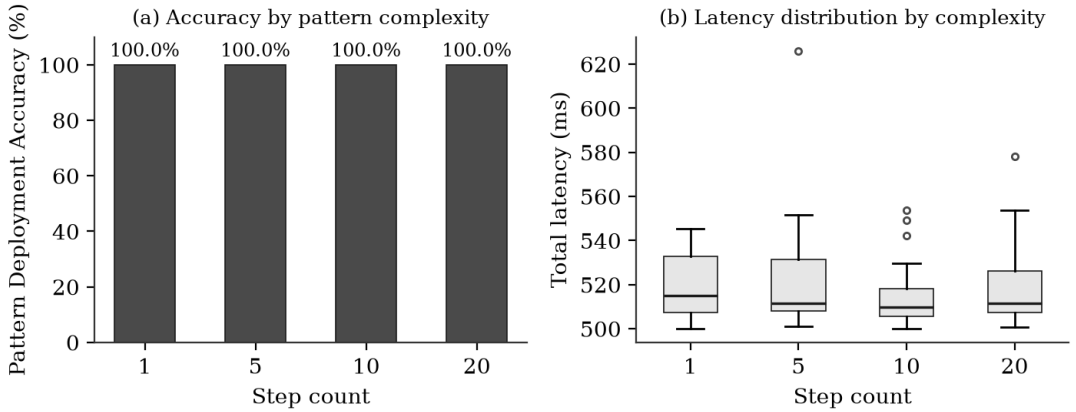

Figure 3. (a) Pattern Deployment Accuracy by step count. (b) Latency distribution by step count, showing no meaningful scaling with pattern complexity.

Fault Injection Reliability Experiment.

The baseline (no-fault) condition confirmed correct experimental setup: **15/15 (100%) true success**. Both fault conditions produced an identical and unexpected result: **15/15 (100%) false failure** in both the short-stall (1 s) and long-stall (5 s) conditions (Table 3; Figure 4). In every interrupted trial, the client reported write failure (a client-side timeout), yet the controller's actual register state was, upon independent read-back after the simulator resumed, correctly updated to the intended values.

Table 3. Fault-injection outcome distribution (N = 15 per condition).

Condition	True success	True failure	False failure	False success
Baseline (no fault)	15 (100%)	0	0	0
Short stall (1 s)	0	0	**15 (100%)**	0
Long stall (5 s)	0	0	**15 (100%)**	0

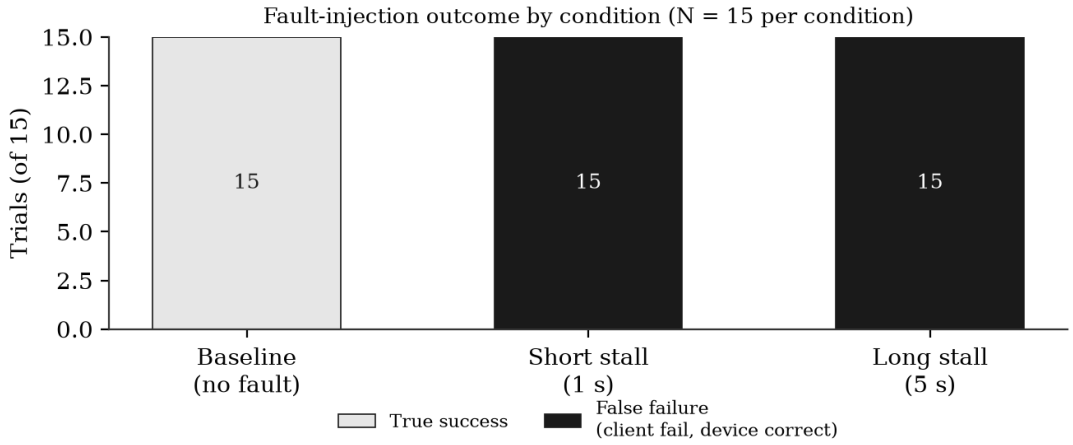

Figure 4. Fault-injection outcome distribution across the three tested conditions.

No *false success* trials were observed in any condition; the more concerning failure mode, in which the client reports success but the device is left incorrect, did not occur under either tested fault type.

3.2 Discussion

The false-failure finding has direct practical implications. A retrofit system that retries a "failed" write without first checking device state risks re-sending an already-successful command, at best redundant, at worst state-corrupting if the retried command differs from the original due to an intervening user action. The Modbus-reliability literature we reviewed emphasizes the risk of *partial* state (a link dropping between two sequential FC16 calls, leaving one write applied and one not) [12,13]; our finding is different in kind: under the specific fault type tested (transient device-unresponsive, not physical byte loss), the write was not partial but *complete and correct*, only reported incorrectly. We consider both risks, partial application from the literature and false-failure reporting from this study, to be real and complementary reliability hazards for this architecture, and recommend that any production retry logic for this system perform an independent read-back check before deciding whether to retry, rather than trusting the write call's own success/failure return value. This recommendation is reinforced, from a different angle, by the Modbus command-integrity literature: Modbus security extensions and non-intrusive legacy-SCADA protection mechanisms similarly treat unauthenticated or unverified command paths as insufficiently trustworthy [17–21].

The measured timings are substantially above the time expected from the Modbus exchange itself. Because this implementation starts a Python process for each operation through Symfony's Process component, process creation is the most plausible source of the observed p50 of 511 ms. The result identifies a local implementation improvement, not an inherent RTU limit. Edge-centered automation keeps time-sensitive work near field devices [35]; measurements of virtual PLC deployments [36] and work on latency-sensitive wireless cloud-fog control [37] support evaluating an edge-resident bridge for this system.

Architecturally, this paper's results argue for, without empirically validating, the necessity of Jalur B. The finding described in Section 2.1, that the direct-serial control path's Modbus client library has no TCP transport and is therefore fundamentally unable to reach a controller from a non-co-located backend, is a structural fact about the existing codebase, verified by source inspection, not a claim requiring the fault-injection or accuracy experiments to support. It establishes *why* a cloud-capable write path is needed; it does not, by itself, establish that the implemented ESP32-based write path (Section 2.1) performs correctly, which remains untested.

3.3 Limitations and Future Work

- ***Jalur* B (ESP32-mediated control) is unvalidated on hardware.** Firmware implementing the write-and-verify sequence has been written and compiles successfully against the target toolchain, mirroring the direct-serial register sequence exactly, but has never been flashed to or executed on physical ESP32 hardware. No performance, accuracy, or reliability claims are made for this path in this paper; validating it, and subsequently conducting a direct head-to-head comparison between the two control paths, is the immediate next step for this research program.
- **All Experiment A and B results are against a software simulator, not a physical TNH controller.** The simulator implements the documented register map faithfully and the fault-injection method operates at the operating-system process level, but neither can substitute for physical RS-485 electrical behavior (line noise, termination, ground loops, genuine byte loss from a severed connection)

that only hardware-in-the-loop testing can characterize. The false-failure finding (Section 3.1) is specific to the *device-unresponsive* fault type tested; we did not test genuine physical-link severance (which would need to sever the serial connection itself, not merely pause the responding process), and do not claim the false-failure rate generalizes to that different fault mechanism.

- **Literature review was targeted, not systematic.** The literature review followed targeted keyword searches rather than a registered or systematic-review protocol. It did not exhaustively screen the major indexing services. Accordingly, the novelty statement is a qualified conclusion from the retrieved studies, not proof that no other relevant work exists.
- **Future work**: (1) hardware validation of Jalur B and the empirical head-to-head comparison between the two control paths; (2) physical-link-severance fault injection to complement the device-unresponsive results reported here; (3) reducing the Jalur A latency overhead identified in Section 3.2; (4) a systematic literature review ahead of any subsequent, more novelty-dependent submission from this research program.

4. Conclusion

We presented a verified, register-level, write-then-read-back protocol for bidirectional process-pattern deployment to a legacy Autonics TNH temperature controller, and evaluated it quantitatively rather than assuming its correctness: 100% deployment accuracy across 100 trials of varying complexity and a previously undocumented false-failure reliability behavior under transient communication faults (100% consistent across 30 interrupted trials). These results, together with a structurally demonstrated (not merely asserted) architectural limitation of the direct-serial control path, motivate and are intended to validate the design of a companion cloud-capable control path, implemented and compile-verified in this work but explicitly reserved as future work pending hardware-in-the-loop validation. We consider the honest scoping of what is and is not empirically established here (synthetic-simulator results clearly labeled as such, an implemented-but-untested control path clearly separated from the paper's empirical claims) to be as much a contribution of this work as the individual numerical results themselves.

Notation

$\boldsymbol{x}_i$	: expected register vector, trial i
$\widehat{\boldsymbol{x}}_i$	: read-back register vector, trial i
M_i	: verifiable fields, trial i
m_{ij}	: field-level match indicator
PDA_i	: Pattern Deployment Accuracy, trial i (%)
N	: number of deployment trials
$\overline{\mathrm{PDA}}$	: mean PDA across N trials (%)
s_{PDA}	: sample SD of PDA (%)
T_i	: total deployment latency, trial i (ms)
$T_{i,\mathrm{write}}$	: write-phase latency, trial i (ms)
$T_{i,\mathrm{readback}}$	: read-back latency, trial i (ms)
T_p	: p-th percentile latency (ms)
C_i	: client-reported outcome, trial i (0/1)
D_i	: verified device state, trial i (0/1)
TS_i	: true-success indicator, trial i
TF_i	: true-failure indicator, trial i
FF_i	: false-failure indicator, trial i
FS_i	: false-success indicator, trial i
R_{FF}	: condition-level false-failure rate (%)
N_c	: number of trials in condition c

References

[1] Y.-C. Tsao, Y.-W. Kuo, M. Shih, J.-W. Chang, J.-S. Wang, and H.-W. Wang, "An Implementation and Development of Information Delivery for PID Controller Based on Internet of Thing," in *2021 IEEE 3rd Eurasia Conference on Biomedical Engineering, Healthcare and Sustainability (ECBIOS)*, 2021, pp. 274–277. doi: 10.1109/ECBIOS51820.2021.9510461.

[2] T. Tran, T. Ruppert, G. Eigner, and J. Abonyi, "Retrofitting-based development of brownfield Industry 4.0 and Industry 5.0 solutions," *IEEE Access*, vol. 10, pp. 64348–64374, 2022. doi: 10.1109/ACCESS.2022.3182491.

[3] B. Rupprecht, E. Trunzer, S. König, and B. Vogel-Heuser, “Concepts for Retrofitting Industrial Programmable Logic Controllers for Industrie 4.0 Scenarios,” in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, 2021, pp. 1034–1041. doi: 10.1109/ICIT46573.2021.9453558.
[4] R. S. Mendonça, R. L. P. Medeiros, L. E. Sales e Silva, R. G. G. Silva, L. G. S. Santos, and V. F. de Lucena Jr., “Enabling Technologies of Industry 4.0 for the Modernization of an Industrial Process,” *Processes*, vol. 13, no. 8, p. 2488, 2025. doi: 10.3390/pr13082488.
[5] A. Alqoud, D. Schaefer, and J. Milisavljevic-Syed, “Industry 4.0: a systematic review of legacy manufacturing system digital retrofitting,” *Manufacturing Review*, vol. 9, p. 32, 2022. doi: 10.1051/mfreview/2022031.
[6] E. V. do Nascimento, V. dos Santos, and R. Aroca, “An applied approach for integrating legacy PLC-based systems into Industry 4.0 environments using low-code platforms,” *The International Journal of Advanced Manufacturing Technology*, pp. 3017–3037, 2026. doi: 10.1007/s00170-026-17921-0.
[7] S. Govardhan, B. E. Narkhede, R. Raut, and S. Ghoshal, “Industry 4.0 adoption barriers in manufacturing supply chain industry: a case study from a developing economy,” *International Journal of Computer Integrated Manufacturing*, pp. 1–19, 2025. doi: 10.1080/0951192X.2025.2544541.
[8] P. Boareto, C. de A. Girardi, F. L. Forin, P. E. R. Moraes, and E. A. P. Santos, “Design and Implementation of Industrial Internet of Things for Legacy Field Devices: A Case Study,” *IEEE Access*, pp. 45882–45895, 2026. doi: 10.1109/ACCESS.2026.3674418.
[9] Y. Ai, Y. Zhu, Y. Jiang, and Y. Deng, “MIGS: A Modular Edge Gateway with Instance-Based Isolation for Heterogeneous Industrial IoT Interoperability,” *Sensors*, vol. 26, no. 1, p. 314, 2026. doi: 10.3390/s26010314.
[10] C. Ventuneac and V. G. Gaitan, “Industrial Internet of Things Gateway with OPC UA Based on Sitara AM335X with ModbusE Acquisition Cycle Performance Analysis,” *Sensors*, vol. 24, no. 7, p. 2072, 2024. doi: 10.3390/s24072072.
[11] V. G. Găitan and I. Zagan, “Experimental Implementation and Performance Evaluation of an IoT Access Gateway for the Modbus Extension,” *Sensors*, vol. 21, no. 1, p. 246, 2021. doi: 10.3390/s21010246.
[12] S. Zhao et al., “Enhancing Bidirectional Modbus TCP to RTU Gateway Performance: A UDP Mechanism and Markov Chain Approach,” *Sensors*, vol. 25, no. 13, p. 3861, 2025. doi: 10.3390/s25133861.
[13] V. Găitan and I. Zagan, “Modbus Protocol Performance Analysis in a Variable Configuration of the Physical Fieldbus Architecture,” *IEEE Access*, vol. 10, pp. 123942–123955, 2022. doi: 10.1109/ACCESS.2022.3224720.
[14] E. Tapia, L. Sastoque Pinilla, U. López-Novoa, I. Bediaga, and L. N. López de Lacalle, “Assessing Industrial Communication Protocols to Bridge the Gap between Machine Tools and Software Monitoring,” *Sensors*, vol. 23, no. 12, p. 5694, 2023. doi: 10.3390/s23125694.
[15] D. Hercog, T. Lerher, M. Truntič, and O. Težak, “Design and Implementation of ESP32-Based IoT Devices,” *Sensors*, vol. 23, no. 15, p. 6739, 2023. doi: 10.3390/s23156739.
[16] I. Ungurean, “Integrating Fieldbus and Data-Centric Middleware: An STM32 Modbus Master Gateway for DDS-Based IIoT Systems,” *Technologies*, vol. 13, no. 11, p. 526, 2025. doi: 10.3390/technologies13110526.
[17] F. Katulić, D. Sumina, S. Groš, and I. Erceg, “Protecting Modbus/TCP-Based Industrial Automation and Control Systems Using Message Authentication Codes,” *IEEE Access*, vol. 11, pp. 47007–47023, 2023. doi: 10.1109/ACCESS.2023.3275443.
[18] T. Martins and S. V. G. Oliveira, “Enhanced Modbus/TCP Security Protocol: Authentication and Authorization Functions Supported,” *Sensors*, vol. 22, no. 20, p. 8024, 2022. doi: 10.3390/s22208024.
[19] A. Chan and J. Zhou, “Non-Intrusive Protection for Legacy SCADA Systems,” *IEEE Communications Magazine*, vol. 61, no. 6, pp. 36–42, 2023. doi: 10.1109/MCOM.003.2200564.
[20] W. Alsabbagh and P. Langendörfer, “Security of Programmable Logic Controllers and Related Systems: Today and Tomorrow,” *IEEE Open Journal of the Industrial Electronics Society*, vol. 4, pp. 659–693, 2023. doi: 10.1109/OJIES.2023.3335976.
[21] F. Trungadi, A. Celesti, A. Galletta, M. Fazio, and F. Longo, “Securing Modbus in legacy industrial control systems: A decentralized approach using proxies, Post-Quantum Cryptography and Self-Sovereign Identity,” *Journal of Information Security and Applications*, p. 104199, 2025. doi: 10.1016/j.jisa.2025.104199.
[22] S.-Y. Lee and M. Sung, “OPC-UA Agent for Legacy Programmable Logic Controllers,” *Applied Sciences*, vol. 12, no. 17, p. 8859, 2022. doi: 10.3390/app12178859.
[23] A. Zielstorff, D. Schöttke, A. Hohenhövel, T. Kämpfe, S. Schäfer, and F. Schnicke, “Overcoming Challenges in Integrating Legacy Devices with Asset Administration Shells: An OPC UA Case Study,” in *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2023, pp. 1–8. doi: 10.1109/ETFA54631.2023.10275536.

[24] M. Kashyap, A. K. Dev, and V. Sharma, “Implementation and analysis of EMQX broker for MQTT protocol in the Internet of Things,” *e-Prime, Advances in Electrical Engineering, Electronics and Energy*, vol. 10, p. 100846, 2024. doi: 10.1016/j.prime.2024.100846.
[25] M. Barón, L. Diez, M. Zverev, J. R. Juárez, and R. Agüero, “On the performance of Zenoh in Industrial IoT Scenarios,” *Ad Hoc Networks*, vol. 170, p. 103784, 2025. doi: 10.1016/j.adhoc.2025.103784.
[26] D. Shvaika, A. Shvaika, and V. Artemchuk, “MQTT Broker Architectural Enhancements for High-Performance P2P Messaging: TBMQ Scalability and Reliability in Distributed IoT Systems,” *IoT*, vol. 6, no. 3, p. 34, 2025. doi: 10.3390/iot6030034.
[27] K. I. Ko and M. H. Lee, “MQTT-Based Architecture for Real-Time Data Collection and Anomaly Detection in Smart Livestock Housing,” *Sensors*, vol. 25, no. 23, p. 7186, 2025. doi: 10.3390/s25237186.
[28] T. Gupta, S. Handa, and A. Nambi, “Verified Telemetry: A General, Easy to Use, Scalable and Robust Fault Detection SDK for IoT Sensors,” in *Proc. 8th ACM/IEEE Conference on Internet of Things Design and Implementation (IoTDI)*, 2023, pp. 381–395. doi: 10.1145/3576842.3582386.
[29] Y. Lin and Y. Lin, “A Bounded-Update Commitment Protocol (BUCP) for Safe Edge-Cloud Adaptive Control of Networked Industrial Robots,” *IEEE Access*, pp. 68485–68503, 2026. doi: 10.1109/ACCESS.2026.3690201.
[30] Y. Ma et al., “Smart Actuation for End-Edge Industrial Control Systems,” *IEEE Transactions on Automation Science and Engineering*, vol. 21, pp. 269–283, 2024. doi: 10.1109/TASE.2022.3216217.
[31] M. Boeding, J. Valente, J. C. Lopez, and J. Lopez, “Exponential Backoff and Its Security Implications for Safety-Critical OT Protocols over TCP/IP Networks,” *Future Internet*, vol. 17, no. 7, p. 286, 2025. doi: 10.3390/fi17070286.
[32] M. Fogli, C. Giannelli, F. Poltronieri, C. Stefanelli, and M. Tortonesi, “Chaos Engineering for Resilience Assessment of Digital Twins,” *IEEE Transactions on Industrial Informatics*, vol. 20, no. 1, pp. 1134–1143, 2024. doi: 10.1109/TII.2023.3264101.
[33] P. Dedousis, G. Stergiopoulos, G. Arampatzis, and D. Gritzalis, “Enhancing Operational Resilience of Critical Infrastructure Processes Through Chaos Engineering,” *IEEE Access*, vol. 11, pp. 106172–106189, 2023. doi: 10.1109/ACCESS.2023.3316028.
[34] A. M. A. Doan, D. R. Sellitto, M. Oppitz, and M. Weigold, “Requirement analysis and approach for Chaos Engineering in industrial production,” *CIRP Journal of Manufacturing Science and Technology*, 2026. doi: 10.1016/j.cirpj.2026.02.013.
[35] J. Jin, K. Yu, J. Kua, N. Zhang, Z. Pang, and Q.-L. Han, “Cloud-Fog Automation: Vision, Enabling Technologies, and Future Research Directions,” *IEEE Transactions on Industrial Informatics*, vol. 20, no. 2, pp. 1039–1054, 2024. doi: 10.1109/TII.2023.3272696.
[36] M. Gaffurini, P. Bellagente, A. Depari, A. Flammini, E. Sisinni, and P. Ferrari, “Virtual PLC in Industrial Edge Platform: Performance Evaluation of Supervision and Control Communication,” *IEEE Transactions on Instrumentation and Measurement*, vol. 73, pp. 1–10, 2024. doi: 10.1109/TIM.2024.3370746.
[37] H. Lyu, Z. Pang, A. Bengtsson, S. Nilsson, A. J. Isaksson, and G. Yang, “Latency-Aware Control for Wireless Cloud-Fog Automation: Framework and Case Study,” *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 5400–5410, 2025. doi: 10.1109/TASE.2024.3420770.