electronics

MDPI

Article

Unified Blockchain Provenance and Bayesian Reputation for Additive Manufacturing Supply Chains †

Raddad Almaayn[1,*], Ashok Karukutla [1], Henok Berhanu Tsegaye [1,*], Petro Mushidi Tshakwanda [1], Harsh Kumar [2], Michael Devetsikiotis [1]

[1] Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131, USA; akarukutla@unm.edu (A.K.); pmushidi@unm.edu (P.M.T.); mdevets@unm.edu (M.D.)
[2] R.B. Annis School of Engineering, University of Indianapolis, Indianapolis, IN 46227, USA; kumarh@uindy.edu
* Correspondence: raddadalmaayn@unm.edu (R.A.); htsegaye@unm.edu (H.B.T.)
† This article is a substantially extended version of our conference papers presented at the 2026 IEEE 23rd Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 9–12 January 2026, and at the 2026 IEEE 16th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 5–7 January 2026. The provenance-to-reputation bridge and its on-chain mapping policy, reputation-gated authorization, the four-organization physical testbed, the calibrated fault-injection validation, and the comparison of the three transactional arrangements (Sections 3.6, 4.4, 5.5, 5.7 and 5.8) are new to this work.

Abstract

In additive manufacturing (AM) supply chains, systems that record provenance and score reputation as separate transactions let lifecycle records and assessments diverge. We present a Hyperledger Fabric chaincode that commits a provenance event, the Bayesian reputation update it implies and their link in one read–write set. An on-chain policy admits ratings only from counterparty organizations; reputation gates authorize future activities but never block recording outcomes. Under client crashes in the measured partial-write window, the unified design produced no divergence in 3900 trials (2900 with version 1.0, 1000 with version 1.1), while a client-orchestrated two-chaincode deployment diverged in all 1000 trials of two control batches. A same-channel InvokeChaincode arrangement also produced none in 1000, at 4.5 ms more endorsement and 11.7% lower throughput at 100 transactions in flight. On four desktops, open-loop load saturates near 12 tx/s, and a 12-h soak at 6 tx/s committed 98.7% of submissions with no partial record found. Ledger experiments cover gate refusal, failed outcomes, recovery and disputes; with client retry, 540 contended ratings each committed once. A simulation quantifies gate errors and shows that evaluation-time decay reacts slowly when active actors' performance drops. The scores record submitted outcomes; they are not validated measures of performance.

Keywords: additive manufacturing; blockchain; Hyperledger Fabric; provenance tracking; Bayesian reputation; supply chain trust; Chaincode-as-a-Service; smart manufacturing; distributed manufacturing

Check for updates

Academic Editor: Nai-Wei Lo

Received: 1 September 2026
Revised: 20 October 2026
Accepted: 28 October 2026
Published: 2 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Additive manufacturing (AM) has become an integral part of Industry 4.0 production [1]. Unlike traditional subtractive manufacturing, AM supply chains are structurally decentralized: a single safety-critical part may involve a powder supplier certifying raw material in one jurisdiction, a print service bureau operating an industrial machine in another, an independent laboratory performing non-destructive testing, a logistics firm

handling temperature-controlled transportation, and a regulator issuing final airworthiness or implant certification. Each organizational hand-off introduces an opportunity for data loss, retroactive manipulation, or ambiguity about accountability when a defect surfaces later in service.

Distributed AM is also part of the infrastructure of smart cities and smart manufacturing ecosystems. Networked printer fleets embedded in urban production networks allow safety-critical parts to be produced at the point of use, so a part's lifecycle record is assembled from several organizations; blockchain has been surveyed as a basis for such product lifecycle management in Industry 4.0 [2]. A taxonomy of blockchain-based AM business models names the lack of trust among participants as a primary barrier to their adoption [1]. A city-scale production network in which hospitals and transit operators source parts from nearby bureaus therefore requires trust infrastructure that no single organization controls.

Quality-management standards in regulated industries require identification and traceability of parts (AS9100 [3] in aerospace, ISO 13485 [4] in medical devices), and the digital design files on which AM depends carry cybersecurity risks that motivate its integration with blockchain [5].

The industry response has largely been to deploy single-organization traceability systems that provide no assurance across organizational boundaries.

Two complementary capabilities are required at the supply chain level. Provenance tracking provides an immutable, tamper-evident record of each submitted lifecycle event so that any stakeholder can reconstruct a part's recorded history and verify that no committed record was later altered. Reputation scoring summarizes the recorded outcomes of each actor's work into quantitative indicators that can inform partner selection and condition the authorization of critical lifecycle steps.

Blockchain technology provides a natural substrate for both capabilities [6]. Hyperledger Fabric, a permissioned blockchain framework, is particularly well-suited for industrial supply chains. It provides authenticated identities through X.509 Certificate Authorities (CAs), configurable endorsement policies requiring agreement from a specified subset of organizations, and deterministic smart-contract execution via the execute-order-validate architecture.

1.1. The Integration Gap

Prior work has addressed provenance and reputation independently. Blockchain-based provenance frameworks, for AM [7] and for electronics supply chains [8], provide immutable traceability but treat all authenticated participants as equally trustworthy. Decentralized reputation models [9] quantify reliability but operate as standalone systems that require humans to submit ratings after each transaction, are not grounded in on-chain physical-world evidence, and lack integration with manufacturing-specific lifecycle records.

The consequence of this separation is a consistency gap: when a lifecycle event and the rating it implies are validated as separate transactions, the provenance record and the reputation state can diverge, so the score a buyer inspects need not reflect the events the ledger records. A related but distinct problem is completeness: an actor that reports selectively, submitting favorable outcomes and omitting adverse ones, corrupts both records alike. Atomic co-commitment closes the first gap. It does not by itself close the second, which depends on who is required to submit outcomes and on what external expectation reveals an omission (Section 3.2).

1.2. Contributions

This paper addresses the consistency gap as follows.

1. We co-deploy three Fabric smart contracts, `ProvenanceContract`, `ReputationContract` and `IntegrationContract`, in a single Go chaincode, so that a provenance event, its reputation update and their link entry commit in one read–write set. Atomicity is Fabric's; the contribution is the design that applies it to provenance and reputation.
2. The provenance layer anchors compact structured metadata for each AM lifecycle event, binding it via a Secure Hash Algorithm 256 (SHA-256) digest to an off-chain artifact store, so that the artifact itself never enters the ledger, at approximately 450 B per provenance-event value and approximately 1.17 kB of serialized world-state value footprint per bridge invocation.
3. The reputation layer implements the Beta reputation model with its two-sided update, $O(1)$ per rating over a runtime-configurable set of dimensions, 95% Bayesian credible intervals computed identically on every endorser, evaluation-time exponential decay, a stake requirement on every rating and a dispute-and-slash mechanism.
4. We define a provenance-to-reputation mapping policy and enforce it on chain: actor labels are bound to the organization that registered them and to a role, and each rating-producing event may be submitted only by a counterparty organization that has declared a designated role. Reputation gates authorize future activities and never apply to the recording of outcomes.
5. We document an end-to-end Chaincode-as-a-Service deployment on Hyperledger Fabric and benchmark the system on a single-host, two-organization deployment and a distributed, four-organization physical local area network (LAN).
6. We validate the atomicity claim with a fault model calibrated to measured transaction phase boundaries, against a real client-orchestrated two-chaincode deployment and against the same-channel `InvokeChaincode` arrangement built from the same source, and we measure the cost of each atomic arrangement.
7. We evaluate the reputation mechanism itself: gate acceptance and refusal on the ledger, a lifecycle scenario with failed outcomes, refusal, recovery and disputes, identity and policy attacks, a bursty workload on one actor with client retry, and a simulation of accuracy, gate errors, decay sensitivity and manipulation.

The underlying claim is a consistency invariant: when a provenance event and the reputation update it implies are produced within one chaincode read–write set, Fabric's execute-order-validate semantics commit both or neither, without application-level coordination. Client-orchestrated invocation of two separate chaincodes produces two independently validated read–write sets and leaves a window in which lifecycle evidence and the responsible actor's score can diverge; Section 5.7 places client crashes in that window. A same-channel `InvokeChaincode` call also closes the window, at some cost in endorsement time and throughput, while allowing the contracts to be upgraded separately (Section 5.8); the choice between the two atomic arrangements is a trade-off, not a matter of correctness. Atomicity itself is Fabric's guarantee. This paper contributes the design that uses it for provenance and reputation, the policy that governs who may submit a rating-producing event, and measurements of what each arrangement protects and costs.

1.3. Paper Organization

Section 2 surveys related work and the research gap, Section 3 presents the architecture, Section 4 discusses the implementation and deployment, Section 5 presents the evaluation results, Section 6 presents the interpretation and the limitations, and Section 7 discusses the conclusions and future directions.

2. Related Work

2.1. Blockchain Provenance in Additive Manufacturing

Alkhader et al. [7] developed a blockchain traceability solution for AM in which Ethereum smart contracts govern and trace the participants' transactions and the InterPlanetary File System stores design files, Internet of Things (IoT) device records and product specifications. The system records what happened and who performed each step, but provides no assessment of whether actors met quality standards and no mechanism to restrict future participation by poor-performing actors. Liu et al. [10] design a blockchain-based supply chain management framework on Hyperledger Fabric to counter information asymmetry and the lack of supervision among parties, and evaluate its prototype with Hyperledger Caliper; the framework is generic to supply chains and models neither manufacturing lifecycle events nor actor-level trust accumulation. Grünewald et al. [1] present a taxonomy of blockchain-based business models in AM and name the lack of trust among participants as a primary barrier to them. Leng et al. [2] survey blockchain-empowered product lifecycle management in Industry 4.0, but do not integrate reputation scoring with lifecycle provenance. Priyadarshini and Singh [5] identify the challenges of integrating blockchain with AM, motivated by the cybersecurity risks of digital design files, and rank their causal relationships with fuzzy DEMATEL. Haw et al. [11] review blockchain along the digitized AM process chain, where data security and logistics records are the main motivations. Ferreira et al. [12] deploy Hyperledger Fabric with attribute-based access control in an additive symbiotic network, in which by-products of other industries become AM feedstock; in that case study, the shared ledger removes intermediaries and keeps exchange records available to stakeholders who had no prior trust relationship, but participants are not scored.

The common limitation of all reviewed AM provenance systems is that they record events without inferring trust from them, and they do not use accumulated trust to condition future access. The provenance record is passive: it describes the past but does not protect the future.

2.2. Decentralized Reputation and Trust Modeling

The Beta reputation system [9] models trust as a Beta distribution parameterized by accumulated evidence from positive and negative outcomes, enabling closed-form Bayesian updates and principled uncertainty quantification. Jøsang et al. [13] survey trust and reputation systems for online service provision, in which parties rate each other and aggregated ratings yield trust scores. Rathee et al. [14] address blockchain-based trust for industrial IoT networks but target device-level authentication, not actor-level reputation tied to process outcomes. Wu and Zhang [15] propose an integrated framework for blockchain-enabled supply-chain trust management directed at smart manufacturing. EigenTrust [16] propagates local trust evaluations globally through a graph, but requires $O(|V|^2)$ computation per update in the worst case, making it unsuitable for on-chain execution.

Cheng and Chow [17] propose an augmented Bayesian reputation metric that evaluates the trustworthiness of the agents of a consensus-based distributed microgrid energy management system and detects attacks on it, which makes it the closest predecessor to the reputation layer in this work. It scores agents of a distributed control system; there is no rating of recorded process outcomes, no provenance trail, and no integration with physical process records.

Existing blockchain reputation systems collect ratings as transactions separate from any record of the work being rated. No reviewed system commits a reputation update in the same transaction as the on-chain event that records the outcome it is based on, which is the defining property of the provenance-to-reputation bridge in this work.

2.3. Research Gap and Positioning

The central gap is the absence of *causal integration*: in no reviewed system does a lifecycle event commit in the same transaction as the reputation consequence attributed to it, so the two are validated independently and can diverge. The bridge presented here co-commits a provenance event, the reputation update submitted with it, and their link entry in one Fabric transaction. Yavaprabhas et al. [18] consolidate 192 studies of blockchain and trust in supply chains and use trust-transfer theory to explain how trust in one target, such as the ledger, transfers to another, such as a trading partner. That framing marks the limit of what a ledger design can show: it can make recorded outcomes and their reputation consequences consistent and attributable to the organization that submitted them, but whether those outcomes deserve trust depends on who supplies them, which is why the mapping policy of Section 3.6 assigns every rating-producing event to a counterparty. Architectures that pair a ledger with other data systems, such as IoT and building information models in offsite manufacturing [19], improve traceability and storage without assessing the participants. A centralized append-only log would provide tamper evidence within a single administrative domain at lower cost. The difficulty is that an AM consortium has no organization that all parties are willing to have operate that log, and a supplier's trust score is the record it has the strongest incentive to influence. A permissioned ledger removes the operator from that role: no single organization can commit a record, and every participant independently validates the same state. Fabric performance studies identify configuration bottlenecks [20], survey latency models [21] and reorder transactions that involve the same key [22]; they do not consider operations that span several contracts in one transaction. Token design work separates wealth from reputation with a two-token scheme [23], but its reputation reflects users' contributions and votes, not verifiable process signals, and multi-agent security architectures for smart manufacturing [24] harden the learning layer without binding agent trust to lifecycle evidence.

Table 1 compares reviewed systems across six functional dimensions derived from the requirements analysis in Section 3.1: provenance tracking specific to AM workflows, reputation scoring, multi-dimensional scoring to capture distinct performance aspects, atomic bridging to prevent divergence, objective evidence grounding to eliminate subjective rating dependence, and trust-conditioned access control. These dimensions reflect the minimum functional surface required to close the consistency gap identified in Section 1. Our prior work [25] demonstrated lightweight provenance anchoring for AM supply chains, and [26] introduced multi-dimensional Bayesian reputation scoring with adversarial robustness evaluation; each system ran on its own, with no connection between a recorded event and a rating. The following are new here. (i) The bridge that records a provenance event, its reputation update and their link in one transaction, governed by an on-chain mapping policy that fixes which organization may submit which rating-producing event (Section 3.6). (ii) Reputation-gated authorization of future activities, separated from the recording of outcomes (Section 3.6.3). (iii) A fault-injection method calibrated to measured transaction phase boundaries, which verifies that kills land inside the interval where a partial write could exist (Section 5.7.1). (iv) A measured comparison of the three ways Fabric can arrange a provenance-plus-reputation operation (Section 5.8). (v) Evaluation on a four-organization physical testbed (Section 5.5).

Table 1. Comparison of provenance and reputation approaches. Prov. = additive manufacturing (AM) provenance tracking; Rep. = reputation scoring; M-D = multi-dimensional scoring; A-B = atomic provenance-to-reputation bridge; O-E = objective on-chain evidence; T-G = trust-conditioned gate. ✓ = full, ◦ = partial, – = none.

Work	Prov.	Rep.	M-D	A-B	O-E	T-G
Alkhader [7]	✓	–	–	–	–	–
Brandín [19]	◦	–	–	–	–	–
Ferreira [12]	◦	–	–	–	–	–
Beta Rep. [9]	–	✓	–	–	–	–
EigenTrust [16]	–	✓	–	–	–	–
Cheng & Chow [17]	–	✓	–	–	–	–
Almaayn et al. [25]	✓	–	–	–	–	–
Almaayn et al. [26]	–	✓	✓	–	–	–
Proposed Approach	✓	✓	✓	✓	◦	✓

The partial mark for objective evidence reflects that a bridge rating co-commits with a lifecycle event whose outcome value is supplied by a counterparty organization under the enforced mapping policy (Section 3.6) and endorsed by three of the four organizations. Counterparty submission and endorsement constrain who may record an outcome; neither attests that it is true (Section 6.4).

3. System Architecture

3.1. Design Rationale

Prior systems [7,9] treat provenance and reputation as independent problems, creating three practical consequences. **Passive audit trail**: provenance records every lifecycle event but assigns no weight to the outcome; a supplier with repeated quality assurance (QA) failures appears identical to one with a perfect record. **Ungrounded ratings**: standalone reputation systems collect ratings as separate manually submitted transactions, allowing strategic omission and enabling the trust score a buyer inspects to reflect a different event set than the provenance ledger. **Atomicity requirement**: closing the gap requires that a lifecycle event and the reputation update submitted with it co-commit or neither commits. The bridge removes the divergence between the two, not the need to supply an outcome.

Figure 1 shows the unified architecture that addresses these consequences. The top layer is a single Go chaincode binary deployed via Chaincode-as-a-Service (CCAAS) that contains three co-deployed contracts. The Provenance contract implements the AM lifecycle state machine and anchors ∼450 B of SHA-256-verified metadata per event. The Reputation contract maintains per-actor Beta(α, β) scores with 95% Bayesian credible intervals, evaluation-time decay, a stake requirement on every rating submission and dispute-driven slashing. The Bridge contract connects the two: it atomically co-commits a provenance event with the reputation update submitted alongside it (solid arrows, left to right), maintains a bidirectional index linking provenance records to their reputation consequences, and provides a reputation gate (dashed arrow, right to left) that checks an actor's score before authorizing a new activity; recording an outcome is never gated (Section 3.6.3). Because all three contracts share a single Fabric read–write set, the provenance event, its reputation consequence, and the linking index entry either commit together or not at all.

The bottom layer shows the underlying Hyperledger Fabric v3.1.4 infrastructure: four organizations: Manufacturer, Supplier, Logistics, and Regulator. Each operates a dedicated peer and controls its own Fabric Certificate Authority, with a three-node Raft ordering service spanning the Supplier, Logistics and Regulator nodes, and a 3-of-4 threshold endorsement. An off-chain MinIO (S3-compatible) store holds the full artifacts referenced by on-chain SHA-256 digests. Client invocations use the naming convention `ContractName:FunctionName` (e.g., `IntegrationContract:RecordProvenanceWithReputation`).

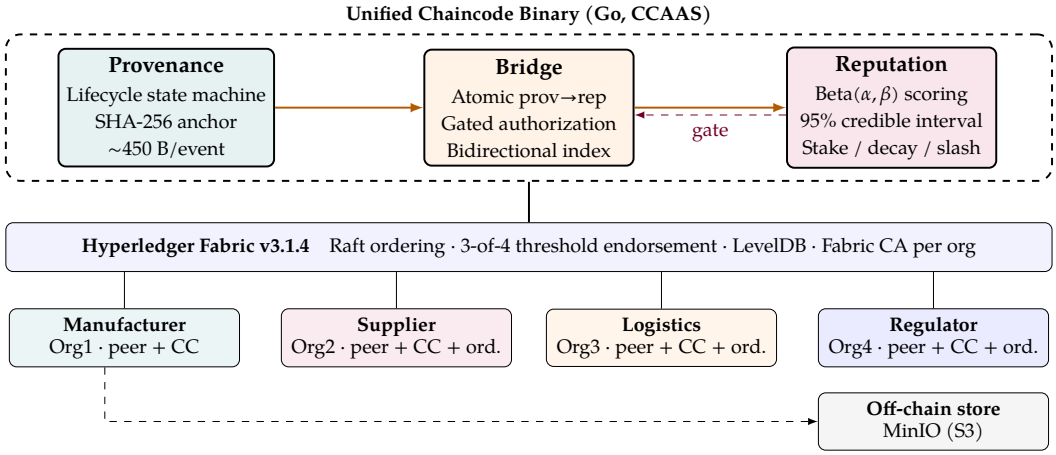

Figure 1. Unified system architecture. Three co-deployed smart contracts share a single Fabric read–write set, enabling the atomic provenance-to-reputation bridge. Solid orange arrows: the atomic provenance-to-reputation (prov→rep) path through the bridge; dashed arrow: the reputation gate, checked before a new activity is authorized; thin dashed arrow: artifacts held off chain. Each organization operates its own peer and Chaincode-as-a-Service (CCAAS) chaincode container (CC) on dedicated hardware and controls its own Fabric Certificate Authority (CA); the three Raft orderers (ord.) run on the Supplier, Logistics and Regulator nodes. SHA-256 = Secure Hash Algorithm 256; S3 = Simple Storage Service.

3.2. Threat Model

We consider a consortium of self-interested organizations operating in an environment where no single party is fully trusted. The adversarial capabilities we consider fall into four categories. *Strategic omission* is different from the other three. Atomic co-commitment guarantees *consistency*: every recorded outcome carries its reputation consequence, and no consequence exists without its record. It does not guarantee *completeness*: an outcome that no one submits leaves neither a record nor a consequence. The design addresses completeness in two limited ways. First, for the four sequence-constrained event types of Section 3.4.3, every path, including the integrated bridge, rejects an out-of-sequence event, so a part cannot advance past an omitted constrained event: an omitted inspection stalls the part rather than disappearing. The revised chaincode also rejects, on every path, an unconstrained event string equal to a stage name, which in the submitted implementation could move an asset into that stage and bypass the constraint (Section 5.6). Second, the enforced mapping policy assigns each rating-producing event to a counterparty organization, not to the performer (Section 3.6), so suppressing an adverse outcome requires the counterparty's cooperation. Omission of an unconstrained event, or of an outcome that the counterparty also declines to submit, cannot be detected from the ledger alone; detecting it requires an external expectation such as a production schedule, a contractual event list or a timeout on an open part, which this paper does not implement. Because the predecessor assertion precedes the first write, a rejected submission fails at endorsement and never reaches the orderer, so no partial state is left behind. Event types outside the constrained set are recorded without a predecessor check and set the asset stage to the event string, so interleaving an unconstrained event between two constrained ones blocks the later constrained event; the sequencing guarantee covers the constrained event types, not the lifecycle as a whole.

Rating manipulation: dishonest ratings submitted through the direct rating path inflate or deflate scores; the stake-and-slash mechanism puts $k \times \alpha_{\text{slash}} \times S_{\text{min}}$ tokens at risk in a campaign against k distinct targets whose ratings are successfully challenged, an economic cost only where the stake is backed (Section 3.5.3). The figure assumes an established identity rating at full weight; a newly registered identity rates at reduced weight (Section 3.5.3)

and requires more ratings, and therefore more slash exposure, to achieve the same effect. *Provenance-reputation divergence*: separate chaincodes leave a window during which a lifecycle event is committed without its reputation consequence; the unified binary closes this window under Fabric's read–write set commit semantics and the stated endorsement policy. *Self-favorable reporting*: an organization records an outcome describing its own performance. Under the enforced mapping policy, the chaincode rejects any rating whose submitting organization is the organization the rated actor is registered to, on the bridge and on the direct rating path, and identities are keyed on the Membership Service Provider (MSP) identifier and certificate subject so that one organization's certificate cannot act as another's member (Section 3.5.3). Counterparty submission and endorsement establish who recorded an outcome, not that it is true; the residual exposure is collusion between organizations (Section 6.4).

The system operates under the following trust assumptions. The Fabric CA infrastructure is not compromised: X.509 certificates faithfully represent organizational identities, and each organization manages its own CA. The Raft ordering service tolerates crash faults but not Byzantine behavior; the distributed deployment evaluated here runs three consenters (Section 3.3), which tolerates the loss of one, while the single-host testbed runs one. A Byzantine orderer is outside our threat model. Evaluating the design under Fabric v3's Byzantine fault-tolerant (BFT) ordering service is future work. Endorsement collusion below the channel threshold is insufficient to commit a transaction, so under the 3-of-4 threshold policy used here, neither a single organization nor a pair acting together can commit ledger state. Endorsement is an authorization and consistency threshold, not a truth-attestation threshold: endorsers validate deterministic chaincode execution and the contract's own checks, not the external truth of a caller-supplied outcome, so no threshold setting establishes that a recorded outcome is accurate.

The following attack surfaces are explicitly out of scope. Pre-upload artifact authenticity (whether the physical material matches the submitted metadata) requires authenticated IoT sensor integration and is discussed as future work in Section 7. Full cryptographic attacks on SHA-256 or the Elliptic Curve Digital Signature Algorithm over the P-256 curve (ECDSA-P256) are not addressed; we rely on the computational hardness assumptions underlying these primitives. Compromise of a majority of organizations (>2 of 4) or denial-of-service at the network layer is treated as a deployment operations concern.

3.3. Network Topology

The reference network models four organizations: Manufacturer (Org1), Supplier (Org2), Logistics (Org3), and Regulator (Org4). Each operates an endorsing peer with a LevelDB state database and maintains its own Fabric Certificate Authority, which issues the X.509 identities under which that organization transacts. The certificate authority is required at enrollment rather than during steady-state operation, and no certificate authority process was running on any node during the measurements reported in Section 5. The ordering service is a three-node Raft cluster with consenters on the Supplier, Logistics, and Regulator nodes (D2, D3, and D4), so it tolerates the loss of any one of them; D4 held leadership throughout the measurements reported in Section 5. Ordering redundancy is orthogonal to the consistency invariant evaluated here, which is enforced at the chaincode read–write set level.

The channel endorsement policy is Fabric's MAJORITY rule evaluated over the four application organizations, which requires three of the four to endorse a transaction:

```
  MAJORITY Endorsement over {MfrMSP, SupMSP, LogMSP, RegMSP}
= OutOf(3, 'MfrMSP.peer', 'SupMSP.peer', 'LogMSP.peer', 'RegMSP.peer')
```
(1)

The chaincode definition inherits this channel policy and declares none of its own, so every transaction reported in Section 5 was validated at the 3-of-4 threshold. The threshold is stated explicitly throughout because it sets the collusion margin discussed in Section 6.4, though the consistency invariant does not depend on it.

3.4. Provenance Layer

3.4.1. Hybrid On-Chain/Off-Chain Data Model

The provenance layer anchors a minimal on-chain record that cryptographically binds each artifact. Each record comprises the event type, the submitting organization's Membership Service Provider (MSP) identifier, a transaction-level timestamp, a 32-byte SHA-256 digest of the off-chain artifact, a presigned retrieval uniform resource locator (URL), the artifact's byte count, an ECDSA-P256 signature over the core fields, and event-specific structured fields defined for each lifecycle stage. The application-level ECDSA-P256 signature binds the core record fields to the submitting operator's key pair, independent of the transport identity that Fabric attaches to the proposal, so an auditor holding only the record and the organization's published key can verify field integrity without access to the Fabric transaction envelope. The digest, not the URL, is the durable binding: presigned URLs expire, and retrieval clients derive a fresh URL from the referenced object. Integrity verification depends only on the SHA-256 digest.

3.4.2. AM Lifecycle Event Taxonomy

Table 2 specifies the on-chain field set for each lifecycle event type a deployment records; the terminal delivery event (Section 3.6) follows the same schema. The table is a schema and taxonomy, not a statement of the enforced chain: the chaincode sequence-constrains four event constants (Section 3.4.3), and event types outside that set are recorded without a predecessor check. The complete `ProvenanceEvent` record in the unified deployment also carries the common envelope fields and a reference to its bridge-generated rating, and measures a mean of 450 B when serialized in JavaScript Object Notation (JSON), measured against live world state in Section 5.4; it is the provenance-event size used in the storage characterization.

Table 2. AM lifecycle event types: on-chain schema and taxonomy. This is the event set a deployment may record, not the enforced chain; the chaincode predecessor-constrains only `MATERIAL_CERTIFICATION`, `PRINT_JOB`, `INSPECTION` and `CERTIFICATION` (Figure 2). The last column gives each label's event string in the reference deployment (Section 3.6 and the default authorization events) and how the chaincode sequences it; the correspondence between labels and strings is a deployment naming choice. `INSPECTION` is the pass/fail gate between `PRINT_COMPLETE` and `CERTIFICATION` and has no separate row here. Record sizes are reported in Section 5.4. Cert. = certification; Final. = finalization; QA = quality assurance.

Event	Event-Specific Fields	Event String; Sequencing
Material Cert.	MaterialType, BatchID, SupplierID, CertificationResult	`MATERIAL_CERTIFICATION`; constrained, creates the asset
Design Final.	DesignFileHash, Version	none in the reference policy; any string is unconstrained
Print Job Start	MachineID, BatchUsedID, BuildJobID	`PRINT_JOB_START`; authorization, gated, no sequence check
Print Completion	BuildJobID, InspectionResult	`PRINT_JOB`; constrained, after material certification [a]

Table 2. *Cont.*

Event	Event-Specific Fields	Event String; Sequencing
Post-Processing	ProcessType, InSpec	`POST_PROCESSING`; unconstrained
QA Cert.	TestStandard, FinalResult, CertID	`CERTIFICATION`; constrained, after a passed inspection
Delivery	CarrierID, ExpectedDeliveryTime, ActualDeliveryTime	`DELIVERY`; unconstrained

[a] The fault-injection conditions of Section 5.7 recorded print completion under the unconstrained string `PRINT_COMPLETION`. An unconstrained event is recorded without a predecessor check and becomes the asset's stage; an authorization event joins the history without changing the stage; extending the enforced chain requires adding predecessor rules in chaincode.

3.4.3. AM Lifecycle State Machine

Figure 2 illustrates the directed state machine encoded in `ProvenanceContract`. Four event types are sequence-constrained: `MATERIAL_CERTIFICATION`, `PRINT_JOB`, `INSPECTION` and `CERTIFICATION`. For these, the dedicated lifecycle function asserts the expected preceding state; out-of-sequence calls are rejected, preventing gaps or duplications in the constrained portion of the lifecycle record. The integrated bridge path of Section 3.6 applies the same assertion through the same helper, evaluated against the asset record it has already read and placed before any write, so the sequencing constraint holds on every path that advances one of those four stages. Event types outside that set are recorded without a predecessor check, and the recorded event string becomes the asset's new stage; a deployment that interleaves such an event between two constrained ones will therefore fail the later constrained event's predecessor check (Section 3.2). An event string equal to a stage name is rejected on every path, as is recording a constrained or authorization event type through the generic history function, so no unconstrained event can place an asset in a constrained stage.

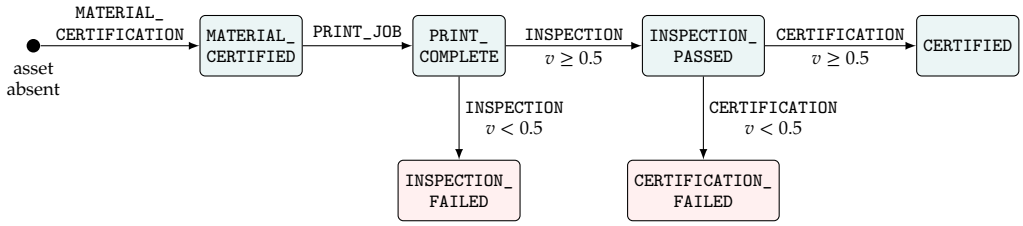

Figure 2. Sequence-constrained transitions that the chaincode enforces. Nodes are asset stages, arrows are event constants, and v is the outcome value supplied with the event; the inspection and certification events resolve to a passed or failed stage on the threshold $v \geq 0.5$. Other event types are recorded without a predecessor check.

Table 3 relates the provenance mechanisms described above to the traceability requirements that AM parts in regulated industries are held to.

Table 3. Provenance mechanisms against traceability requirements in AS9100D and ISO 13485:2016. The mapping is illustrative and is not a compliance assessment.

Requirement	Standard Clause	Chaincode Mechanism
Identification and traceability	AS9100D 8.5.2 [3]	`ProvenanceEvent` chain with SHA-256 digest binding each artifact

Table 3. *Cont.*

Requirement	Standard Clause	Chaincode Mechanism
Release of products and services	AS9100D 8.6 [3]	QA certification event, and a reputation-gated `PART_RELEASE` authorization
Traceability	ISO 13485:2016 7.5.9 [4]	Lifecycle state machine with the predecessor assertion on every path, for the sequence-constrained event types

3.5. Reputation Layer

3.5.1. Beta-Distribution Reputation Scoring

For each actor i and dimension d, the system maintains Beta parameters $(\alpha_{i,d}, \beta_{i,d})$, initialized to the symmetric prior $(2, 2)$, which is weakly informative: it centers a new actor's score at 0.5. The dimension set is a configuration parameter rather than a fixed part of the model: `InitConfig` seeds the accepted set to {quality, delivery, compliance, warranty}, and administrator-restricted `AddDimension` and `UpdateConfig` calls change it at runtime. Warranty is therefore already accepted and is read by `GetPartTrustReport`; populating it requires only a corresponding post-delivery lifecycle event and mapping rule, not the addition of a dimension. The lifecycle evaluated here terminates at delivery, so the three remaining dimensions cover every bridge-generated signal. A rating with outcome $v \in [0, 1]$ and rater weight w updates both stored accumulators in constant time:

$$\alpha_{i,d} \leftarrow \alpha_{i,d} + w \cdot v, \tag{2}$$

$$\beta_{i,d} \leftarrow \beta_{i,d} + w \cdot (1 - v). \tag{3}$$

This is the update of the Beta reputation system [9]: an outcome contributes w units of evidence, divided between success and failure in proportion to v. The submitted version of this work incremented only α when $v \geq 0.5$ and only β otherwise. That rule departed from [9], was discontinuous at $v = 0.5$, and for a constant $v = 0.9$ drove the score toward 1 instead of 0.9: after 100 such ratings from a rater of weight 0.767 it gave 0.973, against 0.880 for the update above. For binary outcomes $v \in \{0, 1\}$, which the reference mapping policy of Section 3.6 produces, the two rules are identical: $v = 1$ adds w to α and $v = 0$ adds w to β. The performance benchmarks of Section 5 submit a constant $v = 0.9$; they measure the cost of a write, which does not depend on the value, and the invariance of the reported results to the change is verified in Section 5.8.

Temporal decay is applied when a score is evaluated rather than when it is written. Every score read, gate check, and trust report derives effective parameters from the stored tuple $(\alpha_{i,d}, \beta_{i,d}, t_{\text{last}})$:

$$\hat{\alpha}_{i,d} = \alpha_0 + (\alpha_{i,d} - \alpha_0)\, \lambda^{\Delta t / T} \tag{4}$$

$$\hat{\beta}_{i,d} = \beta_0 + (\beta_{i,d} - \beta_0)\, \lambda^{\Delta t / T} \tag{5}$$

where $\lambda = 0.98$/day, $T = 86{,}400$ s, and $\Delta t = t - t_{\text{last}}$ is the time since the accumulator was last written. The stored accumulators are never rescaled; decay shrinks the evidence an evaluator sees toward the prior during inactivity and vanishes while an actor remains active. The point-estimate score is $s_{i,d} = \hat{\alpha}_{i,d} / (\hat{\alpha}_{i,d} + \hat{\beta}_{i,d})$. The update of Equations (2) and (3) is $O(1)$, preserving constant-time execution independent of rating history length.

3.5.2. Bayesian Credible Intervals

Uncertainty is quantified by the equal-tailed 95% credible interval (CrI) of the Beta posterior:

$$\mathrm{CrI}_{i,d} = \left[\ I^{-1}_{0.025}(\hat{\alpha}_{i,d}, \hat{\beta}_{i,d}),\ I^{-1}_{0.975}(\hat{\alpha}_{i,d}, \hat{\beta}_{i,d})\ \right] \tag{6}$$

where I^{-1}_q is the inverse of the regularized incomplete beta function, evaluated on the decayed parameters. The chaincode computes it by bisection on a continued-fraction evaluation of $I_x(\hat{\alpha}, \hat{\beta})$, using an exponential and logarithm built only from IEEE 754 arithmetic (Section 6.6); on ten reference cases it agrees with SciPy's Beta quantiles to within 10^{-9}. The submitted version used the Wilson score interval [27,28] with $n = \hat{\alpha} + \hat{\beta}$, which treats weighted, decayed pseudo-counts as a binomial sample size. The credible interval is the interval the model itself implies, so the revision uses it for scores, trust reports and gates. Its interpretation follows from the posterior: $\hat{\alpha} + \hat{\beta}$ is the weighted, decayed evidence the contract holds about the actor, including the four pseudo-observations of the $(2, 2)$ prior, not a count of ratings. A counterparty with no dispute history rates at weight 0.767 (Equation (7)), so each rating contributes less than one unit of evidence and the interval is wider than the posterior of the same ratings at unit weight; Section 5.9.5 measures its coverage. A wide interval signals little evidence and a narrow one an established record; the width shrinks as $O(1/\sqrt{\hat{\alpha} + \hat{\beta}})$.

3.5.3. Temporal Decay, Stake, and Dispute Resolution

Evaluation-time decay means a score cannot be banked: an actor's standing reverts toward the prior during inactivity, while the stored accumulators and the constant-time update of Equations (2) and (3) are unchanged. The prior $(\alpha_0, \beta_0) = (2, 2)$ treats new actors neutrally, avoiding both an optimistic window exploitable by Sybil identities and unfair penalization of legitimate newcomers. The reputation gate of Section 3.6.3 applies only to authorizing designated activities, never to recording outcomes, so a new actor's outcomes are recorded from its first event. With the $(2, 2)$ prior, a gate whose *minScore* does not exceed 0.5 admits new actors and begins discriminating as evidence accumulates; operators can also set *minEvents* no higher than the number of observations a probationary workload produces.

Every rating, whether submitted directly through `SubmitRating` or generated by the bridge, carries a weight w and requires the submitting identity to hold a minimum stake $S_{\min}$ (default 10,000 tokens); an unstaked identity's submission is rejected on either path. A token here is an internal accounting unit, not a Fabric-native asset or a cryptocurrency: each actor's stake is a floating-point balance field in a per-actor record that the chaincode serializes as JSON and writes to world state under a `STAKE` key, so it is ordinary chaincode state and moves only through the contract's own stake, dispute, and slash functions. How a balance is funded or redeemed, and what one unit is worth outside the ledger, is a consortium governance decision taken at deployment, outside the protocol described here. In the implementation evaluated here, `AddStake` credits the caller's own balance without any funding check, so any identity can reach $S_{\min}$. The stake therefore records each rater's exposure in accounting units, and it deters dishonest rating economically only if a deployment backs every unit off the ledger, for example with an authenticated deposit, and restricts who may credit a balance, which the evaluated chaincode does not. The weight is derived on chain from the submitter's record as a rater. For rater r and dimension d the contract keeps a meta-reputation accumulator $(\alpha^m_{r,d}, \beta^m_{r,d})$ with the same $(2, 2)$ prior and decay. It changes only when a dispute over one of r's ratings in d is resolved, gaining one unit of α^m if the rating is upheld and one unit of β^m if it is overturned. With decayed values $\hat{\alpha}^m, \hat{\beta}^m$ and $N = \hat{\alpha}^m + \hat{\beta}^m$,

$$w_{r,d} = \min\left(w_{\max},\ \max\left(w_{\min},\ \frac{\hat{\alpha}^m}{N}\left(1+\sqrt{\frac{N}{N+10}}\right)\right)\right), \tag{7}$$

where the bracketed term is the evidence factor: it rises from 1 toward 2 as the meta-reputation accumulates evidence, reaching 1.5 at $N = 10$. The defaults are $w_{\min} = 0.1$ and $w_{\max} = 5.0$. A rater with no dispute history has $\hat{\alpha}^m = \hat{\beta}^m = 2$ and $w = 0.5\,(1+\sqrt{4/14}) = 0.767$. Because the meta-score is at most 1 and the evidence factor is below 2, $w < 2$ always, so the upper bound never binds; the lower bound binds only after 34 overturned ratings with none upheld. A rater's influence therefore changes only through dispute outcomes: ten upheld disputes raise its weight to 1.51, and ten overturned ones lower it to 0.25 (Section 5.9.1). Because the evidence factor counts resolved disputes whatever their verdict, an even record also raises the weight: one upheld and one overturned verdict take a new rater from 0.767 to 0.806 (Section 5.9.3). A rater can thus gain some influence from disputes it loses half the time, although its weight stays below 1 while at least half of its resolved disputes went against it; a consortium that wants every overturned rating to cost influence would have to count only upheld verdicts in the evidence factor. The stake functions as a participation bond: it is the value at risk when a rating is challenged and found dishonest, and it applies uniformly because a bridge rating, although tied to a recorded lifecycle outcome rather than to a free-standing human judgment, is still submitted by an identity that selects which outcome value to record. Stake calibration and equilibrium analysis are out of scope for this paper. The stake is not a reputation signal: a rater's weight derives from its dispute record, not from its balance, which avoids the confusion of wealth with reputation that token design work warns of [23].

Identities are keyed on the MSP identifier together with the certificate subject's common name. The submitted version keyed them on the common name alone, which a certificate issued by another organization's certificate authority can duplicate: the default client identities of both organizations in the test network carry the common name `user1`, and the submitted binary let the second spend the first's stake (Section 5.9.2). Two guards apply to every rating. The rated-actor label may not equal the submitter's identity; and when the mapping policy is enforced (Section 3.6), the submitting organization may not be the organization the rated actor is registered to, on the bridge or on the direct path. Any stored rating can be challenged by the rated actor's organization: `InitiateDispute` locks a dispute cost from the challenger's stake; the dispute is reviewed off-chain under the consortium's agreed procedure, and a designated arbitrator identity records the verdict through `ResolveDispute`. A verdict of `overturned` reverses both halves of the rating's update and slashes the rater's stake by rate α_{slash} (default 0.10); a verdict of `upheld` leaves the rating standing. Either verdict updates the rater's meta-reputation and so its future weight. The dispute is also where attribution of responsibility is contested: an adverse outcome recorded against an actor whose cause lies elsewhere, such as a powder lot behind a failed inspection, is overturned through this path (Section 5.9.3). `SetReputationGate`, `SetMappingRule`, the enforcement switch and `ResetStake` are restricted in chaincode to administrator identities seeded at initialization.

3.6. Provenance-to-Reputation Bridge

3.6.1. Bridge Mapping Rules

Many AM outcome signals are already recorded: a certification result indicates whether a supplier's material met its standard, and a delivery timestamp indicates whether logistics met its deadline. The bridge carries such an outcome into the reputation state within the same Fabric transaction that records the event. The submitting organization evaluates the outcome condition and supplies the rated actor, the dimension and the outcome value; the chaincode enforces who may do so. Each actor label is bound in an on-chain registry to the organization that registered it and to a supply-chain role, and only an administrator

of that organization can register or change the label, so no other organization can claim it. Each rating-producing event type carries an on-chain mapping rule naming the role it rates, the dimensions it may rate, and the roles the submitting organization must hold. When the mapping policy is enforced, which is the default at initialization, the chaincode rejects a submission whose event type has no rule, whose dimension the rule does not allow, whose rated label is unregistered or registered with another role, whose submitting organization is the rated actor's organization, or whose submitting organization holds none of the rule's submitter roles. Roles are declared, not accredited: an organization holds a role once its administrator has registered an actor with that role, and the chaincode does not check the declaration. The policy therefore ensures that an outcome comes from a counterparty that has declared a submitter role; admitting an organization to a role, for instance as an inspector, is a consortium governance step outside the chaincode. Table 4 is the reference policy as installed on the ledger. It differs from the submitted version in one respect that enforcement makes unavoidable: every rating-producing event is submitted by a counterparty of the rated actor, never by the actor's own organization, so print completion and post-processing are rated by the inspecting organization instead of the manufacturer, and delivery by the receiving organization rather than by the carrier. Continuous AM measurements (e.g., dimensional tolerances, post-process surface roughness) are converted to binary outcomes using acceptance thresholds from deployment-specific quality plans, inspection standards, or AS9100/ISO 13485-compliant acceptance procedures; the chaincode records the outcome a counterparty submits and does not evaluate the measurement itself.

The delivery rule covers the terminal handoff: the receiving organization records the delivery event when it takes the part over, rating the carrier, and the event enters the ledger through the same bridge path as the other lifecycle event types of Table 2. Delivery is one of the unconstrained event types, so it carries no predecessor assertion and does not appear in Figure 2. Its record follows the same hybrid on-chain schema, anchoring event-specific fields and a SHA-256 digest of the off-chain delivery documentation.

Table 4. The reference provenance-to-reputation mapping policy, installed as on-chain rules. Under enforcement the chaincode requires the rated label to be registered with the rated role, the dimension to be one the rule allows, and the submitting organization to hold a submitter role and to differ from the rated actor's organization; it then applies the Bayesian update atomically with the provenance write and the link entry. The submitter evaluates the stated condition. Dim. = reputation dimension; Mfr. = manufacturer; std. = standard; t_a and t_e = actual and expected delivery times.

Event (String)	Rated	Dim.	Submitter Role	$v = 1$ Condition
Material Cert. (`MATERIAL_CERTIFICATION`)	Supplier	Compliance	Manufacturer, inspector	Certified
Print Completion (`PRINT_JOB`)	Mfr.	Quality	Inspector	No in-process nonconformance
Inspection (`INSPECTION`)	Mfr.	Quality	Inspector, regulator	Pass
Post-Processing (`POST_PROCESSING`)	Mfr.	Quality	Inspector	In-spec
QA Cert. (`CERTIFICATION`)	Supplier	Quality, compliance	Regulator	Certified; std. listed
Delivery (`DELIVERY`)	Logistics	Delivery	Customer, manufacturer	$t_a \leq t_e$

A separate test exercised all six rules under enforcement with both outcomes (Section 5.9.2); the lifecycle scenario of Section 5.9.3 exercised `PRINT_JOB`, `INSPECTION` and `DELIVERY` with both outcomes and the two certification rules with passing outcomes, and the identity and mapping-policy tests of the same section every rejection condition. The performance benchmarks used opaque actor labels with enforcement switched off; the cost of enforcement is reported separately (Section 5.8).

3.6.2. Bridge Algorithm

Algorithm 1 formalizes the atomic bridge. Every check precedes the first committed effect: a failed check aborts the transaction during endorsement, so nothing is written. The bridge records outcomes and is never gated.

Algorithm 1 Atomic Provenance-to-Reputation Bridge

Input: Asset A; outcome event e; actor label i; dimension d; outcome $v \in [0,1]$; submitter r of organization o
Output: Provenance event, rating, accumulator update and link entry committed atomically, or transaction aborted with no writes

require $e \notin \mathcal{A}$ {$\mathcal{A}$: authorization event types}
require e is not a stage name **and** PredecessorOK$(A.stage, e)$
require $i \neq r$
if mapping policy enforced **then**
  $\rho \leftarrow$ Rule(e); **require** $\rho \neq \perp$ **and** $d \in \rho.dims$
  $a \leftarrow$ Registry(i); **require** $a \neq \perp$, $a.role = \rho.rated$, $a.org \neq o$
  require o holds a role in $\rho.submitters$
end if
require Stake$(r) \geq S_{\min}$; $w \leftarrow$ Equation (7)
WriteProvenanceEvent(A, e); $A.stage \leftarrow$ NextStage$(A.stage, e, v)$
$(\alpha, \beta) \leftarrow$ Load(i, d); $\alpha \leftarrow \alpha + wv$; $\beta \leftarrow \beta + w(1 - v)$
Save$(i, d, \alpha, \beta, t_{tx})$; WriteRating$(r, i, d, v, w, txID)$
WriteLink$(A.id, txID, i, d)$

3.6.3. Reputation-Gated Authorization

A gate authorizes a future activity; it never applies to an outcome. The submitted version checked the gate inside the bridge, so a gate on an outcome event would have aborted the record of an outcome that had already happened: a failed inspection of a manufacturer below the gate threshold would never have been recorded. The revision separates the two. Gates may be set only on authorization event types, a configured set that by default contains `PRINT_JOB_START` and `PART_RELEASE`; `SetReputationGate` rejects any other type, and the bridge rejects authorization types because they carry no outcome. Authorization is requested through `AuthorizeActivity` (Algorithm 2). It either appends an authorization event to the asset's history without changing its lifecycle stage, or refuses without writing anything, since no activity has occurred and there is nothing to record. A gate that cannot be read refuses. A *maxCI* of zero disables the interval condition, and the event count m is the stored number of ratings, which decay does not reduce.

Algorithm 2 Reputation-Gated Authorization

Input: Asset A; authorization type $a \in \mathcal{A}$; actor label i
Output: Authorization event appended to A's history, or refusal with no writes

$g \leftarrow$ GetState(`REP_GATE`$\|a$) {read failure: refuse}
if $g \neq \perp$ **and** $g.enforced$ **then**
  $(\hat{\alpha}, \hat{\beta}, m) \leftarrow$ LoadDecayed$(i, g.dim)$
  $\bar{s} \leftarrow \hat{\alpha}/(\hat{\alpha} + \hat{\beta})$; $\omega \leftarrow$ width of Equation (6)
  if $\bar{s} < g.minScore$ **or** $m < g.minEvents$ **or** $(g.maxCI > 0$ **and** $\omega > g.maxCI)$ **then**
    refuse {nothing written}
  end if
end if
AppendAuthorization(A, a, i) {stage unchanged}

4. System Implementation

4.1. Chaincode Architecture

The unified chaincode binary registers three Go structures, Provenance Contract, Reputation Contract and Integration Contract, via the `fabric-contract-api-go v2` multi-contract runtime. Fabric admits three arrangements for a logical provenance plus reputation operation. First, a single chaincode binary, adopted here, produces one read–write set per operation. Second, two chaincodes linked by a same-channel `InvokeChaincode` call also produce a single read–write set, because the callee's writes are folded into the caller's transaction; the set spans two namespaces, each validated against its own endorsement policy, so both chaincodes must be installed on a common set of endorsing peers and kept interface-compatible. Third, two chaincodes orchestrated by the client as two separate transactions produce two independently validated read–write sets and are not atomic. The first two are atomic by Fabric's commit semantics. Section 5.8 measures both, compiled from the same source, and Section 5.7 fault-injects all three. The choice between the two atomic arrangements is a trade-off, not a question of correctness: co-deployment is one deployment unit with one endorsement policy, whose upgrade redeploys all three contracts, while `InvokeChaincode` allows separate upgrades and policies at the cost of a second deployment unit, an interface to keep compatible and a chaincode-to-chaincode call in every bridge invocation. CGO is disabled for fully static binaries. Timestamps for reputation accumulator updates and bridge link records use `ctx.GetStub().GetTxTimestamp()` rather than the Go runtime clock, and decay and the credible interval use only IEEE 754 arithmetic (Section 6.6), so every endorser computes the same write set.

4.2. CCAAS Deployment

On the single-host two-organization testbed, Chaincode-as-a-Service (CCAAS) was adopted: the chaincode process runs in an independently managed container, and each peer connects via gRPC Remote Procedure Call (gRPC) using a connection address provided by the lifecycle package. The chaincode was compiled as a static Linux/amd64 binary (18 MB), packaged into a minimal Alpine 3.19 Docker image, and deployed as two containers, one per organization, on a shared Docker bridge network. A single connection-address template allows a single package identifier to be installed on both peers; intra-container round-trip latency was measured at $<$0.5 ms.

4.3. Channel Configuration Tuning

The Fabric default `BatchTimeout` of 2000 ms was reduced to 10 ms using Fabric's channel configuration toolchain (`configtxlator` and `peer channel update`), and `PreferredMaxBytes` was raised to 2 MB; `MaxMessageCount` is 100 on this channel. Section 5.2 reports the measured effect of the final step of this tuning, from 50 ms to 10 ms. The sequential workload is closed loop: the client submits the next transaction only after the previous one commits, so the queue never approaches `MaxMessageCount` and the count ceiling is not binding. The 10 ms setting applies to the single-host two-organization testbed. The distributed four-organization testbed of Section 4.4 uses `BatchTimeout` = 50 ms with `MaxMessageCount` = 10.

4.4. Distributed Four-Organization Testbed

The unified chaincode was subsequently deployed across four physical desktop machines connected via an Ethernet switch. Table 5 describes the hardware and inter-node latency. The four machines are not identical. D2 provides four hardware threads where the others provide twelve, so the testbed reflects the mixed hardware a small consortium is likely to assemble. The topology mirrors a realistic AM consortium: Manufacturer (Org1), Supplier (Org2), Logistics (Org3), and Regulator (Org4), each operating its own physical

infrastructure. Each organization runs one peer together with its own CCAAS chaincode container on its own machine, so every peer executes chaincode locally. The ordering service runs as three Raft consenters, one each on D2, D3 and D4, so it tolerates the loss of any single node; D4 held leadership throughout the reported measurements. The software development kit (SDK) client runs on D1. Channel configuration: `BatchTimeout` = 50 ms, `MaxMessageCount` = 10, `PreferredMaxBytes` = 2 MB, 3-of-4 threshold endorsement.

Table 5. Distributed testbed: node roles (Desktops D1–D4), hardware, and round-trip time (RTT) to D1 (ms). Mfr. = manufacturer; Reg. = regulator; SDK = software development kit; ord. = Raft orderer; T = hardware threads; RAM = random-access memory.

Node	Role	Processor	RAM	RTT to D1
D1	Org1 (Mfr.); SDK	i7-8700 (12 T)	8 GB	n/a
D2	Org2 (Supplier); ord.	Xeon E5-1603 v3 (4 T)	16 GB	0.37 ms
D3	Org3 (Logistics); ord.	Xeon E-2136 (12 T)	16 GB	0.65 ms
D4	Org4 (Reg.); ord. leader	Xeon E-2136 (12 T)	16 GB	0.53 ms

The complete system can be reproduced on any Linux host with Docker, Go 1.21+, Node.js 18+, and Fabric v3.1.4 binaries.

5. Evaluation

All measurements are obtained from direct end-to-end instrumentation of a live Hyperledger Fabric deployment; no values are simulated or extrapolated unless explicitly noted. Methodology follows the conventions of the Hyperledger Foundation reference benchmarks [29], although those benchmarks target Fabric 2.5 and the implementation evaluated here runs on the Fabric 3.1 releases stated in Section 5.1.

5.1. Experimental Setup

Evaluation was conducted on two testbeds, which run different Hyperledger Fabric releases: the single-host testbed runs Fabric v3.1.0 and the distributed four-organization testbed runs Fabric v3.1.4. Both execute the same chaincode source revision, compiled separately for each deployment.

The distributed four-organization LAN is the primary performance testbed. The single-host two-organization deployment is an auxiliary controlled testbed, used for the bridge-overhead decomposition, the `BatchTimeout` probe, the storage measurement, the contention stress test and the fault-injection study. The distributed testbed spans four desktop machines (Section 4.4) with `BatchTimeout` = 50 ms and 3-of-4 threshold endorsement; the single-host testbed is one Ubuntu 24.04 LTS host (Intel Core i7-13700H, 16 GB RAM) running a two-organization network via CCAAS with the channel tuned to `BatchTimeout` = 10 ms (Section 4.3). Because the two differ in Fabric release, channel configuration, endorsement policy and physical topology, figures from one are not directly comparable to figures from the other; each is reported against its own testbed throughout and no cross-testbed ratio is drawn.

Two versions of the chaincode are evaluated. The submitted implementation (v1.0) produced the latency, throughput and failure measurements of Sections 5.2, 5.3 and 5.5.2, the fault-free baseline of Section 5.7.2 and the v1.0 conditions of Section 5.7.3. The revised implementation (v1.1) of Section 3 produced the lifecycle-enforcement test of Section 5.6, the v1.1 conditions of Section 5.7.3, and the experiments of Sections 5.8 and 5.9. On the single-host ledger, the two versions ran side by side, each in its own chaincode namespace, so v1.1 could be measured against v1.0 in one session at one ledger state (Section 5.8). The v1.1 experiments used the binary built from commit `ecd829a`, except the lifecycle scenario of

Section 5.9.3, which used commit `d1f2efd`: that commit marks two optional fields in the return schema of the history queries, which had failed for assets holding an authorization event, and changes no write path. The four-desktop experiments of Section 5.5.4 and the mapping-coverage test of Section 5.9.2 used binaries built from the tagged release, whose chaincode is identical to that of `d1f2efd`. Supplementary Table S5 lists, for every experiment of the revision, the testbed, the chaincode binary, the driver, the output and the software versions.

Every write transaction in the benchmark records a genesis material certification event against a newly created asset, so no timed transaction advances an asset through a later lifecycle transition; the state-machine assertion of Section 3.4.3 is exercised by the separate correctness experiment of Section 5.6, not by the measured workload, and later transitions, failed outcomes, gates, disputes and decay by the experiments of Section 5.9. Throughput is measured with a closed-loop generator that holds a fixed number of transactions in flight and dispatches a replacement only as an earlier transaction commits, so the offered arrival rate falls as latency rises, and the throughput figures reported below are closed-loop values at the stated concurrency rather than open-loop capacity.

5.2. Single-Host Write Latency

Table 6 reports end-to-end write latency for each of the three contract types, with each phase boundary timestamped, so the total decomposes into measured parts. For the provenance write, endorsement takes a median 12.3 ms and ordering and commit 35.0 ms; medians are not additive, so the decomposition locates the cost rather than partitioning the 47.3 ms median. The second figure covers the ordering wait (at most `BatchTimeout` = 10 ms), Raft replication, validation system chaincode (VSCC) validation, the multi-version concurrency control (MVCC) check and the LevelDB write. Endorsement rises across the three contract types, 12.3 ms for provenance, 14.1 ms for reputation and 16.4 ms for the bridge write, while ordering and commit stay flat at 33.7–35.0 ms, so the bridge write's 2.8 ms median overhead over standalone provenance (5.9%; 50.1 versus 47.3 ms) is incurred in chaincode execution, not in the consensus path; it is smaller than the run-to-run range reported beneath the table. Read latency (P50: 7.6 ms) reflects peer-only evaluation without an ordering phase. Measured on one deployment at both settings, reducing `BatchTimeout` from 50 ms to 10 ms (Section 4.3) lowers mean sequential provenance write latency from 76.6 ms to 40.1 ms and leaves concurrent throughput at W=100 essentially unchanged (+2.7%), because block occupancy and block rate trade off and their product is conserved; that deployment held 79,424 counted keys, more than for Tables 6 and 7, so these two figures are comparable to each other but not to those tables.

5.3. Single-Host Concurrent Throughput

Table 7 reports sustained concurrent throughput at $W = 100$ transactions in flight. All three distinct-key workloads achieve 346–390 transactions per second (TPS) with every submitted transaction committing and no MVCC conflicts, as expected since each transaction reads and writes a unique state key. Block occupancy on this testbed is 6.5–7.5 transactions per block against a `MaxMessageCount` of 100, so blocks were underfilled and the count ceiling was not the active constraint. The same is true of the distributed testbed (Section 5.5.2), where occupancy is 3.3–3.5 against a limit of 10.

The high-contention scenario (writes targeting the same actor's reputation state) yields 7.78 TPS with only 27 of 1500 transactions committing, as expected when transactions in a block involve the same key [22]. This is a conservative stress case, not the dominant part-level provenance workload, where each part carries a unique identifier. However, large

suppliers receiving many concurrent reputation updates may require sharding or batching of actor-dimension accumulators in production deployments.

Table 6. Sequential end-to-end write latency of the submitted chaincode (v1.0), both testbeds (500 transactions per type per run × 3 runs, $W = 1$; medians across runs). Single-host: Fabric v3.1.0, two organizations, `BatchTimeout` = 10 ms. Distributed 4-org LAN: Fabric v3.1.4, `BatchTimeout` = 50 ms, 3-of-4 threshold endorsement. Latencies are in milliseconds. TPS = transactions per second; P50 and P95 = 50th and 95th percentiles; LAN = local area network.

Testbed	Operation	TPS	Mean	P50	P95	Committed
Single-host	Provenance write	20.48	48.6	47.3	60.3	1500/1500
	Reputation write	19.57	50.9	48.8	62.5	1500/1500
	Bridge write (atomic)	20.11	49.5	50.1	53.4	1500/1500
	Trust-report read	131.11	7.6	7.6	9.0	1500/1500
Distributed	Provenance write	2.32	431.1	402.3	679.8	1500/1500
	Reputation write	2.27	440.1	422.1	680.3	1500/1500
	Bridge write (atomic)	2.26	442.4	412.3	689.9	1500/1500
	Trust-report read	251.59	4.0	3.9	4.8	1500/1500

Range across the three runs, single-host: provenance write: P50 47.2–56.2 ms; Reputation write: P50 47.7–51.2 ms; Bridge write (atomic): P50 49.4–54.7 ms; Trust-report read: P50 7.6–7.8 ms. Range across the three runs, distributed: provenance write: P50 401.6–421.9 ms, TPS 2.28–2.34; Reputation write: P50 411.2–423.0 ms, TPS 2.25–2.28; Bridge write (atomic): P50 402.7–413.3 ms, TPS 2.24–2.30; Trust-report read: P50 3.8–3.9 ms, TPS 249.95–255.58.

Table 7. Sustained concurrent throughput of the submitted chaincode (v1.0), both testbeds (2000 transactions per type per run × 3 runs for the distinct-key conditions, 500 per run for high contention; medians across runs). TPS counts successfully committed transactions. Single-host: Fabric v3.1.0, two organizations, `BatchTimeout` = 10 ms, measured at a state size of 7509 counted keys (ledger height 4534); single-host concurrent throughput is state-dependent and this figure is not comparable to a measurement taken at a different state size. Distributed 4-org LAN: Fabric v3.1.4, `BatchTimeout` = 50 ms, 3-of-4 threshold endorsement.

Testbed	Operation	W	Steady TPS	Total Window TPS	P50 (ms)	Tx/ Block	Blocks/s	Committed
Single-host	Provenance write	100	389.69	397.35	243.1	7.52	51.01	6000/6000
	Reputation write	100	355.07	362.78	259.5	6.99	49.21	6000/6000
	Bridge write (atomic)	100	346.22	357.83	274.5	6.45	52.53	6000/6000
	High contention †	100	7.78	8.13	225.9	0.14 ‡	47.96	27/1500
Distributed	Provenance write	20	11.68	11.97	1609.2	3.29	3.63	6000/6000
	Reputation write	20	11.93	12.17	1597.0	3.40	3.57	6000/6000
	Bridge write (atomic)	20	12.16	12.52	1557.3	3.48	3.54	6000/6000
	High contention †	20	2.35	2.43	453.1	0.74 ‡	2.99	83/1500

Steady TPS is measured over the interval in which exactly W transactions were in flight; total-window TPS spans the whole run. Blocks per second and transactions per block are whole-run quantities, so their product approximates the total-window column, not the steady column; for the distinct-key rows it falls short by at most 5.3%, because the block count's window comes from the run manifest, not from the transaction log. The submitted version computed the single-host block rate over the steady window, which overstated it. The multi-version concurrency control (MVCC) conflict rate was zero across all distinct-key conditions on both testbeds; under high contention on the distributed testbed 39.47% of submissions were MVCC-rejected. † Single hot key. Under high contention on the single-host testbed only 9 of 500 transactions committed per run, of which 6 fell inside a steady window of 0.77 s; both throughput figures for that row rest on those few commits and are not comparable to the distinct-key rows, bounding the serialization rate rather than a sustained rate. Failure classes for the distributed high-contention condition are given in Table 8. ‡ Transactions per block is committed/blocks; blocks also carry transactions that fail validation, so this understates occupancy when the failure rate is high.

5.4. Storage Footprint

Record sizes were measured against live world state as the serialized length of the value the chaincode returns, excluding the key and LevelDB framing. For the submitted

chaincode, the three event types on the ledger measured 429–487 B (median over six records each; mean of the type medians 450 B), a `Rating` 472 B, a reputation accumulator 247 B and the `Asset` record 195 B; no query returns its link entry, which is carried forward at 251 B from the June 2026 measurement. The revised chaincode returns link entries, so all its sizes were measured, on bench assets of identical shape written by both versions in one session: the provenance event is 460 B under both; the rating grows from 445 to 453 B because the rater is keyed by MSP and certificate subject; the accumulator grows from 229 to 246 B because the two-sided update stores a non-integer β; the asset is 184 B under both; and the link entry is 268 B.

Each bridge invocation writes exactly one provenance event, one rating and one link entry: approximately 1.17 kB of world-state values with the submitted chaincode, and 1181 B, every component measured, with the revised one. The accumulator and the asset record are updated in place after creation, so their contribution grows with the number of actor-dimension pairs and assets, not with the event count. No complete-lifecycle projection is reported, because a deployment defines its own mix of constrained and unconstrained events.

5.5. Distributed Four-Organization Evaluation

This testbed uses `BatchTimeout` = 50 ms. Its sequential latency reflects that setting together with cross-machine endorsement, Transport Layer Security (TLS), and block delivery to four independent physical machines, not any property of the unified architecture.

5.5.1. Sequential Write Latency

Each distributed condition was run three times with a 60-second cooldown, by a closed-loop driver that dispatched a replacement transaction whenever one resolved, holding one transaction in flight for sequential conditions and twenty for concurrent ones (500 and 2000 transactions per run). The first transactions of each run, 5 sequentially, 100 concurrently and 50 under high contention, are treated as warm-up and excluded (Supplementary Methods S1 gives the rule), and throughput is computed over the steady-state interval in which the nominal number of transactions was in flight. Every transaction was recorded when it resolved, all statistics are recomputed from those records by a separate program, and failures fall into ten classes; a run was accepted only if submissions equalled commits plus failures and all peers agreed on ledger height and block hash. The two exclusion grounds, a leadership change within the run window and a violation of that identity, were fixed before measurement, and no run was excluded on any other. Zero-failure observations carry rule-of-three 95% upper bounds [30] below 0.20% sequentially ($n = 1500$) and 0.05% concurrently ($n = 6000$) per operation type. The full protocol is given in the Supplementary Materials.

Table 6 reports end-to-end write latency (1500 transactions per type; medians across three runs; `BatchTimeout` = 50 ms, 3-of-4 threshold endorsement). The 431–442 ms mean latency reflects cross-machine gRPC endorsement, TLS overhead, Raft log replication, and block delivery to four physical nodes. All three write types exhibit 0 observed failures across 1500 transactions per operation type, with 95% upper bound on the true failure rate below 0.20%. Read latency (mean 4.0 ms) bypasses the ordering phase entirely.

5.5.2. Concurrent Throughput and Scalability

Table 7 reports concurrent write throughput (2000 transactions per type per run × 3 runs; medians reported) at $W = 20$ with distinct actor-dimension pairs. All three write types achieve 11.7–12.2 TPS with 0 observed failures and 0 observed MVCC conflicts across 6000 transactions per type, confirming that the AM-realistic distinct-key workload operates without contention.

Throughput on this testbed is the product of two measured quantities, the rate at which blocks are cut and the number of transactions each block carries. Blocks were cut at 3.5–3.6 per second. Blocks carried 3.3 to 3.5 committed transactions against a configured `MaxMessageCount` of 10, so the message-count ceiling was not the active constraint; raising `MaxMessageCount` alone cannot help a workload that never reaches it. Across the sweep of Figure 3, occupancy stayed between 3.3 and 4.9 transactions per block over a four-hundredfold range of offered load.

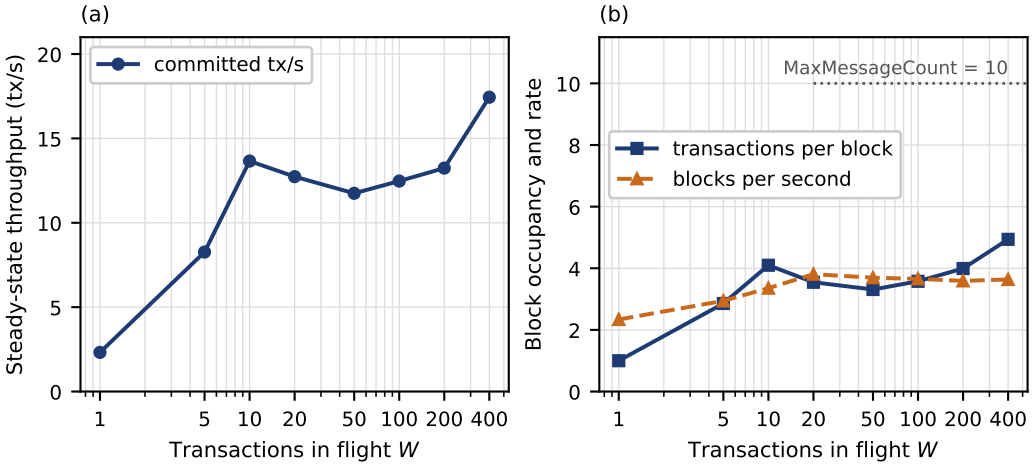

Figure 3. Concurrency sweep on the distributed testbed, three runs per level, medians reported. (**a**) Steady-state committed throughput against the number of transactions held in flight, using the steady-window definition of Section 5.5.1. Throughput rises to $W = 10$, is flat between $W = 10$ and $W = 200$, and rises again at $W = 400$, the highest level measured, reaching 17.44 TPS; it had not stopped rising there, so this sweep does not establish an upper bound on the throughput of this system. (**b**) Block production rate and block occupancy, both over the whole run; the submitted figure divided whole-run blocks by the shorter steady window, which overstated the rate. Occupancy never approaches the configured `MaxMessageCount` of 10, staying at or below 4.9 transactions per block at every level, while the block rate stays between 2.3 and 3.8 blocks per second.

Under high contention only 83 of 1500 transactions committed, and the dominant failure is not MVCC conflict (Table 8). 58.2% of failures are endorsement mismatches: the endorsing peers returned different read–write sets, so the transaction was never assembled and never reached the ordering service. The remaining 41.8% are MVCC read conflicts, which do occupy block space before being invalidated; they explain the sub-unity transactions per block of that condition in Table 7.

Table 8. Failure classes under high contention, submitted chaincode (v1.0), distributed 4-org LAN testbed (single hot key, $W = 20$, 500 transactions per run × 3 runs). The eight remaining failure classes recorded zero occurrences and no failure was left unclassified.

Failure Class	Count	% of Failures
`ENDORSE_MISMATCH`	825	58.2
`MVCC_READ_CONFLICT`	592	41.8

83 of 1500 committed; 1417 failures across 2 classes, all others zero.

5.5.3. Resource Utilization

Every container on all four nodes was sampled from cgroup v2 every two seconds throughout both the concurrent conditions and the concurrency sweep, at idle (every sample outside a run window) and under load (every sample inside one). Peers hold 490 MB at idle and 838 MB at peak, and the Raft leader holds roughly two and a half times

its followers, so leadership is a node-sizing property and not only an availability one. Full per-container figures are given in the Supplementary Materials (Table S1).

5.5.4. Open-Loop Load, Wide-Area Latency, Scale and Soak

The measurements above are closed-loop runs of a few minutes on a LAN with four organizations. To extend them in each of these respects, the revised chaincode was deployed on the four-desktop testbed on 12–13 October 2026 as a new namespace beside the submitted one, on the same channel: three Raft consenters, `BatchTimeout` = 50 ms, `MaxMessageCount` = 10, MAJORITY endorsement, and one chaincode server per desktop. Every transaction was the bridge write under the enforced mapping policy: Org1 records the certification of a new asset and rates one of 3000 Org2 suppliers, taken in turn. The client ran on D1 against Org1's gateway peer, as before. Open-loop load drew Poisson arrivals from a seeded generator, independent of completions, and every transaction's endorsement, submission and commit times were recorded; no client process exceeded 34% of a CPU. Before the runs, a load step was defined as sustained when committed throughput reached 90% of the offered rate, at most 1% of transactions failed and the 95th percentile of latency stayed within 5 s.

A same-day control compared the revised chaincode with the submitted one under identical conditions; run alternately on the same network, the two performed alike. One transaction at a time, median latency was 428 and 413 ms for the submitted chaincode and 413 and 404 ms for the revised one. With 20 transactions in flight, throughput was 8.8 and 13.6 tx/s against 10.2 and 12.8 tx/s; the spread between runs is wider than the difference, and both bracket the 11.7–12.2 tx/s of Section 5.5.2. The figures below therefore describe the testbed, not a cost of the revision.

Figure 4 shows three ascending open-loop sweeps. Committed throughput tracked the offered rate, and the highest sustained rate was 10, 8 and 10 tx/s in the three sweeps. Median latency rose from 563 ms at 2 tx/s to 3.1 s at 10 tx/s as transactions queued for ordering and commit. Above that the network fell behind the offered load: at 14 tx/s offered, committed throughput reached only 11.8–12.5 tx/s, while the 95th percentile of latency grew to 18–23 s. About 12 tx/s is therefore the highest rate this testbed carries with bounded latency, consistent with the closed-loop throughput at $W = 20$ of Section 5.5.2. It is a saturation point, not an absolute maximum: with up to 400 transactions in flight, the closed-loop sweep of Figure 3 committed up to 17.44 TPS with the submitted chaincode.

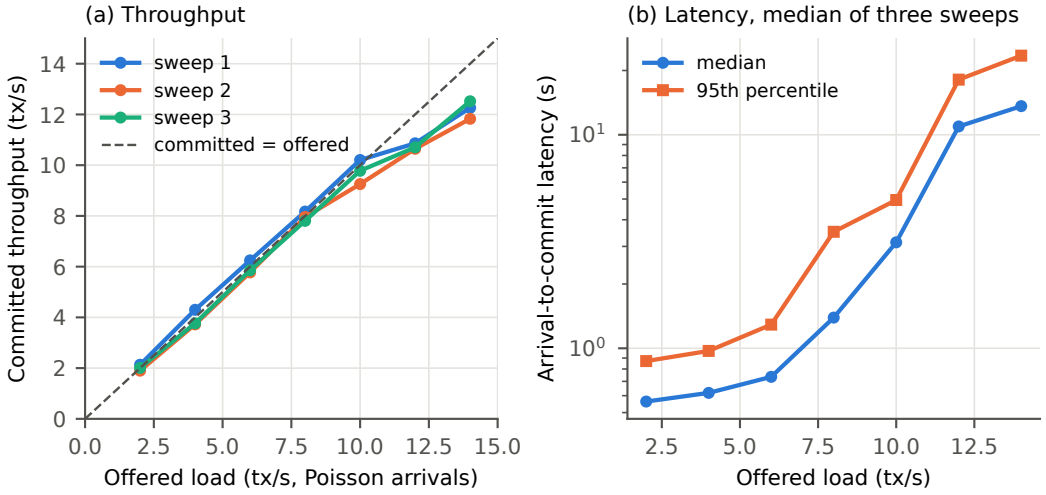

Figure 4. Open-loop load on the four-desktop testbed, revised chaincode: three ascending sweeps, Poisson arrivals, 150 s per step with the first 30 s excluded. (**a**) Committed throughput against offered load. (**b**) Arrival-to-commit latency, median over the three sweeps of each step's median and 95th percentile.

Table 9 emulates wide-area distance by delaying every packet that leaves a desktop (Linux `netem`), raising the round-trip time between desktops from 0.5 to 201 ms. The client and Org1's gateway share D1, and each peer reaches its chaincode on its own machine, as they would within one organization's site. One transaction at a time, median latency rose from 401 to 1,178 ms; from 50 ms of added delay on, each millisecond of added round-trip time added 3.8–3.9 ms, about four round trips per transaction across endorsement, ordering and delivery. At 6 tx/s, about half the capacity, every delay was sustained without a failure. With 20 transactions in flight, throughput varied between 6.1 and 12.2 tx/s without a trend across delays; we report it without interpretation. The delay is uniform and constant, so this experiment shows the latency cost of distance but not the effect of variable latency or loss on a real wide-area network (WAN).

Table 9. Emulated wide-area latency on the four-desktop testbed, revised chaincode. Every packet leaving a desktop is delayed by the added delay, so the round-trip time between desktops grows by twice that amount; the RTT column is the measured mean from D1 to the other three. Latencies are from arrival to commit (median/95th percentile). Closed loop: 150 transactions at 0–25 ms and 100 at 50–100 ms with $W = 1$; 1000 and 600 with $W = 20$. Open loop: 90 s at 6 tx/s, the first 15 s excluded.

Added Delay (ms)	RTT (ms)	$W = 1$ Latency (ms)	6 tx/s: Committed tx/s	6 tx/s: Latency (ms)	Failures	$W = 20$ tx/s
0	0.5	401/566	6.09	755/1195	0	11.0
10	20.7	557/735	5.94	832/1347	0	12.2
25	50.8	579/653	5.82	1430/4581	0	6.1
50	100.9	790/868	5.78	1316/1871	0	8.2
100	200.9	1178/1246	6.26	1792/2509	0	8.8

Note: a first pass also offered 15 tx/s, above the testbed's capacity. It saturated at 0 ms, and at 10 ms a harness error reused asset identifiers, so the chaincode rejected every transaction; neither step is reported, and the 6 tx/s step was run at every delay instead.

Table 10 adds organizations. Each transaction then needs a majority of 8 or 12 endorsements, five or seven, instead of three. With four machines, each desktop also hosts two or three peers, and every peer validates and commits every block. Sequential latency rose moderately, from 412 to 678 and 749 ms. Throughput fell sharply: the 8- and 12-organization channels sustained 2 tx/s and peaked at 3.3–3.7 tx/s. The experiment cannot separate the cost of more endorsements from that of more peers per machine, and a consortium of 12 organizations on separate machines would show the first alone. In the 4-organization run at $W = 20$, 409 of 1500 transactions were rejected for mismatched endorsements, the failure mode the soak explains.

Table 10. Organization counts on the four-desktop testbed, revised chaincode, MAJORITY endorsement. The 8- and 12-organization channels share the four desktops, the three Raft consenters and the ordering parameters of the 4-organization channel. Closed loop: 200 transactions with $W = 1$ and 1500 with $W = 20$; open loop: ascending 150 s steps, sustained by the criterion of the text.

Organizations	Endorsements Required	Peers per Desktop	$W = 1$ Latency (ms)	$W = 20$ tx/s	$W = 20$ Median (ms)	$W = 20$ Failures	Highest Sustained tx/s	Peak Committed tx/s
4	3	1	412/634	13.0	1458	409	8	11.7
8	5	2	678/924	2.5	8231	0	2	3.7
12	7	3	749/926	1.8	11,630	0	2	3.3

At 6 tx/s, 60% of the open-loop capacity, the network ran for 12 h (Figure 5). Of 259,617 transactions offered, 256,309 committed (98.7%). Committed throughput was 5.7–6.0 tx/s in every hour, and median latency stayed between 729 and 847 ms with no

upward drift, while the 95th percentile varied between 1.7 and 3.0 s. The soak also revealed an effect that the short sweeps did not. Org2's peer, on D2, the desktop with four hardware threads, committed blocks more slowly than the channel produced them and fell behind at a steady 2162 blocks per hour, against about 10,678 blocks produced per hour. After 12 h it trailed by 26,222 blocks. Its container averaged 15% CPU, so processing was not the limit; storage on that machine is the likely one, but it was not measured. A peer that trails endorses against an older state, so its endorsement differs from the others'. The gateway rejected those transactions before ordering: 3125 (1.2%) failed this way, and 183 more when the gateway briefly found no set of peers satisfying the policy. None left a partial record. Every failed submission, 3308 of 3308, had neither the event nor the link on the ledger, and 2000 committed transactions sampled at random all had both, with the link naming the event's rating; the same check over the open-loop sweeps found 1000 sampled commits complete and all 27 failures absent. When the load stopped, Org2's peer caught up, and all four peers then reported the same height (191,785) and current block hash. On machines of unequal capacity, the rate a consortium can sustain for hours is therefore set by its slowest peer's commit rate, which a sweep of a few minutes does not reveal. A deployment needs peers sized for the block rate or a client that resubmits transactions rejected for mismatched endorsements.

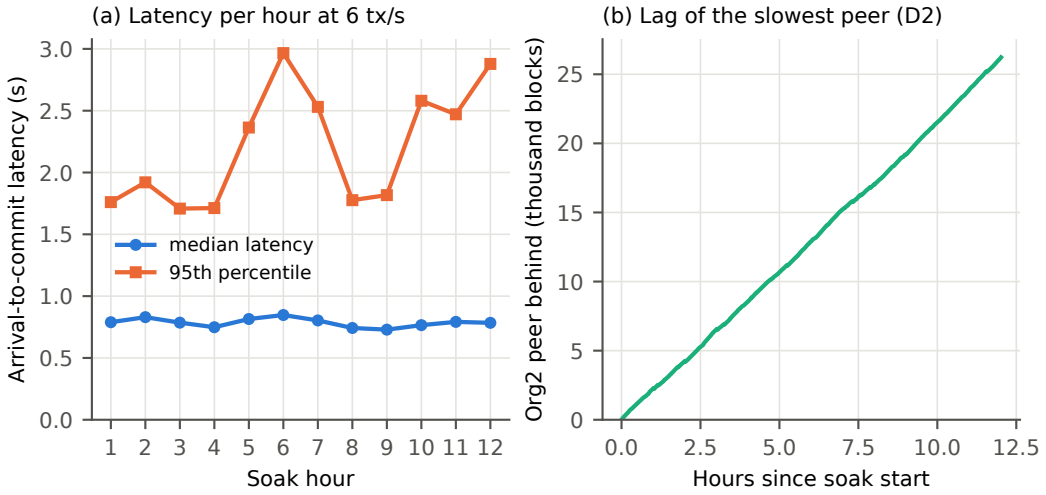

Figure 5. Twelve-hour soak at 6 tx/s on the four-desktop testbed. (**a**) Arrival-to-commit latency in each hour. (**b**) Blocks by which Org2's peer trailed the other peers, sampled every minute.

5.6. Lifecycle Assertion Enforcement

The measurements above exercise only the genesis event, so they do not show whether the lifecycle state machine of Section 3.4.3 is enforced. On the direct provenance path it is, by construction, since each lifecycle function asserts its expected predecessor stage. The test below checks that the bridge path enforces it too; otherwise an out-of-order event could enter the ledger carrying a reputation consequence.

Table 11 reports a direct test, repeated with the revised chaincode; the submitted chaincode gave the same counts in the original run and in a repeat on the current ledger. Three constrained transitions were submitted through `IntegrationContract:RecordProvenanceWithReputation` against assets deliberately in the wrong stage, 100 submissions each: `PRINT_JOB` against an asset that does not exist, and `INSPECTION` and `CERTIFICATION` against assets standing at `MATERIAL_CERTIFIED`. Each submission used a rated actor unique to itself, so its reputation is half independently observable.

All 300 were rejected, every one during endorsement, with the chaincode returning `invalid lifecycle transition` and naming both the required and the actual stage. Be-

cause the rejection occurs at endorsement, no transaction was ever assembled or sent to the ordering service. A subsequent ledger walk over all 300 confirmed that nothing was written: no provenance event of the rejected type, no reputation accumulator incremented above zero, and no link entry, in any trial. The assets targeted by the `INSPECTION` and `CERTIFICATION` attempts remained at `MATERIAL_CERTIFIED` with a single event in their history, and the assets targeted by `PRINT_JOB` were never created.

A rejection result is only meaningful against a control, since a chaincode that rejected everything would produce the same table. One hundred in-order operations were therefore run on fresh assets, each creating the asset with `MATERIAL_CERTIFICATION` and then advancing it with `PRINT_JOB`. All 100 committed both events, ending at `PRINT_COMPLETE` with two events in history. The assertion rejects the out-of-order case and admits the in-order case on the same path in the same deployment.

The link column now uses a corrected check. In the submitted version the check compared a field that the returned rating records do not carry, so it could not have found a link. It now reads the rating's actor field. A read-only re-walk of the original run with the corrected check found no link naming a rejected trial's actor while seeing all 200 genesis links of the `INSPECTION` and `CERTIFICATION` conditions, so the check does detect a link where one exists.

A second test probes a bypass the original test did not cover. An event string outside the constrained set sets the asset stage to the string itself, so in the submitted chaincode an unconstrained event named `INSPECTION_PASSED` moved an asset into that stage, after which `CERTIFICATION` passed its predecessor check. On 100 assets at `MATERIAL_CERTIFIED`, the submitted chaincode accepted the stage-named event and the certification on 100 of 100 and certified 100 parts that were never inspected. The revised chaincode rejected the stage-named event at endorsement on 100 of 100, and none of those assets left `MATERIAL_CERTIFIED`.

Table 11. Lifecycle assertion enforcement on the bridge path, single-host testbed, revised chaincode (v1.1). Each row is 100 submissions of an event against an asset in the wrong stage, made through `IntegrationContract:RecordProvenanceWithReputation`. The three ledger columns are counted by a read-only walk after the fact.

Event	Requires Stage	Actual Stage	*N*	Rej.	Prov.	Acc.	Link
`PRINT_JOB`	`MATERIAL_CERTIFIED`	asset absent	100	100	0	0	0
`INSPECTION`	`PRINT_COMPLETE`	`MATERIAL_CERTIFIED`	100	100	0	0	0
`CERTIFICATION`	`INSPECTION_PASSED`	`MATERIAL_CERTIFIED`	100	100	0	0	0
Control, in order			100	0	100/100 committed		

Rej. counts submissions rejected; Prov., Acc. and Link count trials in which a provenance event of the rejected type, a reputation accumulator above zero, or a link entry naming the trial's rated actor was found on the ledger afterwards. Every rejection carried the error class `LIFECYCLE_ASSERTION` at the endorsement phase, so no rejected submission reached the ordering service. The control creates each asset with `MATERIAL_CERTIFICATION` and then advances it with `PRINT_JOB` on the same path.

5.7. Atomicity Validation: Unified vs. Two-Chaincode

The preceding results characterize the unified system in isolation. All fault-injection results in this section were obtained on the single-host two-organization testbed. The phase boundaries against which the fault model is calibrated, and therefore the partial-write window reported below, are properties of that deployment; the corresponding interval on the distributed testbed, where a bridge write commits in roughly 412 ms, was not measured. The consistency invariant follows from Fabric's read–write set commit semantics and does not depend on testbed scale, but its empirical demonstration here does. We

now test the paper's central claim directly: that co-locating provenance and reputation in a single Fabric read–write set makes the consistency invariant of Section 3.1 structurally enforceable (under the trust assumptions of Section 3.2), whereas the natural alternative, two independent chaincodes orchestrated by the client at the application layer, cannot guarantee it. For an empirical comparison, we deployed the two-chaincode design as two separate Fabric chaincodes (`prov` and `rep`) with independent packages, CCAAS containers, and ledger-state namespaces, so that a logical provenance+reputation operation spans two independent transactions and two independent read–write sets. The unified design (`IntegrationContract:RecordProvenanceWithReputation`) is unchanged. Both designs were driven by an identical workload on the single-host testbed; the only variable is the architecture. The control is the client-orchestrated arrangement, the one that independently developed provenance and reputation services naturally take. The same-channel `InvokeChaincode` arrangement, atomic under Fabric's semantics, was built from the revised source and fault-injected with the same method; its rows appear with the other fault conditions in Section 5.7.3 and its cost in Section 5.8. We follow a pre-registered condition matrix, with fault timing fixed by definition and never tuned to alter a divergence rate, and report every condition as measured.

5.7.1. Fault Model and Verification

A logical operation runs in a child process that a controller terminates with `SIGKILL`. The child writes a progress marker at each of the four phase boundaries of a Fabric submission (the operation starts, endorsement completes, the ordering service accepts the transaction, and the commit event resolves) and `fsync`s it before continuing, so the point a kill landed is recoverable from disk. The marker mechanics are detailed in the Supplementary Materials.

Kill timing is defined against those markers rather than wall-clock offsets, because a fault model that cannot say which phase it interrupted cannot support a claim about atomicity. The *pre-endorsement* policy kills on the start marker; the *post-endorsement* policy kills after endorsement but before submission; the *post-submission* policy kills once the ordering service has accepted the transaction; and the *random* policy arms on the start marker and kills after a delay drawn uniformly from zero to the 95th percentile of the measured transaction lifetime, so that kills cover the whole operation.

The lifetime the random policy is calibrated against was measured first, in a fault-free calibration condition of 200 operations on the same deployment (reported with the fault conditions in Section 5.7.3). From the start marker, endorsement completed at a mean of 190.7 ms, submission returned at 208.2 ms, and the commit resolved at 242.3 ms, with a 95th percentile of 285 ms for the complete operation. The random policy therefore draws from $U[0, 285]$ ms. Figure 6 places those markers on a single time axis.

That calibration also bounds the interval in which a partial write could exist at all. A transaction that has not yet been accepted by the ordering service cannot have written anything, and one whose commit has resolved has written everything; the only interval in which the question is open is between those two markers. Measured over the 200 calibration operations, it is 34.1 ms on average (median 34 ms, 95th percentile 39 ms, range 27–54 ms), or about 14% of the 242.3 ms operation. This is the window the post-submission policy targets directly, and it is the window in which the two-chaincode design is vulnerable for the whole of its second transaction.

Verification is an independent, key-based ledger walk performed after the trial loop, reading only through the chaincodes' read-only query interfaces and never from a marker. For the unified design three keys are read: the provenance event, the reputation accumulator for that operation's rated actor, and the link entry binding them. All three present is a

complete operation, none present is a clean abort, and any proper subset is a divergence. For the two-chaincode design, the corresponding keys are the provenance asset on `prov` and the reputation accumulator on `rep`. Each operation uses a rated actor unique to itself, so the reputation half of every trial is observable independently of every other trial.

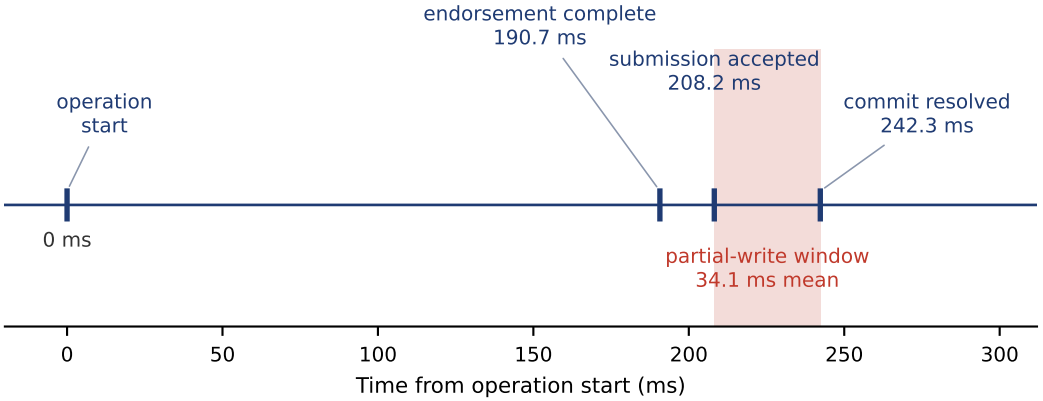

Figure 6. Measured transaction timeline from the fault-free calibration condition, 200 operations on the single-host testbed. Markers are the mean offsets from the operation start at which endorsement completed, the ordering service accepted the submission, and the commit resolved. The shaded interval between submission accepted and commit resolved is the window in which a partial write could exist; it is 34.1 ms on average. Every value is taken from the per-trial progress markers written during the calibration run.

Accumulator presence is tested on the count of ratings recorded against the actor rather than on the value of the Beta parameters, because both chaincodes synthesise a default record for an actor that has never been rated; the Supplementary Materials give the predicate and its validation.

The condition matrix fixes the trial budget before execution: 500 operations for each of the four kill policies, 300 for each of the three event and bridge-rule types, and 200 for the fault-free calibration. No condition was extended, truncated, or re-run after its outcome distribution was observed.

5.7.2. Baseline Cost (No Faults)

Table 12 compares both designs under fault-free load (500 logical operations sequentially and 500 at twenty in flight, with a rated actor distinct to each operation so that the architectural difference is not confounded with hot-key contention). The unified design is 1.81× faster in mean sequential latency and 1.90× higher in concurrent throughput, a direct consequence of committing one read–write set instead of two. The atomicity guarantee therefore imposes no performance penalty in this comparison: it improves performance by halving the ordering and commit rounds.

These baselines were taken with a benchmark client that opens one gateway connection and reuses it for every operation. They do not carry the fault-injection harness's per-operation child process, per-operation gateway, or `fsync`'d markers, because that harness was not used to produce them. They are therefore directly comparable to Table 6, and the comparison is informative. The unified logical operation is a bridge write, so its 42.7 ms sequential mean is the figure to set against the 49.5 ms bridge write of Table 6, both being one Fabric transaction of the same kind on the same deployment. The 6.8 ms gap runs in the direction of the baseline being the faster of the two, which harness overhead cannot explain. The two were measured in separate sessions at different ledger sizes, and single-host latency is state-dependent (Section 5.2), so the gap is most plausibly a state effect rather than an architectural one; it was not isolated, and the agreement should be read as approximate

rather than as a repeat measurement. Two further differences are not harness artifacts. A two-chaincode logical operation is two transactions, so its latency is close to twice a single write by construction; and the throughput column is measured at twenty transactions in flight, whereas Table 7 uses one hundred.

Table 12. Baseline cost (no faults), single-host testbed: submitted unified chaincode (v1.0) against the two-chaincode deployment. 500 logical operations sequentially and 500 at $W = 20$; a rated actor distinct to each operation. Measured with a single-gateway benchmark client, not the fault-injection harness.

Metric	Two-CC	Unified
Transactions per logical op	2	**1**
Seq. latency mean	77.3 ms	**42.7 ms**
Seq. latency P50	74 ms	**40 ms**
Seq. latency P95	94 ms	**58 ms**
Throughput ($W = 20$)	159.8 TPS	**304.0 TPS**
MVCC conflict rate	0/500	0/500
Completed	498/500	**500/500**

Two-CC = two-chaincode deployment; Seq. = sequential. Bold marks the better value of the two designs. The two incomplete two-chaincode operations failed at the gateway, not at validation; neither was an MVCC conflict.

5.7.3. Divergence Under Fault Injection

Table 13 reports every condition as an outcome distribution, not a bare divergence count, because the distribution shows whether the fault model reached the interval it targeted. Three results carry the argument.

The calibration condition, with no fault injected, resolved to all three keys present in 200 of 200 operations. This is the positive control: the verification predicate sees a complete operation when nothing interferes with it.

The three deterministic policies partition the transaction exactly as the phase markers predict. Killing before endorsement leaves nothing on the ledger in 500 of 500 operations. Killing after endorsement but before submission also leaves nothing, again in 500 of 500, because an endorsed read–write set that is never submitted writes nothing. Killing after the ordering service has accepted the transaction produces all three keys in 500 of 500. This last condition is the decisive one: the client process is destroyed inside the 34.1 ms window of Section 5.7.1, and the write still lands complete, because once the transaction is ordered its commit is Fabric's responsibility and no longer the client's.

The randomly timed conditions produce a mixture, as expected when kills are spread over the whole operation: 417 clean aborts against 83 complete operations under the general random policy, and similar mixtures across the three event and bridge-rule types. In none of the eight unified conditions did any operation resolve to a proper subset of the three keys.

The two-chaincode control diverges in 500 of 500 operations under a kill placed in its inter-write interval, and it does so twice: once before the unified conditions were run and once after they had all completed (Section 5.7.4).

The revised implementation and the `InvokeChaincode` arrangement were measured with the same fault model, each calibrated on its own deployment. The partial-write window measured 33.7 ms on average for the revised unified binary and 36.0 ms for `InvokeChaincode`, against 34.1 ms in the submitted calibration. Killing each client after the ordering service had accepted its transaction left all three keys in 500 of 500 and 500 of 500 operations, and random kills produced only complete operations and clean aborts: 0 divergences in 1000 fault-injected v1.1 unified operations and 0 in 1000 `InvokeChaincode` operations.

Across the seven fault-injected unified conditions of the submitted implementation (v1.0), which exclude the fault-free calibration, 2900 operations produced no divergence: a rule-of-three 95% upper bound of 0.103% on the divergence rate and a Wilson 95% upper bound of 0.132%. This is wider than the 0.0435% of the earlier version of this work, which rested on more trials whose kills were never confirmed to reach the interval where a divergence could occur; a wider bound around a verified quantity is the stronger result.

Table 13. Outcome distribution under fault injection, single-host testbed. Unified conditions report all three ledger keys present (ALL), none present (NONE), or a proper subset (SPLIT); a SPLIT is a divergence. Kill timing is defined against the phase markers of Section 5.7.1. The two-chaincode control uses the same ledger-only rule over its two keys. The v1.1 arrangements were each calibrated afresh, and their random kills drawn up to their own P95 lifetime; for `InvokeChaincode` the three keys span its two namespaces. Totals count fault-injected operations and exclude the fault-free calibration runs. The 3900 unified trials of the abstract are the v1.0 total (2900) and the v1.1 total (1000); the two-chaincode control is two batches of 500, run before and after the v1.0 unified conditions.

Condition	N	Killed	NONE	ALL	SPLIT
Unified design, submitted implementation (v1.0)					
Calibration, no fault	200	0	0	200	**0**
Kill pre-endorsement	500	500	500	0	**0**
Kill post-endorsement	500	500	500	0	**0**
Kill post-submission	500	500	0	500	**0**
Kill random $U[0, 285]$ ms	500	484	417	83	**0**
Unified design, by event and bridge rule (random timing)					
Material certification	300	289	247	53	**0**
Print completion	300	284	233	67	**0**
Delivery	300	297	268	32	**0**
Fault-injected total, v1.0	2900	2854	2165	735	**0**
Two-chaincode control, kill in the inter-write interval					
Before unified conditions	500	500	0	0	**500**
After unified conditions	500	500	0	0	**500**
Total, two batches	1000	1000	0	0	**1000**
Unified design, revised implementation (v1.1)					
Calibration, no fault	200	0	0	200	**0**
Kill post-submission	500	500	0	500	**0**
Kill random $U[0, 189]$ ms	500	449	308	192	**0**
Fault-injected total, v1.1	1000	949	308	692	**0**
`InvokeChaincode` arrangement, revised implementation (v1.1)					
Calibration, no fault	200	0	0	200	**0**
Kill post-submission	500	500	0	500	**0**
Kill random $U[0, 188]$ ms	500	467	324	176	**0**
Fault-injected total	1000	967	324	676	**0**

Italics mark the group headings; bold marks the SPLIT column, which counts divergences, and the labels of the total rows. Killed counts are below N in the randomly timed conditions because a delay drawn near the upper end of the distribution can exceed the operation's own lifetime, in which case the operation completes before the kill fires. Those operations are retained and counted, not discarded. The unified fault-injected total excludes the 200 calibration operations, which carry no fault; including them gives 3100 operations and the same zero. The two-chaincode arm has no NONE or ALL column entries because the window kill lands in its inter-write interval by construction, so every operation reaches the divergent state rather than aborting cleanly or completing. Of the three event-type conditions, only material certification is sequence-constrained; the print-completion and delivery conditions exercised the unconstrained recording path. The atomicity result does not depend on the sequencing constraint, so the conditions remain comparable.

5.7.4. Interpretation

In the session of the submitted implementation, the two-chaincode control diverged in all 1000 of its operations, two batches of 500, and the unified design in none of its 2900 fault-injected operations; the revised implementation (v1.1), measured in a later session with its own calibration, added 1000 operations without divergence, which gives the 3900 of the abstract. Two properties of the method make these results directly comparable. Both arms are judged by the same rule, the per-trial ledger walk of Section 5.7.1, and never by what a client recorded before it died. And the two-chaincode control brackets the unified conditions: run once before and once after them, on the same deployment and binaries without a restart, it diverged in 500 of 500 operations both times, so the deployment did not drift during the session.

How often a randomly timed fault produces a divergence depends on the fault model, but on an append-only ledger a single divergence is permanent. In the unified design the divergent state cannot occur, because provenance and reputation occupy one read–write set; in the two-chaincode design it can, however rare the triggering fault. The unified design is also the faster of the two (Table 12).

5.8. Comparison of the Atomic Arrangements

Section 4.1 lists the three ways Fabric can arrange a provenance-plus-reputation operation. The submitted version measured co-deployment against the client-orchestrated arrangement only. To measure the same-channel `InvokeChaincode` arrangement under identical business logic, the revised chaincode source was compiled three ways from the same files: as the unified binary, and, selected by build tags, as a provenance chaincode holding assets, events and links and a reputation chaincode holding configuration, stakes, ratings, accumulators, gates and the actor registry. The provenance chaincode's bridge runs the same provenance and link code as the unified binary and obtains the reputation half through an `InvokeChaincode` call to the reputation chaincode, whose bridge function runs the unified binary's rating code and refuses any call whose top-level proposal does not target the provenance chaincode; without that check a client could create a rating with no provenance event. Both arrangements commit one read–write set per operation, spanning one namespace in the unified case and two in the other. The arms were interleaved in one session: each of three rounds ran every arm once, with 60 s between runs, so all arms saw the same ledger state.

Table 14 reports the result. At $W = 1$ the two atomic arrangements are not distinguishable in end-to-end latency: the median P50 was 75.1 ms for `InvokeChaincode` and 75.9 ms for the unified binary, with overlapping run-to-run ranges. The phase split shows where the second chaincode costs time: endorsement took 4.5 ms longer for `InvokeChaincode` (32.6 against 28.1 ms), the cost of the chaincode-to-chaincode call through the peer, while ordering and commit were unchanged within noise. Under load the difference becomes measurable. Steady throughput was 8.1% lower for `InvokeChaincode` at $W = 20$ (215.53 against 234.59 TPS, ranges overlapping) and 11.7% lower at $W = 100$ (282.39 against 319.85 TPS, ranges not overlapping). The client-orchestrated arrangement, which commits two transactions per operation, was measured in Table 12 at 1.81 times the unified latency. Enforcing the mapping policy, measured at $W = 1$, did not measurably change the unified binary's latency: median P50 74.2 against 75.9 ms and endorsement 32.3 against 28.1 ms, with overlapping run-to-run ranges.

The same session measured the revised chaincode against the submitted one in their two namespaces of one ledger. The revision changes the values written, not which keys are read or written, apart from one gate read the bridge no longer performs. The fault-injection verdicts and the lifecycle-assertion counts depend only on key presence and on the count of ratings, which the update increments by one regardless of the outcome value, so they

are unaffected by construction. Measured cost changed little: the bridge's P50 at $W = 1$ was 75.9 ms against 72.0 ms for v1.0, a 5.4% increase whose run ranges do not overlap, and steady throughput at $W = 100$ was 319.85 against 315.20 TPS, within the run-to-run range. Absolute latencies in this session are higher than in Table 6 for the same v1.0 binary (72.0 against 50.1 ms), because single-host performance depends on ledger state (Section 5.2) and the ledger had grown since; comparisons are therefore drawn only within the session. Enforcing the mapping policy added 4.2 ms to endorsement, for the registry and rule reads, and no measurable end-to-end latency (P50 74.2 against 75.9 ms).

Table 14. Atomic arrangements and chaincode versions measured in one interleaved session on the single-host testbed (bridge writes; $W = 1$: 500 per run, $W = 20$ and $W = 100$: 2000 per run; three runs each). Latencies in ms; TPS counts committed transactions in the steady window.

Arrangement	P50	Mean	P95	Endorse P50	Order + Commit P50	TPS $W = 20$	TPS $W = 100$
Unified, v1.0 (submitted)	72.0	71.7	84.0	30.3	41.8	–	315.2
Unified, v1.1	75.9	75.5	94.8	28.1	43.9	234.6	319.9
Unified, v1.1, mapping enforced	74.2	75.3	88.4	32.3	42.1	–	–
InvokeChaincode, v1.1	75.1	75.3	90.1	32.6	42.6	215.5	282.4

Medians across three runs. Ranges across runs: P50 at $W = 1$ 70.3–72.3 (v1.0), 75.6–77.7 (v1.1), 70.0–75.7 (enforced), 65.8–78.2 (InvokeChaincode) ms; steady TPS at $W = 20$ 219.1–236.2 (v1.1) and 191.7–246.0 (InvokeChaincode); at $W = 100$ 299.5–340.4 (v1.0), 312.2–351.4 (v1.1) and 277.4–298.2 (InvokeChaincode). Every one of the 36,000 submitted transactions committed. The enforced condition rates a pool of 100 registered suppliers; the others rate a fresh unregistered label per transaction with enforcement off.

Both atomic arrangements also held under fault injection. With the fault model of Section 5.7.1, calibrated afresh for each, the revised unified binary produced 0 divergences in 1000 operations and `InvokeChaincode` 0 in 1000 (Table 13); the partial-write window was 33.7 and 36.0 ms on average respectively. The two arrangements therefore differ in cost and in maintenance, not in correctness.

Deployment and maintenance cost is a matter of counting operations, not a measurement, and Table 15 counts them. Co-deployment halves the lifecycle operations of an initial deployment and of any change that crosses the interface between the two halves. For the common case of a change to one contract that keeps the interface, the two arrangements cost the same number of operations, but co-deployment redeploys, and should retest, all three contracts, whereas `InvokeChaincode` redeploys only the changed one and allows a separate endorsement policy per contract. The submitted version claimed a lower operational cost for co-deployment without this accounting; the revision replaces that claim with the trade-off above.

Table 15. Chaincode lifecycle operations for N organizations under Chaincode-as-a-Service. The single-host deployments of this section ($N = 2$) used exactly these counts.

Operation	Unified	InvokeChaincode
Initial deployment: packages, installs, approvals, commits	$1 + N + N + 1$	$2(1 + N + N + 1)$
Initial deployment: chaincode containers	N	$2N$
Upgrade of the reputation logic, interface unchanged	$2N + 2$ (all three contracts)	$2N + 2$ (reputation chaincode only)
Upgrade that changes the cross-contract interface	$2N + 2$	$4N + 4$, coordinated
Separate endorsement policy per contract	not possible	possible

5.9. Evaluation of the Reputation Mechanism

The measurements above concern recording; this section evaluates the reputation mechanism the recording feeds, all with the revised chaincode on the single-host testbed: the gate, the mapping policy and identity guards, a complete lifecycle with failures, recovery and disputes, contention on one actor, and an offline simulation of the model.

5.9.1. Gate Acceptance and Refusal

A gate was set on the authorization event `PRINT_JOB_START` for the quality dimension, and four manufacturers were given ledger histories, through counterparty outcomes under the enforced mapping policy, that fail no criterion or exactly one. The interval case uses a rater whose weight had fallen from 0.767 to 0.252 after ten of its ratings were overturned on dispute, which is the situation the interval condition is meant for: eight successes from such a rater give a passing score and event count but little evidence. Table 16 gives the state each manufacturer presented and the decision. The ledger walk confirms that each refusal left the asset's history unchanged and that the one authorization appended a single event without moving the asset's lifecycle stage.

Table 16. Gate acceptance and refusal on the ledger, single-host testbed, revised chaincode (v1.1). Gate on `PRINT_JOB_START`, quality: *minScore* 0.6, *minEvents* 6, maximum credible-interval width 0.62. CrI = credible interval. Ratings are counterparty outcomes (success/failure); score, events and interval width are those the gate evaluated; history counts the asset's events before and after the request.

Manufacturer	Ratings	Score	Events	CrI Width	Fails	Decision	History
pass	16/0	0.877	16	0.298	none	authorized	1→2
lowscore	12/12	0.500	24	0.401	score	refused	1→1
few	4/0	0.717	4	0.594	events	refused	1→1
wideci	8/0, low-weight rater	0.668	8	0.663	interval	refused	1→1

Two further checks confirm that authorization is separated from outcome recording. A failed inspection of the refused low-score manufacturer was recorded, moving its part to `INSPECTION_FAILED` and its β from 11.207 to 11.974; and `SetReputationGate` refused a gate on the outcome event `INSPECTION`, while the bridge refused an authorization event submitted as an outcome.

5.9.2. Mapping Policy and Identity

Table 17 runs each way an organization might rate itself, rate an unregistered or mismatched actor, or act as another organization, against the submitted and the revised chaincode. The submitted chaincode accepted every attempt it could express; the revised chaincode, with the mapping policy enforced, rejected each at endorsement, so none left anything on the ledger, and accepted the control in which the counterparty holding the inspector role submits the same outcome. Case S1 is the identity defect of Section 3.5.3: the default client identities of the two organizations share the common name `user1`, and the submitted chaincode let Org2's identity rate with Org1's stake.

The cases of Table 17 test rejection. A separate test on the single-host testbed exercised every rule of the reference policy (Table 4) under enforcement: for each event type, one passing and one failing outcome, each on its own part taken first through the preceding lifecycle stages, submitted through the bridge by an organization holding a submitter role. All 12 committed with the event, its link and the rating on the ledger, and the rated actor's accumulator grew by the recorded weight $w = 0.767$ in α for a pass and in β for a failure,

the other parameter unchanged and the rating count up by one (to within 10^{-4}, the decay between the two reads; Supplementary Section S5).

Table 17. Identity and mapping-policy tests on the ledger, single-host testbed, submitted (v1.0) and revised (v1.1) chaincode. v1.1 ran with the mapping policy enforced, except S1, which isolates the identity key with enforcement off. Org1's and Org2's client identities both carry the certificate common name `user1`.

Case	Attempt	v1.0	v1.1
S1	Org2 identity carrying the common name of a staked Org1 identity submits a rating without staking	accepted	rejected (no stake under its own key)
S2	Rated actor's own organization submits a rating-producing event about it	accepted	rejected (same organization)
S3	Control: counterparty organization holding the inspector role submits the same event	–	accepted
S4	Rating names a label that no organization registered	accepted	rejected (label not registered)
S5	`MATERIAL_CERTIFICATION` rates a logistics actor	accepted	rejected (rated role mismatch)
S6	`MATERIAL_CERTIFICATION` rates the delivery dimension	accepted	rejected (dimension not in rule)
S7	Org2 administrator re-binds a label registered by Org1	no registry	rejected (label bound to Org1)
S8	Client identity registers an actor label	no registry	rejected (administrator required)
S9	Direct rating by the rated actor's own organization	accepted	rejected (same organization)
S10	Arbitrary caller zeroes a stake record	accepted	rejected (administrator required)
S11	Administrator sets a gate on an outcome event	accepted	rejected (not an authorization type)

5.9.3. Lifecycle Scenario

A scripted scenario takes parts through the full lifecycle on the ledger under the enforced reference policy. Org1 holds the manufacturer, customer and regulator roles and Org2 the supplier, inspector and logistics roles, so every rating-producing event is submitted by the counterparty organization. Each part is certified (rating the supplier), authorized for printing through a gate on the manufacturer's quality score (*minScore* 0.45), printed and inspected (each rating the manufacturer), and, if it passes, certified and delivered (rating the supplier and the carrier). Decay time is compressed so that one day of $\lambda = 0.98$ decay elapses every 6 s; nothing else in the model is changed. Figure 7 traces the manufacturer's score through 16 parts, of which 8 passed inspection, 6 failed and one delivery was late.

In the degraded phase each failed build and inspection added to β until, at part P10, the gate refused the print authorization; the score had fallen to 0.421. Parts in this phase were printed in pairs before inspection, so the inspection of part P09, printed while the manufacturer was still admitted, arrived after its score had fallen below the gate; it was recorded with its rating, which is the property the revision restores. During inactivity, evaluation-time decay drew the score toward the prior mean of 0.5, and the gate admitted the manufacturer again after 399 s, 66.6 compressed days. The recovery lasted only until the manufacturer's next rating. Decay is applied when a score is evaluated, from the stored accumulators and the time of the last rating, and a new rating adds to the undecayed accumulators and resets that time. The print completion of part P11, the first after the pause, therefore returned the score below the gate, where it stood when the part's inspection was recorded. This is the on-ledger instance of the behaviour the simulation quantifies

(Section 5.9.5): evaluation-time decay discounts inactivity, not old evidence. Passing parts then raised the score to 0.560.

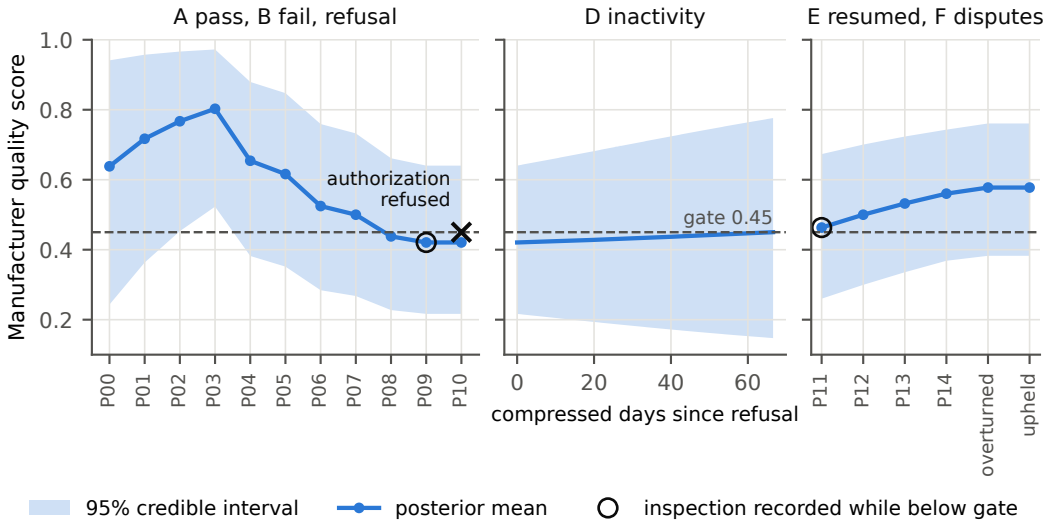

Figure 7. Lifecycle scenario on the ledger (single-host testbed, revised chaincode): the manufacturer's quality score (posterior mean and 95% credible interval) against the gate on print authorization. (**Left**): parts pass, then fail, printed in pairs before inspection, until a print authorization is refused. (**Centre**): inactivity, with decay time compressed to 6 s per day. (**Right**): production resumes and two disputes are resolved. Ringed points mark inspections recorded while the manufacturer was below the gate. Dashed line: the gate threshold (*minScore* 0.45); cross: the refused print authorization.

Finally the manufacturer's organization disputed two failed inspection ratings, claiming the cause lay in a powder lot. The arbitrator overturned one, and its update was reversed: β fell from 11.207 to 10.440 and the rating count by one, while α was unchanged because a failure adds nothing to it. The inspector's stake was slashed from 20,000 to 18,000. It upheld the other, which left the rating and the stake unchanged. The two verdicts also entered the inspector's meta-reputation, one unit of β^m and one of α^m, and the weight recorded with its next rating was 0.806, against 0.767 before the disputes. The weight rose because the evidence factor of Equation (7) counts both verdicts while the meta-score stays at 0.5; the overturned verdict alone would have lowered it to 0.631. A ledger walk at the end found 61 outcome events, 61 link entries and 61 committed bridge submissions, one each, together with 15 authorization events and 1 refused request that left nothing on the ledger.

5.9.4. Contention on One Actor

Distinct-key workloads hide one case: many parts producing ratings of the same actor at once. Every rating of an actor in a dimension reads and writes one accumulator, so concurrent ratings of one actor conflict. The burst workload of Table 18 submits all parts of three manufacturers at once; every part's certification rates the single supplier, the hottest key, and its print completion rates its manufacturer. Without retry, 2 of 91 submissions committed and the rest were rejected for multi-version concurrency conflicts, leaving no partial state: the failed parts are absent. With a client that resubmits after a jittered, exponentially growing delay, every submission of three bursts committed, after at most 20 attempts and at most 8.2 s. A ledger walk over all 360 parts found every committed part with exactly its two events, two links and two distinct rating records (542 in all), no anomaly, and every accumulator's rating count equal to the ratings committed against it (271 for the supplier).

Table 18. Bursty hot-key workload, single-host testbed, revised chaincode (v1.1), mapping policy enforced. Each burst submits 30 parts for each of three manufacturers at once; every part's certification rates the single supplier and its print completion rates its manufacturer. Attempts and time to commit are per transaction (median/P95/maximum).

Burst	Retry	Parts	Committed/ Submitted	Attempts	MVCC Rejects	Endorse Mismatch	Time to Commit (ms)	Makespan (s)
0	no	90	2/91	1/1/1	89	0	265/265/265	0.4
1	yes	90	180/180	3/15/20	694	88	378/6133/8074	8.2
2	yes	90	180/180	2/14/17	712	71	322/6225/8195	8.3
3	yes	90	180/180	2/16/19	701	109	360/6397/8162	8.3

5.9.5. Simulation of the Reputation Model

The ledger experiments show that the mechanism behaves as specified. Whether its scores are accurate, and how the gate trades false acceptance against false rejection, depends on actors' true performance, which a ledger experiment cannot know. An offline simulation answers those questions with the chaincode's formulas (the two-sided update, weight 0.767 for a counterparty with no dispute history, the $(2, 2)$ prior, evaluation-time decay and the 95% credible interval) and binary outcomes drawn with a known success probability p. It compares them with the simple average of outcomes and an exponentially weighted moving average (EWMA, $a = 0.1$). Figures 8 and 9 report the results; the random seed is 20261009.

With one rating the Beta mean has the lowest error (0.243 against 0.299 for the simple average), because the prior keeps a single outcome from setting the score to 0 or 1. From five ratings the simple average is more accurate (0.132 against 0.160), because the prior, worth 5.2 ratings at the chaincode's weight, shrinks scores toward 0.5; by 100 ratings the two agree (0.032 and 0.030), while the EWMA settles at 0.069, the noise of its effective window. Averaged over the five values of p, the credible interval covers the true p in 95.2–96.9% of replications from 20 ratings on, slightly above nominal because a weight below one widens it, and in 75–92% below 20 ratings. The average hides a dependence on p, which an exact binomial calculation for the same interval shows (Supplementary Table S3). For p from 0.6 to 0.8, coverage is 94.2–99.0% at every count from 5 to 200 ratings, close to or above nominal. For high performers it is lower and varies with the count, as binomial coverage does [28]: at $p = 0.9$ it is 91.9–95.8% up to 100 ratings, and at $p = 0.95$ it is 59.9% at 10 ratings, 89.6% at 50 and 95.5% at 200. At $p = 0.95$ almost every interval that misses lies below the true value, because the prior pulls the interval toward 0.5. As a statement about the evidence the contract holds, the interval is reliable for actors with moderate success rates; for an actor with a near-perfect record it can lie below the true rate even after tens of ratings, an error toward refusal.

A gate on the Beta mean at 0.85 admitted at most 2.7% of unacceptable actors at any rating count, but refused every acceptable actor up to 10 ratings and 14.2% of them after 100. Lowering *minScore* to 0.80 reduced false rejection after 100 ratings to 1.9% at the cost of 11.4% false acceptance. The simple average at 0.85 admitted 17.0% of unacceptable actors after five ratings, the lucky-streak exposure that the prior removes. Adding the interval condition (width at most 0.30) changed false acceptance only early, at 10 ratings from 4.2% to 0.0%; at 20 ratings from 7.7% to 2.7%, by refusing more actors while evidence was thin. Gate thresholds therefore encode a choice between probation length and exposure that a consortium has to make explicitly. The gate errors measure decision quality, not score accuracy: a score accurate on average can still admit or refuse the wrong actors near the threshold. The thresholds are illustrative, not recommended values, and because outcomes are simulated as independent draws with a known p, the comparisons do not show which

model performs better on real manufacturing evidence, whose outcomes are correlated and whose p is unknown.

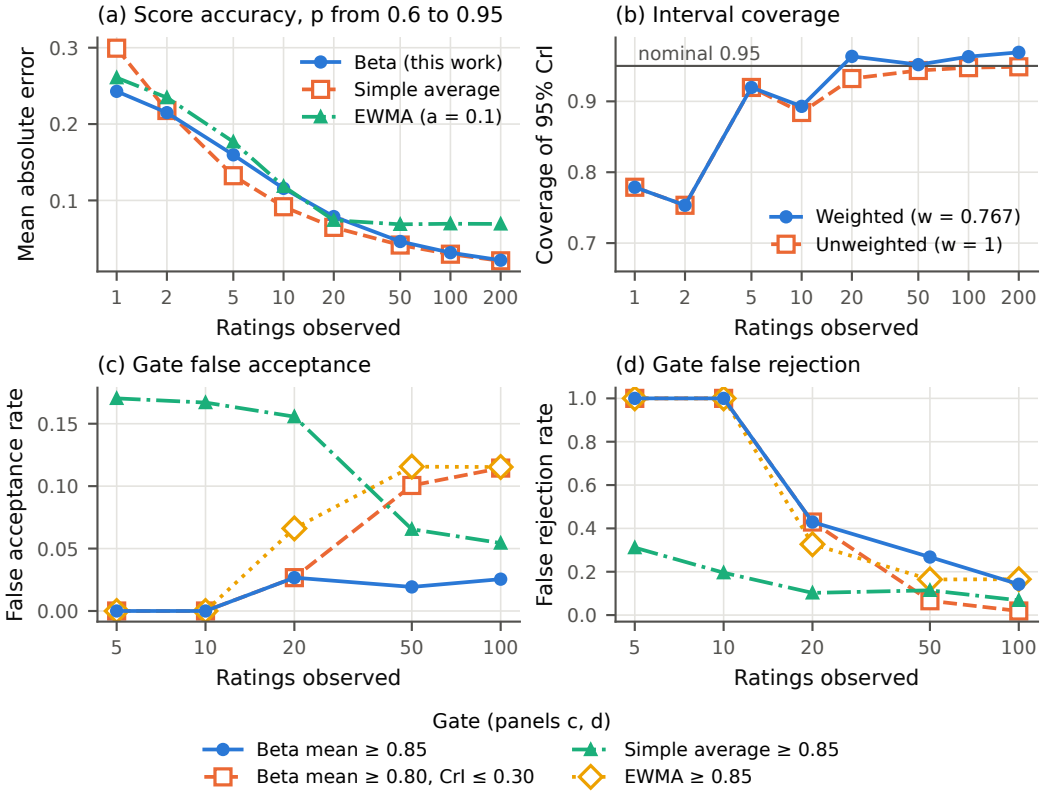

Figure 8. Simulated reputation model. (**a**) Mean absolute error of the score against the true success probability, 4000 replications for each of five values of p from 0.6 to 0.95. (**b**) Coverage of the 95% credible interval, averaged over the five values of p, at the chaincode's weight and at unit weight; Supplementary Table S3 gives exact coverage for each p. (**c**,**d**) Gate errors for 20,000 actors with p uniform on $[0.5, 1]$, acceptable when $p \geq 0.85$: false acceptance is the share of unacceptable actors admitted, false rejection the share of acceptable actors refused. CrI = credible interval; EWMA = exponentially weighted moving average.

During inactivity, $\lambda = 0.95$, 0.98 and 0.99 give half-lives of 13.5, 34.3 and 69.0 days, and an actor with 50 successes loses eligibility at *minScore* 0.80 after 37, 92 and 185 inactive days. While an actor stays active, however, evaluation-time decay discounts nothing, because every write resets the elapsed time. After p dropped from 0.95 to 0.60 at one event per day, the gate refused a median 137–139 days later for every λ, with 6.0–6.3% of actors still admitted after 200 days, little faster than the simple average (146 days). Applying decay on the write path would detect the drop in 19 days at $\lambda = 0.98$ with 0.25% false refusals before it, or in 4 days at $\lambda = 0.95$ at the cost of 49.4% false refusals; the EWMA is as fast but falsely refuses 32.7% of actors that had not changed. Section 6.6 discusses this trade-off.

Finally, manipulation. With the gate at *minScore* 0.80, interval width at most 0.30 and at least 30 events, and unacceptable actors (p uniform on $[0.6, 0.8]$) after 60 true outcomes, the gate admitted 5.8% without manipulation. A colluding counterparty that reported failures as successes at rate 0.3 raised admission to 32.5%, and at rate 0.5 to 69.9%; omitting failures at the same rates raised it to 23.5% and 49.7%. Omission is less effective than false reporting at equal rates because each omitted failure also removes evidence and widens the interval. Neither attack is visible to the ledger. What the enforced mapping policy changes is who must collude: under it, both attacks require a counterparty organization.

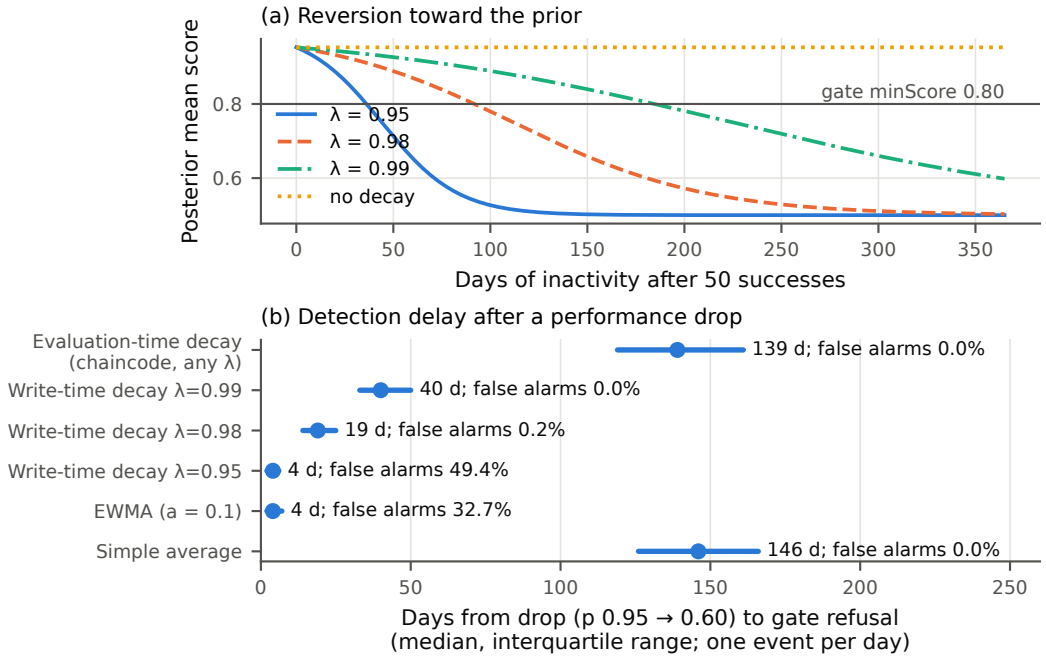

Figure 9. Decay sensitivity. (**a**) Score of an actor with 50 successes during inactivity, for $\lambda \in \{0.95, 0.98, 0.99\}$/day and no decay. (**b**) Days from a drop in success probability from 0.95 to 0.6 to the first gate refusal at *minScore* 0.80, one event per day, 2000 replications; false alarms are refusals during days 100–199, before the drop. Write-time decay is a counterfactual not implemented in the chaincode.

6. Discussion

6.1. Latency in the Context of AM Supply Chains

The bridge write latencies of Table 6 should be read against the timescale of the process they record: AM lifecycle steps take hours to days, so the commit latency of either testbed is negligible at that timescale. For decision timing, an authorization is one transaction, so a decision to release a build job is available within one write latency, and a part's recorded history with the current score of every actor that touched it within one read latency (Table 6). For workload capacity, a bureau processing 50 parts per day at several rating-producing events per part submits well under one event per minute, while one supplier's accumulator absorbed 10.9 contended ratings per second (Section 5.9.4). For service requirements, the slowest rating in that contended workload committed 8.2 s after its first submission, which bounds how far a gate decision can lag the latest submitted outcome under contention.

6.2. Atomicity as a Correctness Guarantee

By Fabric's execute-order-validate protocol, a transaction commits only if its read–write set passes validation, so one read–write set holding the provenance event, the accumulator update and the `PROV_REP_LINK` entry commits all three or none. A client-orchestrated two-chaincode design splits the operation into two independently validated transactions and leaves a window between them; same-channel `InvokeChaincode` closes it, because the callee's writes join the caller's read–write set. Sections 5.7 and 5.8 measure both, and the choice between the two atomic arrangements is a trade-off of cost and maintenance, not of correctness.

The guarantee holds at commit and therefore covers the whole execution path: whatever fails, the transaction commits completely or not at all. A failure that stops the transaction before it reaches the ordering service, whether of the client, an endorsing peer or the chaincode container, leaves it absent, as the pre- and post-endorsement kill conditions of Table 13

confirm. Once the ordering service has accepted it, the transaction can still commit after the client has crashed and before the client learns the outcome: in the post-submission conditions, every killed operation committed with all three keys present. A client that crashed therefore cannot tell from its own records whether its operation was recorded, and must check the ledger before resubmitting it (Section 6.3). The claim is scoped to the trust assumptions of Section 3.2: Raft ordering without a Byzantine orderer and the 3-of-4 endorsement threshold.

A two-chaincode client could retry the second write or issue a compensating transaction, and under faults that leave it running, such as a timeout, a rejected endorsement or a transient orderer failure, this would lower the observed divergence rate. It does not change what the comparison measures. The injected fault destroys the client itself, and recovery logic in the client goes with it. On an append-only ledger, a committed divergence also cannot be undone: a compensating transaction annotates the orphaned event rather than removing it, and a reader between the two writes sees the divergent state as current. Recovery therefore reduces how often divergence is reached and how long it stays unannotated, not whether it is reachable; with one read–write set there is no partial state to recover from. We did not implement or measure a recovery-enabled two-chaincode client.

6.3. Throughput, Contention, and the MVCC Ceiling

The distinct-key regime of Table 7 is the expected operating mode for AM provenance, where each part carries its own identifier, and it recorded no MVCC conflicts on either testbed; the flat region of the distributed curve reflects the orderer configuration, not the design (Section 5.5.2). Updates to one actor's accumulator serialize on one state key. Fabric does not resubmit a rejected transaction, so recovery is the client's. Without retry most contended submissions failed (Section 5.9.4); with a jittered exponential-backoff retry every one committed exactly once, at 5.4 submissions per committed rating on average and up to 20 attempts and 8.2 s for the slowest. Eventual commitment is therefore a property of the retrying client, not a protocol guarantee, and an application that needs completion within a fixed time must budget for that tail or avoid concentrating ratings on one accumulator. Retry is safe because a rejected submission writes nothing and the lifecycle assertion rejects a resubmitted constrained event that had in fact committed. For every other event type, the chaincode provides no duplicate prevention. When a client cannot tell whether a submission committed, for example after a timeout or a crash between submission and confirmation, checking the ledger before resubmitting is the client's responsibility; a client that skips the check can record an unconstrained event, such as a delivery, and its rating twice. Buffering ratings under per-transaction keys and folding them in periodically, which the contract supports, removes the hot key at the cost of delayed score visibility and was not evaluated. On the distributed testbed, most contention failures occur at endorsement, not as MVCC conflicts (Table 8), and the soak of Section 5.5.4 shows a lagging peer producing the same failure, so a capacity model that treats contention as purely an MVCC phenomenon overstates the load it places on the orderer.

6.4. Collusion and Sybil Resilience

The 3-of-4 endorsement policy prevents any single organization from committing ledger state alone, but endorsement attests deterministic execution, not the truth of a submitted outcome. The enforced mapping policy removes self-report: an organization cannot rate an actor registered to itself on either rating path, cannot register or take over another organization's label, and cannot act under another organization's identity (Table 17). It does not establish that a submitter is entitled to its role: roles are declared by each organization's administrator (Section 3.6), so an organization that declares itself an inspector

can report inspection outcomes about another organization's manufacturer. The remaining exposure is collusion between organizations, a counterparty that misreports or withholds outcomes about a partner; the simulation of Section 5.9.5 quantifies its effect on the gate. Detecting it needs evidence the ledger cannot supply by itself: co-signature of an outcome by both parties, authenticated sensor attestation of the measurement behind it, or inspection by organizations the consortium has accredited. None is implemented here. Preventing self-rating is therefore not collusion resistance: only the attacks of Table 17 were tested on the ledger, and collusion and omission only by simulation. The consistency invariant of Section 3.1 is independent of these questions and of the endorsement threshold: it is enforced at the read–write set level. The stake-and-slash mechanism quantifies each rater's exposure; it becomes an economic deterrent only where the consortium backs the stake, and it is not a formal Nash equilibrium. A deployment seeking stronger guarantees can disable the direct rating path, removing ratings that are not tied to any recorded lifecycle event, while bridge ratings continue to be produced, staked, weighted and open to dispute (Section 3.5.3). Sybil resistance, requiring one identity per legal entity, is delegated to the issuance policy of each organization's certificate authority; because identities are keyed on MSP and certificate subject, an identity issued by one organization can act only as that organization. A consortium unwilling to place issuance with member organizations could instead bind participants to decentralized identifiers [31], at the cost of an external resolution and governance layer.

6.5. What the Scores Measure

A score in this system summarizes outcomes that counterparties recorded about an actor and that the consortium's endorsers accepted; it is not a validated measure of the actor's demonstrated performance, and this paper does not present it as one. An authorized counterparty can still record an inaccurate outcome, by error or by design: the mapping policy fixes who reports an outcome, not whether it is true, and endorsement attests that the chaincode executed deterministically, not that the physical event occurred as reported. Three gaps separate a score from such a measure, and closing each requires a consortium-wide assessment methodology, not a ledger mechanism. The first is acceptance criteria and verification: what counts as a pass for each event type, under which inspection or test standard, verified by whom, and with what supporting evidence, which the ledger can anchor by digest but not judge. The second is attribution of responsibility: an adverse outcome shows that a requirement was missed, not who caused it. A late delivery may reflect the carrier, the shipper's release time or a customs hold. The dispute path lets the rated organization contest attribution and an arbitrator reverse the rating with both halves of its update (Section 5.9.3), but the criteria the arbitrator applies are a consortium agreement this paper does not define. The third is comparability and thresholds: scores are comparable across actors only when their outcomes were assessed under comparable conditions and part complexity, and gate thresholds need a justification from the consortium's risk tolerance, for example from the false-acceptance and false-rejection trade-off of Section 5.9.5, rather than from defaults. Table 19 sets these questions out for each rating-producing event of the reference policy, with illustrative answers; the answers are the consortium's to give, and the ledger's part is to anchor the evidence by digest and record who submitted each outcome. The lifecycle scenario (Section 5.9.3) traces one case from evidence to rating: the manufacturer's organization disputed a failed inspection, attributing it to a powder lot, and the arbitrator overturned the rating, which subtracted it from the manufacturer's β and slashed the inspector's stake. In a deployment the verdict would rest on the inspection report, the build log and the supplier's certificate of conformance for the lot, each anchored

by digest; a finding against the lot would justify a separate rating of the supplier, which the dispute does not create.

Table 19. What an assessment method must define before the scores produced under the reference mapping policy (Table 4) can be read as measures of performance, by rating-producing event. Entries are the questions a consortium has to settle, with illustrative answers; none is implemented or validated here. The ledger anchors the evidence by digest and records the submitting organization, and the dispute path of Section 3.5.3 reverses a rating whose attribution is overturned.

Event (Rated)	Acceptance Criterion	Verification and Evidence	Attribution to Settle	Comparability and Threshold Basis
Material cert. (supplier)	Lot conforms to the material specification agreed with the buyer	Certificate of conformance; incoming test report	Supplier, or storage and handling after receipt	Like specifications compared; share of lots accepted first time
Print completion (mfr.)	Build finished with the approved parameters and no unresolved in-process nonconformance	Build log; parameter set; process monitoring record	Machine and process, powder lot, or design file	Part complexity; process family
Inspection (mfr.)	Part within the inspection plan's acceptance limits	Inspection report and scan data; inspector qualification	Manufacturing, post-processing, or inspection error	Same plan; detection limits of the method
Post-processing (mfr.)	Treatment and finishing to specification	Process records, e.g. furnace logs	Manufacturer or a separate post-processing provider	Process type
QA cert. (supplier)	Required standards listed and met	Certificate; standards cited	Which contribution the certificate covers	Standard; part class
Delivery (logistics)	Arrival by the agreed time ($t_a \leq t_e$) in the agreed condition	Proof of delivery; condition records	Carrier, shipper's release time, or customs hold	Route, distance and service level; width of the window

What the system guarantees is narrower, and it is what the evaluation demonstrates: every outcome recorded through the bridge commits together with its reputation consequence or not at all; under the enforced policy, outcomes are submitted only by counterparties that have declared the required role; and every input to a score is on the ledger, where it can be audited and disputed. The first guarantee is that of the bridge function `RecordProvenanceWithReputation`, which the fault-injection experiments exercised; the buffered variant defers the score update (Section 6.3), and the direct provenance and rating paths each record one half by design, so an outcome whose responsible party is unknown can be recorded without rating anyone.

6.6. Reputation Model Selection

The reputation model is an engineering choice constrained by on-chain execution requirements, not a claim of statistical optimality. Four constraints applied: constant-time updates (ruling out graph-propagation methods such as EigenTrust, which requires $O(|V|^2)$ computation per update in the worst case), explicit uncertainty quantification (ruling out simple averaging, which cannot distinguish a 0.8 score from two observations from a 0.8 score from 200), closed-form update rules (ruling out methods needing off-chain training or inference), and interpretability (ruling out opaque machine-learning-based scoring that

would obstruct regulatory audit). Beta-distribution scoring is the simplest model that satisfies all four: an update is two multiply-adds, the posterior supplies its own credible interval, and the (α, β) parameters are directly auditable as the accumulated weighted evidence of success and failure. Richer alternatives, such as Dirichlet distributions for multi-outcome ratings or hierarchical Bayesian models for inter-actor correlations, would provide more fine-grained statistical modeling but violate one or more of these constraints. Beta scoring is sufficient for the binary-outcome bridge signals that dominate AM lifecycle events. Section 5.9.5 compares it with a simple average and an exponentially weighted moving average: from five ratings on the Beta mean is slightly less accurate than the simple average, because its prior shrinks scores toward 0.5, with the gap nearly closed by 100 ratings; in exchange it removes the false acceptance of actors with a short lucky streak, which is the property a gate needs. The current model treats dimensions and actors independently; incorporating cross-dimension or inter-actor correlation structures remains future work and is not required to demonstrate the architectural consistency claim.

Every value that reaches a write set or a gate decision is computed with IEEE 754 addition, subtraction, multiplication, division and square root, which are correctly rounded, and with an exponential and logarithm built only from those operations. The submitted version computed decay with the Go standard library's `math.Pow`, whose exponential on amd64 is an assembly routine that selects a fused multiply-add or a separate multiply and add at run time according to the processor's features, so endorsers on different processors could in principle have rounded a decayed value differently. The revised chaincode uses the library's portable algorithms for the exponential and logarithm in place of the architecture-specific ones and is built without compiler fusion of multiply and add, so the computation no longer depends on the processor. Decay is an evaluation-time transform: gate checks and score queries derive effective parameters from the stored $(\alpha, \beta, t_{\text{last}})$ tuple using the transaction timestamp and never persist them, while the update path increments the stored accumulators without rescaling. Stored state therefore evolves only through the constant-time increments, and every endorser evaluates decay from the same timestamp.

Because decay is keyed to the time since an accumulator was last written, it acts differently on idle and active actors. During inactivity it discounts all evidence, and the score returns toward the prior. During continuous activity it discounts nothing, because every rating resets the elapsed time: old evidence keeps its full weight, the score tends toward a lifetime average, and it responds slowly to a change in performance. In the simulation of Section 5.9.5, after a drop in success probability from 0.95 to 0.60 at one event per day, a gate at 0.80 refused a median 139 days later, against 19 days had decay been applied on the write path. Evaluation-time decay was chosen because it keeps the stored accumulators an exact sum of recorded outcomes: a dispute reverses a rating by subtracting that rating's own contribution, and an auditor can recompute every score from the ratings on the ledger; write-time decay, which rescales the counts at every rating, would lose both. The cost is responsiveness, which matters for safety-critical AM: in the simulated case an actor whose success rate fell to 0.60 stayed above the gate for a median 139 days while four in ten of its outcomes were failures. Where a change in performance must be detected promptly, the score should therefore not be the only control. A deployment should act on individual adverse outcomes directly, for example by holding further authorizations after a failed inspection until its cause is resolved, or apply decay on the write path, which in the simulation detected the drop in 19 days at $\lambda = 0.98$ with 0.25% false refusals before it. Neither is implemented here; configurable write-time decay is future work.

6.7. Testbed Limitations

The wide-area experiment of Section 5.5.4 emulates distance with a uniform, constant delay on a LAN; a real WAN adds variable latency, loss and asymmetric paths, which were not evaluated. The scale experiment ran 8 and 12 organizations on four machines, so it measures more endorsements and more peers per machine together; consortia of that size on separate machines remain to be measured. These throughput figures describe desktop testbeds running synthetic transactions and do not predict an industrial deployment, whose capacity depends on its hardware, network and slowest peer. The open-loop sweeps located the saturation point of the four-desktop testbed, but neither testbed's absolute maximum was established: both closed-loop sweeps were still rising at $W = 400$, and on the single-host testbed the client process peaked there at 89.4% CPU, so that value is a lower bound; which trigger cut each block was not isolated. The soak lasted 12 h at one rate on one testbed. LevelDB supports only key-based reads; complex analytical queries require CouchDB or an off-chain index.

The Chaincode-as-a-Service deployment model carries a further limitation. The lifecycle package installed on each peer contains only a connection endpoint, so the chaincode definition committed to the channel provides no cryptographic binding to the binary that executes. Identity of the deployed binary across the four peers was verified out of band by comparing SHA-256 digests; Fabric does not enforce it.

Two governance functions are left to the consortium: admitting organizations to roles and funding stake (Sections 3.5.3 and 3.6); until then, the mapping policy constrains reporting relative to declared roles, and stake measures exposure in accounting units only.

Benchmarks used synthetic workloads with uniform random payloads, and the lifecycle scenario of Section 5.9.3 uses scripted outcomes rather than process data; real AM workflows exhibit temporal locality and correlated failures (e.g., a bad powder lot affecting multiple parts) that are not captured here, and characterizing them requires industrial data. Ordering-layer crash recovery was exercised once and is reported as a single observation. The distributed testbed runs three Raft consenters and the single-host testbed one. Stopping the leader mid-run, at transaction 401 of 1000, produced a term change from 4 to 5 and a new leader roughly 951 ms later, with no fork and with all four peers agreeing on height and current block hash afterwards, so the consensus layer behaved as Raft specifies. The client nevertheless committed nothing after the stop: all 593 subsequent transactions failed as `ORDERER_UNAVAILABLE`, because the channel's `OrdererOrg` `Endpoints` value listed a single orderer and the peer gateway had no other target. A consortium deployment must list every consenter in `Endpoints`. The same setting stalled block delivery during the revision: after the orderer it names had been stopped for seven days, three peers received no new blocks, although the other two consenters were ordering them, until the peers were restarted. The lifecycle state machine constrains four event constants only; a longer enforced chain requires extending the predecessor map in chaincode, so the taxonomy of Table 2 is an event set a deployment may record, not a sequence the implementation enforces end to end. The bridge overhead isolated by the phase decomposition of Section 5.2 is smaller than the run-to-run range and should be read as an upper bound. The Raft leader's memory grew monotonically within each measurement session, from 868 to 951 MB over 50 min and from 920 to 1085 MB over 60 min. Over the 12-h soak of Section 5.5.4, the three orderers' memory rose from 370–471 MiB to 402–500 MiB in the first hour and by a further 32–45 MiB over the remaining eleven, to at most 537 MiB, so growth slowed below the rate of the earlier sessions; a slow leak is not excluded. The decay coefficient $\lambda = 0.98$/day was chosen analytically and is runtime-configurable; Section 5.9.5 simulates its sensitivity, but calibrating it to industrial data remains open.

Every condition in Sections 5.2–5.5 before Section 5.5.4 is three independent runs: 500 transactions per run sequentially and 2000 concurrently, except the high-contention conditions at 500 per run. In Section 5.5.4 only the open-loop sweeps (three) and the control (two) are repeated; the delay, scale and soak results are single runs. The fault-injection conditions are single runs of 200 to 500 operations each, so the divergence bound is pooled across conditions rather than reported per condition. Endorsement latency was measured at the 3-of-4 threshold; because proposals are dispatched concurrently, a 2-of-4 policy would set the endorsement phase by the second-fastest peer rather than the third. The fault-injection study, its calibration, the comparison of arrangements and the reputation experiments of Sections 5.7–5.9 were all run on the single-host testbed. The empirical demonstration of the invariant and of the arrangements' relative cost is therefore scoped to that deployment, even though the invariant itself is not, and the single-host network's two organizations hold all six supply-chain roles in the reputation experiments.

6.8. Deployment in Urban Production Networks

Consider, as an illustrative deployment scenario, a municipal hospital sourcing a patient-specific implant from a print bureau a few miles away. The powder supplier certifies the lot, the bureau prints and post-processes the part, an accredited laboratory certifies the finished component, and a courier delivers it under controlled conditions. Selected outcomes among these are submitted through the bridge according to the deployment's reference mapping policy, and each such submission carries its reputation consequence for the organization responsible in the same transaction that records the event. At the point of release, the receiving institution issues a single `GetPartTrustReport` call, which returns the recorded part history together with the current standing of every organization that touched the part, at the read latency of Section 5.2. No party in that chain reports to any other, and none of them operates the ledger. Urban resilience planning considers this case, in which point-of-use manufacturing substitutes for a severed supply line and a city must accept parts from a bureau it has not worked with before. A reputation gate on release could exclude a newly registered bureau from the steps a city cares most about; with the $(2, 2)$ prior, a gate whose *minScore* does not exceed 0.5 admits newcomers, and Section 5.9.5 shows what such a probation costs in false acceptance.

7. Conclusions

Additive manufacturing supply chains span organizations whose records of each step and judgments of each partner must agree; systems that record provenance and score reputation as separate transactions let the two diverge.

This work co-deploys provenance, reputation and integration contracts in one Hyperledger Fabric chaincode, so that a lifecycle event, the reputation update it implies and their link commit together or not at all, by Fabric's commit guarantee applied to one read–write set. Under client crashes in the measured partial-write window, the unified design produced no divergence in 3900 fault-injected trials, 2900 with the submitted implementation (v1.0) and 1000 with the revised one (v1.1), whereas a client-orchestrated two-chaincode deployment diverged in all 1000 trials of its two control batches of 500; a same-channel `InvokeChaincode` arrangement built from the same source gave the same guarantee at 4.5 ms more endorsement and 11.7% lower throughput, in exchange for separately upgradeable contracts. On four desktops, open-loop load saturates near 12 transactions per second, and a 12-h soak at half that rate committed 98.7% of submissions; the ledger checks, covering every failed submission and 2000 sampled commits, found no partial record. These results establish atomicity, that each submission commits completely or not at all, not that every valid submission is eventually recorded: the 1.3% of soak submissions that failed were not

resubmitted and are absent from the ledger, and recording such submissions requires a client that retries, as in the contention experiment.

The reputation layer summarizes outcomes that counterparty organizations submit under an on-chain policy in a Beta posterior with a credible interval; gates use it to authorize future activities and never block the record of an outcome. The scores are records of submitted outcomes, not validated measures of performance (Section 6.5).

Three limitations matter most (Section 6.7). Roles are declared and stake is credited without accreditation or funding, so collusion between organizations, quantified by simulation, remains open. Evaluation-time decay responds slowly when an active actor's performance drops. Wide-area latency was emulated, and the larger consortia shared four machines. Future work follows from these: co-signature, accredited inspection or authenticated sensor attestation of outcomes, with zero-knowledge proofs [32] where a measurement is commercially sensitive; write-time decay as a configurable option; assessment criteria and attribution rules developed with industrial partners; and evaluation over a real WAN, with larger consortia on separate machines and under BFT ordering.

Supplementary Materials: The following supporting information can be downloaded at https://www.mdpi.com/article/10.3390/electronics15215086/s1: Supplementary Methods S1: Benchmark protocol of the distributed testbed; Supplementary Methods S2: Fault-injection markers and the accumulator-presence predicate; Table S1: Resource utilisation on the distributed testbed by container class; Table S2: The ten failure classes of `bench.js`, in the order they are tested; Table S3: Exact coverage (%) of the 95% credible interval, and probability (%) that the interval lies entirely below the true p, by success probability p and number of ratings n; Table S4: Mapping-coverage test: one passing ($v = 1$) and one failing ($v = 0$) outcome for each rule of the reference policy, under enforcement; Table S5: Experiments of the revision: testbed, chaincode binary, program and output; and the figure scripts `gen_tx_timeline.py` and `gen_scaling.py`, with their provenance files.

Author Contributions: Conceptualization, R.A.; methodology, R.A.; software, R.A.; validation, R.A., A.K., H.B.T. and P.M.T.; investigation, R.A.; writing—original draft preparation, R.A.; writing—review and editing, A.K., H.B.T., P.M.T., H.K. and M.D.; supervision, M.D.; funding acquisition, M.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Science Foundation E-RISE RII program, grant number OIA-2417062.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The chaincode implementation, the benchmark and fault-injection harnesses, the analysis and simulation programs that recompute the reported results, and the underlying run outputs are available in the `am-unified` repository on GitHub (versions `v1.0-smartcities` and `v1.1-electronics`). The implementation evaluated in the submitted version (v1.0) is the immutable commit `089d99c67583c6186d3ce2c6b4f9c9d196131082`, tagged `v1.0-smartcities`: https://github.com/raddadalmaayn/am-unified/tree/089d99c67583c6186d3ce2c6b4f9c9d196131082 (accessed on 27 October 2026), archived at Zenodo, DOI https://doi.org/10.5281/zenodo.21970890. The revised implementation (v1.1), the drivers of the revision experiments and their outputs (Sections 5.5.4, 5.8 and 5.9) are the immutable commit `3be00ccaa33c8423478f77101c0046134ceec042`, tagged `v1.1-electronics`: https://github.com/raddadalmaayn/am-unified/tree/3be00ccaa33c8423478f77101c0046134ceec042 (accessed on 27 October 2026), with the per-transaction benchmark records attached to that release, and archived at Zenodo, DOI https://doi.org/10.5281/zenodo.23351118. Programs added or corrected after that release, namely the mapping-coverage test, the exact coverage calculation, the throughput-table generator, the four-desktop analysis and the figure scripts, are provided with their outputs in the Supplementary Materials, at their repository paths. The fault-injection harness and the per-condition aggregates for the conditions reported in Section 5.7, together with the complete fault-free calibration run from which the transaction timeline

of Figure 6 is derived, are under `atomicity_comparison/`. Internal laboratory network addresses and operator paths are redacted in the published artifacts; no measured value, count or identifier was altered.

Acknowledgments: The authors thank the Research Center for Distributed Resilient and Emergent-Intelligence-Based Additive Manufacturing (DREAM) at the University of New Mexico for supporting this work.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AM	Additive Manufacturing
BFT	Byzantine Fault Tolerant
CA	Certificate Authority
CCAAS	Chaincode-as-a-Service
CrI	Credible Interval
ECDSA	Elliptic Curve Digital Signature Algorithm
EWMA	Exponentially Weighted Moving Average
gRPC	gRPC Remote Procedure Call
IoT	Internet of Things
JSON	JavaScript Object Notation
LAN	Local Area Network
MSP	Membership Service Provider
MVCC	Multi-Version Concurrency Control
QA	Quality Assurance
RTT	Round-Trip Time
SDK	Software Development Kit
SHA-256	Secure Hash Algorithm 256
TLS	Transport Layer Security
TPS	Transactions Per Second
URL	Uniform Resource Locator
VSCC	Validation System Chaincode
WAN	Wide Area Network

References

1. Grünewald, A.; Stuckmann-Blumenstein, P.; Keitzl, P.; Krämer, L. Blockchain and Additive Manufacturing: A Taxonomy of Business Models. *Front. Blockchain* **2025**, *8*, 1563909. [CrossRef]
2. Leng, J.; Ruan, G.; Jiang, P.; Xu, K.; Liu, Q.; Zhou, X.; Liu, C. Blockchain-Empowered Sustainable Manufacturing and Product Lifecycle Management in Industry 4.0: A Survey. *Renew. Sustain. Energy Rev.* **2020**, *132*, 110112. [CrossRef]
3. *AS9100D*; Quality Management Systems, Requirements for Aviation, Space, and Defense Organizations. SAE International: Warrendale, PA, USA, 2016. [CrossRef]
4. *ISO 13485:2016*; Medical Devices, Quality Management Systems, Requirements for Regulatory Purposes. International Organization for Standardization (ISO): Geneva, Switzerland, 2016. Available online: https://www.iso.org/standard/59752.html (accessed on 9 October 2026).
5. Priyadarshini, J.; Singh, R.K. Blockchain Technology Integration with Additive Manufacturing: Analysis of Challenges. In Proceedings of the 14th Annual International Conference on Industrial Engineering and Operations Management, Dubai, United Arab Emirates, 12–14 February 2024; pp. 623–630. [CrossRef]
6. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In Proceedings of the 13th EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15. [CrossRef]
7. Alkhader, W.; Alkaabi, N.; Salah, K.; Jayaraman, R.; Arshad, J.; Omar, M. Blockchain-Based Traceability and Management for Additive Manufacturing. *IEEE Access* **2020**, *8*, 188363–188377. [CrossRef]

8. Cui, P.; Dixon, J.; Guin, U.; Dimase, D. A Blockchain-Based Framework for Supply Chain Provenance. *IEEE Access* **2019**, *7*, 157113–157125. [CrossRef]
9. Jøsang, A.; Ismail, R. The Beta Reputation System. In Proceedings of the 15th Bled Electronic Commerce Conference, Bled, Slovenia, 17–19 June 2002. Available online: https://aisel.aisnet.org/bled2002/41/ (accessed on 9 October 2026).
10. Liu, J.; Yeoh, G.; Gao, L.; Gao, S.; Ngwenyama, O. Designing a Secure Blockchain-Based Supply Chain Management Framework. *J. Comput. Inf. Syst.* **2023**, *63*, 592–607. [CrossRef]
11. Haw, J.; Dola, T.A.; Sing, S.L.; Tan, E.Y.S.; Liu, A.Z. Blockchain for Additive Manufacturing. In *Additive Manufacturing Design and Applications*; ASM Handbook, Volume 24A; ASM International: Materials Park, OH, USA, 2023; pp. 582–587. [CrossRef]
12. Ferreira, I.A.; Palazzo, G.; Pinto, A.; Pinto, P.; Sousa, P.; Godina, R.; Carvalho, H. A Blockchain Architecture with Smart Contracts for an Additive Symbiotic Network—A Case Study. *Oper. Manag. Res.* **2025**, *18*, 537–553. [CrossRef]
13. Jøsang, A.; Ismail, R.; Boyd, C. A Survey of Trust and Reputation Systems for Online Service Provision. *Decis. Support Syst.* **2007**, *43*, 618–644. [CrossRef]
14. Rathee, G.; Ahmad, F.; Jaglan, N.; Konstantinou, C. A Secure and Trusted Mechanism for Industrial IoT Network Using Blockchain. *IEEE Trans. Ind. Inform.* **2023**, *19*, 1894–1902. [CrossRef]
15. Wu, Y.; Zhang, Y. An Integrated Framework for Blockchain-Enabled Supply Chain Trust Management towards Smart Manufacturing. *Adv. Eng. Inform.* **2022**, *51*, 101522. [CrossRef]
16. Kamvar, S.D.; Schlosser, M.T.; Garcia-Molina, H. The EigenTrust Algorithm for Reputation Management in P2P Networks. In Proceedings of the 12th International Conference on World Wide Web (WWW), Budapest, Hungary, 20–24 May 2003; pp. 640–651. [CrossRef]
17. Cheng, Z.; Chow, M.-Y. An Augmented Bayesian Reputation Metric for Trustworthiness Evaluation in Consensus-Based Distributed Microgrid Energy Management Systems with Energy Storage. In Proceedings of the 2nd IEEE International Conference on Industrial Electronics for Sustainable Energy Systems (IESES), Cagliari, Italy, 1–3 September 2020; pp. 215–220. [CrossRef]
18. Yavaprabhas, K.; Pournader, M.; Seuring, S. Blockchain and Trust in Supply Chains: A Bibliometric Analysis and Trust Transfer Perspective. *Int. J. Prod. Res.* **2025**, *63*, 5071–5098. [CrossRef]
19. Brandín, R.; Abrishami, S. IoT-BIM and Blockchain Integration for Enhanced Data Traceability in Offsite Manufacturing. *Autom. Constr.* **2024**, *159*, 105266. [CrossRef]
20. Thakkar, P.; Nathan, S.; Viswanathan, B. Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform. In Proceedings of the IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Milwaukee, WI, USA, 25–28 September 2018; pp. 264–276. [CrossRef]
21. Enare Abang, J.; Takruri, H.; Al-Zaidi, R.; Al-Khalidi, M. Latency Performance Modelling in Hyperledger Fabric Blockchain: Challenges and Directions with an IoT Perspective. *Internet Things* **2024**, *26*, 101217. [CrossRef]
22. Kim, J.-W.; Song, J.-G.; Park, I.-H.; Jo, D.-H.; Kim, Y.-J.; Jang, J.-W. Dependency Reduction Techniques for Performance Improvement of Hyperledger Fabric Blockchain. *Big Data Cogn. Comput.* **2025**, *9*, 32. [CrossRef]
23. Baldo, T.; Migliardi, M. Factoring Economic Biases Out of Tokenomics. In Proceedings of the 46th MIPRO ICT and Electronics Convention, Opatija, Croatia, 22–26 May 2023; pp. 1364–1369. [CrossRef]
24. Tshakwanda, P.M.; Tsegaye, H.B.; Almaayn, R.; Devetsikiotis, M. Fortifying Multi-Agent Architectures in Smart Manufacturing: Leveraging Federated Learning and Blockchain for Security and Resilience. In Proceedings of the 2025 IEEE 22nd Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2025; pp. 1–2. [CrossRef]
25. Almaayn, R.; Tsegaye, H.B.; Karukutla, A.; Tshakwanda, P.M.; Worku, Y.M.; Kumar, H.; Devetsikiotis, M. Lightweight Blockchain Data Model for Provenance Tracking in Additive-Manufacturing Supply Chains: A Quantitative Performance Analysis. In Proceedings of the 2026 IEEE 23rd Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 9–12 January 2026; pp. 1–6. [CrossRef]
26. Almaayn, R.; Karukutla, A.; Tsegaye, H.B.; Tshakwanda, P.M.; Kumar, H.; Ali, A.M.; Devetsikiotis, M. Adversarially Robust Blockchain Reputation System for Additive Manufacturing Supply Chains. In Proceedings of the 2026 IEEE 16th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 5–7 January 2026; pp. 1401–1407. [CrossRef]
27. Wilson, E.B. Probable Inference, the Law of Succession, and Statistical Inference. *J. Am. Stat. Assoc.* **1927**, *22*, 209–212. [CrossRef]
28. Brown, L.D.; Cai, T.T.; DasGupta, A. Interval Estimation for a Binomial Proportion. *Stat. Sci.* **2001**, *16*, 101–133. [CrossRef]
29. Hyperledger Foundation. Benchmarking Hyperledger Fabric 2.5 Performance. LF Decentralized Trust Blog. 2023. Available online: https://www.lfdecentralizedtrust.org/blog/2023/02/16/benchmarking-hyperledger-fabric-2-5-performance (accessed on 2 July 2026).
30. Hanley, J.A.; Lippman-Hand, A. If Nothing Goes Wrong, Is Everything All Right? Interpreting Zero Numerators. *JAMA* **1983**, *249*, 1743–1745. [CrossRef]

31. World Wide Web Consortium. Decentralized Identifiers (DIDs) v1.0. W3C Recommendation. 19 July 2022. Available online: https://www.w3.org/TR/2022/REC-did-core-20220719/ (accessed on 9 October 2026).
32. Goldreich, O. Zero-Knowledge Twenty Years After Its Invention. Electronic Colloquium on Computational Complexity (ECCC), Technical Report TR02-063. 2002. Available online: https://eccc.weizmann.ac.il/report/2002/063/ (accessed on 9 October 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.