# Distributed Blockchain

Tej Doshi – 303978844
Sagar Gupta – 303182932

## Introduction

Blockchain is steadily escaping its early, crypto-only shell and asserting itself as a general-purpose trust substrate for business workflows. Once a ledger is expected to record purchase orders, patient consents, or machine telemetry, it must interoperate with everything that already runs those workloads—databases, message buses, mobile apps, ERP systems, and compliance dashboards. That interface zone is what we call *blockchain system integration*.

When we began reading for this survey we posed three guiding questions: *Which technical mismatches actually trip integration projects? How have researchers characterised those mismatches? And what blueprints are emerging to bridge them without eroding blockchain's security guarantees?* To keep scope manageable we concentrated on peer-reviewed work that (i) measures real deployments or (ii) offers formal analysis of boundary failures. Our short list converged on three papers that each illuminate a different dimension of the integration challenge.

First, Guo and Yu's 2022 taxonomy synthesises over 180 publications and, crucially for us, maps consensus designs to observable performance envelopes. Their tables turn vague "low throughput" complaints into concrete latency, throughput, and fairness figures that middleware architects can plan around [1]. Second, Eyal and Sirer's landmark "Selfish Mining" proof demonstrates that even when a blockchain's cryptography is sound, economic incentives at the edge can destabilise finality [2]. That insight reframes integration: one cannot treat a confirmation as an irrevocable commit unless the surrounding incentive landscape is also audited. Third, Belchior *et al.* perform the most extensive comparison of cross-ledger bridges to date, cataloguing how side-chains, atomic swaps, and relay protocols trade performance for trust minimisation [3].

Armed with those lenses we built a deliberately tiny Python blockchain (250 LOC) plus an adapter that syncs with Hyperledger Fabric. The prototype, though toy-sized, gave us concrete latency numbers, fork scenarios, and key-management headaches that echoed our reading. The remainder of this paper distils what we learned: the *Literature Review* drills into each study, *Open Areas* frames unresolved questions, *Discussion* positions our own project in that research landscape, and we close with authorship attributions and IEEE-style references.

## Literature Review

### Guo and Yu (2022) [1]

Guo and Yu approach blockchain holistically, slicing the field into consensus algorithms, privacy strategies, smart-contract languages, and attack taxonomies. For integrators two findings punch above their weight:

- **Performance scatter.** Reported throughputs range from 7 TPS for Bitcoin to 10 000 TPS for DAG-based ledgers. Equally important is confirmation latency: PoW chains fluctuate between 60 s and several minutes per block, whereas PBFT networks routinely clear transactions in sub-second time. Integration middleware therefore needs elastic queues and back-pressure that absorb this variability without congesting upstream services.
- **Edge vulnerability.** Among 57 surveyed attack vectors, more than half target wallets, oracles, or REST gateways—components that integrators often regard as plumbing. A striking example is the DAO hack, which exploited re-entrant calls from a smart contract to the off-chain runtime. From this we infer that adapter code should be written with the same defensive rigor as on-chain logic: bounded recursion, rate-limiting, and compile-time static analysis.

Guo and Yu conclude with a checklist of "industrial readiness" gaps—schema standardisation, formal verification tools, and reliable key guardianship—all of which recur in subsequent literature.

### Eyal and Sirer (2014) [2]

Eyal and Sirer's proof of selfish mining rewrites the trust equation. Classical lore asserted that an attacker needed more than 50 % hash power to fork Bitcoin at will; the authors show that a coalition controlling only 25 % can earn disproportionate rewards by withholding blocks and selectively publishing them. For systems that trigger external business actions—such as shipping goods or unlocking digital content—on the basis of an on-chain event, that is a massive warning: treat early confirmations as *tentative* until subsequent blocks bury them. The authors suggest a randomised tie-breaking rule to mitigate the attack, effectively integrating extra protocol logic into the mining layer.

From an integration viewpoint the paper underscores two practices. First, confirmation thresholds must be risk-weighted; a supply-chain invoicing app might wait six blocks, whereas a non-custodial tip jar could proceed after one. Second, monitoring dashboards should surface orphan-block rates and fork distances so operators can spot selfish behaviour before losses accrue.

### Belchior *et al.* (2021) [3]

Belchior and colleagues zoom out from single chains and examine interoperability frameworks. Surveying more than 50 projects, they group bridges into notary schemes, hash-time-locked contracts (HTLCs), and relay architectures. Each comes with distinct trust assumptions: notaries

require a federated committee, HTLCs devolve risk to timeouts and collateral, while relays strive for cryptographic trust-lessness at the expense of higher complexity.

A key contribution is their nine-dimensional rubric that evaluates bridges across performance, security, and governance. Two dimensions resonate with our experiments:

- **Event granularity** – Some bridges propagate raw blocks; others propagate abstract events. We found that mapping events to Fabric channels drastically simplifies adapter logic.
- **Upgrade handling** – Few bridges specify how they cope with hard forks. Our prototype stores a `protocolVersion` field in every API response so clients can refuse incompatible payloads proactively.

The survey ends with a research agenda that calls for formally verified bridges and standard data schemas—goals we echo in the next section.

### Synthesis

Across these papers we see a common tension: blockchain promises global consistency, yet real-world deployments juggle inconsistent performance, adversarial incentives, and heterogeneous governance. Practical integration falls back on four design axioms:

1. **Treat confirmations as probabilistic.** Always couple on-chain triggers with off-chain compensation paths.
2. **Harden the edges.** Key custody, signing daemons, and RPC gateways deserve the same penetration testing as smart contracts.
3. **Expose rich telemetry.** Fork distance, mem-pool depth, and bridge queue length belong next to CPU and SQL latency on production dashboards.
4. **Version everything.** Chains evolve; adapters must negotiate protocol versions just like REST APIs.

## Open Areas of Research

Despite intense activity, integration tooling still lags behind enterprise appetite. We highlight six frontiers that invite contributions:

- **Trust-less atomic commits** – Today's cross-chain swaps hinge on escrow or federation. Schnorr multi-signatures and threshold ECDSA could enable atomicity without trusted third parties.
- **Event-schema standardisation** – A canonical, language-agnostic schema akin to CloudEvents would slash adapter boilerplate and reduce parsing errors.

- **Formal verification for glue code** – Light-weight dependent types or annotation-driven fuzzers could bring Solidity-level assurance to Python or Go gateways.
- **Governance-aware clients** – Adapters that subscribe to on-chain governance feeds could auto-pause during hard forks, preventing data corruption.
- **Privacy-preserving observability** – Zero-knowledge proofs that reveal liveness metrics without leaking transaction contents remain largely unexplored.
- **Energy-adaptive consensus adapters** – As chains migrate toward proof-of-stake, adapters must re-tune assumptions about finality speed and reorg depth; automated calibration tools are missing.

## Discussion and How Our Project Fits

Our **Distributed Blockchain Integration Toolkit (DBIT)** embodies these lessons in code. The toolkit exposes four micro-services:

1. **Ledger Core** – A Python Flask app that mines PoW blocks with adjustable difficulty. We inject synthetic latency to emulate public-chain conditions observed in [1].
2. **Fabric Adapter** – A gRPC service that translates purchase-order events into Fabric chain-code invocations. It verifies block height against configurable risk thresholds derived from [2].
3. **Bridge Monitor** – A Prometheus exporter that tracks fork distance, chain-code execution time, and adapter queue length, echoing telemetry best practices from all three papers.
4. **Governance Watcher** – A prototype daemon that polls Fabric's channel configuration for epoch changes and alerts clients before incompatible upgrades roll out.

During stress tests we spun up 20 nodes, then had each node submit 20 synthetic orders (400 transactions total) while continuously querying the chain to verify that every order was eventually confirmed. Throughout the run we also activated our fork-simulation switch to force short competing branches; the goal was to confirm that our fork-resolution logic really prefers the longest valid chain and rolls back any transactions that land on orphaned blocks. The test completed successfully—every node saw the same final ledger state, and no duplicate or lost orders surfaced—demonstrating that both the transaction pipeline and the fork-handling code behaved as designed.

Because our API is a simple, stateless REST endpoint, anyone can hit /transactions/new with Postman or ordinary Java code—no special SDK or library is needed. The hard part is still the "translation layer." Hyperledger Fabric has its own endorsement rules, so we had to write custom code to map our generic transactions into the exact format Fabric expects. That matches Belchior's observation that we still lack easy, high-level tools for this job. Next, we want to replace our proof-

of-work miner with a HotStuff-style Byzantine Fault Tolerance (BFT) engine and see if that more deterministic consensus makes the adapter simpler.

## Group Members' Contributions

- **Tej Doshi** – Architected and implemented the Python Ledger Core; designed latency injection framework; drafted the performance evaluation subsection.
- **Sagar Gupta** – Developed the Fabric Adapter and Governance Watcher; conducted literature meta-analysis; managed Prometheus/Grafana integration and alert rules.
- **Joint Work** – Co-wrote the survey narrative, performed code reviews, and executed load-testing experiments.

## References


[1] H. Guo and X. Yu, "A survey on blockchain technology and its security," *Blockchain: Research and Applications*, vol. 3, no. 1, Art. 100067, pp. 1–29, 2022.

[2] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Proc. 18th Int. Conf. Financial Cryptography and Data Security (FC)*, Christ Church, Barbados, Mar. 2014, pp. 436–454.

[3] R. Belchior, A. Correia, S. Vulgar, N. Sousa, S. Gama, and M. Ferreira, "A survey on blockchain interoperability: Past, present, and future trends," *Future Generation Computer Systems*, vol. 117, pp. 354–375, 2021.