Fast and Interactive Byzantine Fault-tolerant Web Services via Session-Based Consensus Decoupling

Ahmad Zaki Akmal, Azkario Rizky Pratama, and Guntur Dharma Putra
Universitas Gadjah Mada, Indonesia
ahmad.zaki.akmal@mail.ugm.ac.id, {azkario, gdputra}@ugm.ac.id

***Abstract*—Byzantine fault-tolerant (BFT) web services provide critical security and integrity guarantees for distributed applications but face significant latency challenges that hinder interactive user experiences. We propose a novel two-layer architecture that reconciles the fundamental trade-off between security and responsiveness in BFT systems. Our approach introduces a session-aware transaction buffer layer (Layer 2) that delivers immediate feedback to users through consensus simulation, while periodically committing batched operations to a fully Byzantine fault-tolerant consensus layer (Layer 1). By separating interactive operations from consensus finalization, our system achieves responsive user experiences of under 200ms, while maintaining strong BFT security guarantees. We demonstrate the efficacy of our architecture through a supply chain management implementation, where operators require both immediate feedback during multi-step workflows and tamper-proof record keeping. Our extensive evaluation shows that our Layer 2 operations perform four times faster than the Layer 1 counterpart, while substantially preserving the end-to-end transaction integrity. Our approach enables BFT applications in domains previously considered impractical due to latency constraints, such as metaverse environments, where users require both responsive interaction and guaranteed state consistency.**

***Index Terms*—web services, Byzantine fault tolerance, session, interactivity, responsiveness, layered architecture**

I. Introduction

The concept of metaverse represents a shift in digital interaction, evolving beyond isolated virtual experiences toward persistent, interconnected 3D spaces where users can socialize, work, and engage in commerce [1], [2]. The metaverse vision aligns naturally with Web3 principles, where blockchain technology enables decentralized ownership, transferable digital assets, and trustless transactions [3]. Unlike traditional web platforms where central authorities alone control user data and digital possessions, Web3 metaverses allow users to truly own their virtual assets and maintain their identity across multiple virtual worlds. As these interconnected economies mature, enabling users to trade assets with real-world values [4], the underlying web services must evolve beyond traditional centralized architectures. Open metaverses particularly benefit from Byzantine fault tolerance due to their decentralized nature [5], ensuring consensus and transaction validity without relying on trusted intermediaries, a necessity when handling valuable digital assets in environments without central authority.

System reliability approaches have evolved to address increasingly complex fault scenarios. For relatively minor crash faults, various strategies including replication, monitoring, and machine learning algorithms have been developed [6]. However, handling Byzantine faults, where nodes behave arbitrarily or maliciously, presents a significantly greater challenge [7]. While Crash Fault-tolerant (CFT) systems maintain consistency when nodes simply fail [6], Byzantine Fault-tolerant (BFT) systems should operate correctly even when up to one-third of the nodes act maliciously [7], [8]. The capability to tolerate Byzantine faults is crucial for metaverse applications where digital assets represent real economic value, making them attractive targets for sophisticated attacks beyond simple node failures.

In the literature, BFT web services have been developed extensively over the past two decades. Early approaches like Thema [9] and BFT-WS [10] implemented BFT properties while maintaining compatibility with standard web protocols, but relied on centralized middleware components that could become bottlenecks. More recent designs like WebBFT [11] improved upon these foundations but still faced scalability challenges. The state-of-the-art DeWS (Decentralized Web Services) made a significant advancement by replacing the conventional *request-compute-response* model with a *request-compute-consensus-log-response* approach that ensures all transactions are validated by a fully distributed network of nodes [12]. However, DeWS introduces significant latency between user actions and system responses, which takes approximately one second for a 15-node system configuration. Such delays, while acceptable for some applications, create substantial friction in interactive environments where users expect near-instantaneous feedback. The latency challenge represents a well-known limitation in blockchain systems generally, creating a fundamental tension between security guarantees and interactive responsiveness required for some applications.

In this paper, we propose a two-layered web service architecture to address the latency challenge while maintaining Byzantine fault tolerance. The first layer (L1) conducts a full BFT consensus for each request across a distributed network of nodes, while the second layer (L2) implements a novel session-based transaction buffer that maintains synchronization with the first layer's state, enabling accurate consensus simulation that provides immediate feedback to users. Our approach delivers interactive responsiveness without sacrificing the integrity protections essential for mission-critical applications. We introduce the concept of *sessions*, where related operations are grouped and processed through the responsive L2 simu-

lation before being committed as a batch to the more secure but slower L1 consensus. As such, our approach significantly reduces both the perceived latency and the frequency of slower L1 consensus, by providing immediate feedback to users and by batching multiple related operations into a single session, respectively.

Our work and innovation offer these key contributions:

- We introduce *a two-layer architecture for BFT web services* that separates interactive simulation from consensus finalization, enabling responsive user experiences while maintaining security guarantees.
- We develop *a transaction buffering mechanism* that groups related operations into sessions, simulates their execution with up-to-date state data, and commits them as atomic units, reducing perceived latency while preserving transactional integrity.
- We provide *a fully functional proof-of-concept* for our proposed solution through a supply chain scenario that demonstrates its practical feasibility along with a detailed performance evaluation. Our proof-of-concept showcases an architecture that can be applied to metaverse environments, where users require responsive multi-step interactions while maintaining BFT for final state changes.

The remainder of this paper is organized as follows. Section II reviews our motivation and related work on BFT systems and Layer 2 technologies. Section III presents our system architecture, while Section IV describes our implementation details. We conclude our work in Section V.

II. Motivation and Related Work

We introduce a motivating scenario involving a supply chain workflow with multiple steps of verification, which includes several stakeholders: a supplier that provides packages, a warehouse operator that processes and inspects packages, and a courier that delivers them to final destinations. Each package undergoes verification, quality checks, and labeling before being handed off to couriers.

A major challenge in cases such as supply chain processes is balancing the need for immediate feedback during operations while also maintaining a secure and immutable record. Traditional systems often struggle with this trade-off, either reducing the throughput to ensure security or sacrificing data integrity for better responsiveness. In our scenario, warehouse operators require instant feedback during each step of package processing, while the business management needs to ensure correctness and establish trust between different stakeholders.

The challenge becomes evident during high-volume periods when operators cannot afford delays waiting for consensus verification between steps, yet the business cannot compromise on creating an auditable and tamper-proof record of all activities. For example, when an operator scans an incoming package, they need immediate confirmation and details about expected contents, not a delayed response while the system reaches consensus with other nodes.

Our two-layer architecture addresses this fundamental tension between immediate feedback and ensuring tolerance to Byzantine faults. The Layer 2 component provides immediate validation and feedback to operators, maintaining a session that tracks the package through all processing steps. Once the package completes its warehouse journey, the entire session is committed as a single transaction to Layer 1, where BFT consensus ensures the record becomes immutable and verifiable by all stakeholders.

While we choose this supply chain workflow to illustrate the challenge, similar requirements exist in numerous domains including healthcare, financial services, e-government, and even metaverse environments. In metaverses, for example, the session-aware buffer layer would allow participants to engage in complex multi-step interactions (such as crafting, property development, or collaborative creation) with responsive feedback, while still ensuring that the final state changes are properly recorded on a distributed ledger for long-term persistence and verification. In each of these domains, participants need both high responsiveness during operations and guaranteed integrity of the final record, which our proposed architecture directly addresses.

This widespread challenge of balancing responsiveness and fault tolerance leads us to develop a solution that does not force a choice between the two, but instead offers both. To design a system that solves the problem, we study several notable research efforts and implementations of BFT in the context of web services, summarized in the related work section below.

Related work. Byzantine Fault Tolerance (BFT) addresses the challenge of maintaining system correctness despite arbitrary failures or malicious behavior in some components. In web services, BFT is particularly important when operations span multiple organizations with different trust boundaries [7].

Practical Byzantine Fault Tolerance (PBFT) [8] made BFT implementations feasible by reducing communication complexity from exponential to polynomial. Zyzzyva [13] further improves performance by lowering latency, involving client to assist in validations. However, PBFT, Zyzzyva, and its derivatives still face scalability challenges, particularly in high-throughput interactive applications.

Several approaches have applied BFT concepts to web services, including Thema [9], BFT-WS [10], WebBFT [11], and the most recent, DeWS [12]. Thema and BFT-WS implement Byzantine fault tolerance for Web Services while maintaining compatibility with standard SOAP and WSDL protocols, but they rely on centralized middleware architectures, which can become potential bottlenecks or single points of failure. WebBFT later improved on these ideas by making BFT services accessible to browser-based clients. It supported real-time collaborative features like publish-subscribe updates and worked with modern web technologies such as WebSockets [11]. Despite these advances, WebBFT faced several challenges. It introduced noticeable performance overhead from browser-side cryptography and JSON handling, and was vulnerable to denial-of-service attacks caused by malicious clients triggering repeated leader changes. These issues limited its scalability and robustness in adversarial environments. A key similarity among these earlier efforts is that they achieve

BFT by reaching consensus on the computation result of a single web server or through the inclusion of centralized components, thus lacking full computation replication across nodes. In contrast, Ramachandran et al. argue that replicating the complete computation at each node is necessary to ensure safety in the presence of Byzantine faults, a principle that underpins their design of DeWS [12].

DeWS implements a brand new interaction model: *request-compute-consensus-log-response*. All operations undergo replication and consensus validation before responses are returned to clients, ensuring that responses are agreed upon by a quorum of nodes. This approach provides strong integrity and auditability guarantees but introduces significant latency—approximately 935ms with 15 nodes even in a Docker container network, which represents an ideal environment compared to real-world deployments [12].

This latency creates a fundamental tension between Byzantine fault tolerance and interactivity, making DeWS challenging to apply in scenarios requiring frequent user interactions or real-time feedback.

Layer 2 blockchain solutions address similar scalability challenges by offloading transaction processing from the main chain. Techniques such as rollups and sidechains enable higher throughput and lower latency while maintaining security by periodically anchoring state to the layer 1 chain [14], [15].

Session management, the grouping of related operations into logical units, has been explored in distributed systems primarily for user authentication and state tracking [16]. However, its potential for optimizing consensus operations remains largely unexplored in BFT web services.

While these previous works have advanced Byzantine fault tolerance in web services, they collectively fail to address the fundamental trade-off between interactive responsiveness and strong BFT guarantees. The absence of a solution that maintains both low-latency interactivity and strong BFT guarantees represents a significant gap in the literature. Our two-layer architecture directly addresses the gap by separating the concerns of immediate feedback and consensus validation, enabling responsive user experiences without sacrificing the integrity guarantees of Byzantine fault tolerance.

III. System Model and Architecture

In this section, we describe our proposed solution that addresses the latency challenges in distributed web services while maintaining Byzantine fault tolerance.

A. System Components

Our model consists of two distinct layers: Layer 2 (L2) for transaction simulation and Layer 1 (L1) for Byzantine fault-tolerant consensus. These layers work together to provide low-latency interactions during workflow execution while ensuring final state consistency through consensus. Below we describe the definition of our system components in more detail.

1) ***Node****:* A node N is the fundamental building block of the distributed network. In our design, each node consists of a web server, a consensus engine, and a database. Nodes connect with each other, forming the network of a layer.

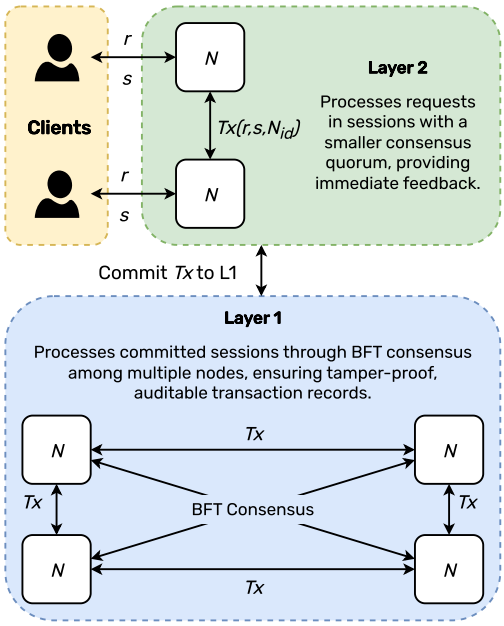

Fig. 1. Our two-layer architecture for BFT web services, showing client interactions with Layer 2 nodes and eventual session commitment to the Layer 1 BFT consensus network.

2) ***Service Handler****:* A Service Handler is a function that processes a specific type of HTTP request, identified by its method and path. Formally, for a request r, a service handler is a function $H : R \to S$ that maps from the request domain R to the response domain S. The function may also signal exceptional conditions through implementation-specific mechanisms. Service handlers encapsulate endpoint-specific functionality, allowing the system to maintain separation between routing and business logic.

3) ***Service Registry****:* The Service Registry is a management component that maintains a mapping between API routes (HTTP method and path combinations) and their corresponding service handlers. It can be represented as $\mathcal{R} = \{(m_i, p_i) \mapsto H_i\}$ where m_i is an HTTP method, p_i is a URL path pattern, and H_i is the associated handler function. The registry provides a lookup function $\mathcal{L}(m, p) \to H$ that maps an incoming request's method m and path p to the appropriate handler H. By centralizing route management, the Service Registry enables the system to apply consistent processing between the web server and the consensus replication.

4) ***Transaction****:* A Transaction $Tx = (r, s, N_{id})$ consists of a request r, its associated response s generated by a service handler, and the identifier N_{id} of the originating node. Transactions are the fundamental unit of consensus in the system, allowing nodes to verify whether a given request produces the expected response according to the business rules encoded in service handlers. A transaction is considered valid when it meets two conditions: (1) the response was generated by applying the correct service handler to the request, and (2) a majority of nodes in the network have independently verified

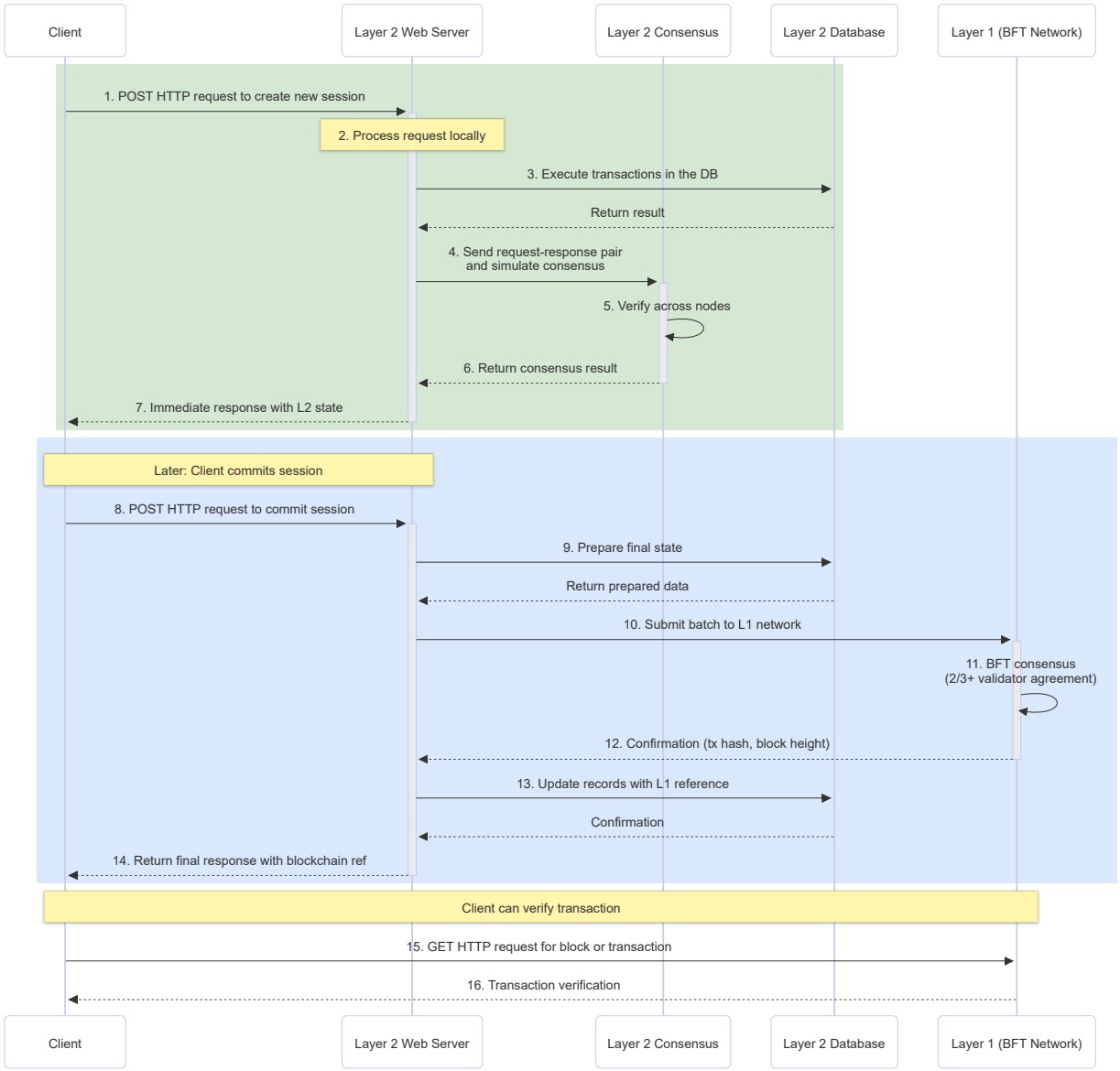

Fig. 2. Sequence diagram illustrating the two-layer transaction processing flow. The workflow begins with client session creation handled locally by L2 (marked green), providing immediate responses. Transactions are then committed to the L1 (marked blue), requiring validator agreement before confirmation.

that applying the same service handler to the request produces an identical response. This verification process ensures that Byzantine nodes cannot tamper with transaction results. Otherwise, it is marked as an invalid transaction.

*5) **Transaction Originator**:* The Transaction Originator is the node that first receives a client request, processes it using the appropriate service handler, and creates the initial transaction with its identifier N_{id}. This node is responsible for broadcasting the transaction to other nodes in the network to begin the consensus process. The originator's node ID is included in the transaction to maintain provenance and accountability in the network.

*6) **BFT Consensus**:* Byzantine Fault Tolerant (BFT) Consensus is a distributed agreement process that can tolerate Byzantine faults—nodes that may behave arbitrarily or maliciously. In our system, BFT consensus requires a minimum of 4 nodes and achieves agreement when at least $\lceil \frac{2n+1}{3} \rceil$ nodes (where n is the total number of nodes) validate a transaction. This ensures system correctness even if up to $\lfloor \frac{n-1}{3} \rfloor$ nodes are Byzantine. The BFT consensus layer provides strong finality guarantees but introduces latency proportional to network size.

*7) **Simulation Consensus**:* Simulation Consensus is a lightweight agreement process used in Layer 2 that prioritizes responsiveness over Byzantine fault tolerance. It mimics the validation logic of the main BFT consensus but operates with relaxed participation requirements. This approach enables interactive, low-latency responses during multi-step workflows while maintaining consistent rule application. Transactions

reaching simulation consensus are stored in a buffer and can later be committed to the BFT consensus layer as a batch, providing eventual Byzantine fault tolerance while significantly improving response times.

B. System Architecture

Figure 1 illustrates the architecture of our proposed system. The system employs a dual-layer approach, each consisting of interconnected nodes with specific responsibilities. As shown in the diagram, every node in both layers incorporates these core components:

- Consensus Engine: Manages all consensus-related functions, including transaction verification, BFT consensus execution, block commitment, and ledger storage. This component also handles peer-to-peer (P2P) communication with other nodes within the same layer.
- Web Server: Provides an HTTP interface exposing endpoints that enable clients to interact with the system through standardized APIs.
- Service Registry: As defined in III-A, this component maps API routes to their corresponding handlers.
- Database: Stores session data, transaction history, and application-specific information required for system operation.

These components work together to form a cohesive system that achieves both Byzantine fault tolerance and responsive performance.

C. Layer 1 (L1): Byzantine Fault-Tolerant Foundation

The first layer implements a design inspired by DeWS, where each node represents a domain operated by a distinct organization participating in the application ecosystem [12]. L1 serves as the primary foundation for BFT by implementing full BFT consensus protocols. Nodes in L1 maintain identical service registries and execute consensus only when receiving commit requests from L2 nodes. Each validator independently verifies transaction validity by applying the same service handler as the transaction originator and comparing response equivalence.

To satisfy BFT requirements, L1 must contain at least four nodes, which is the minimum necessary to tolerate one Byzantine fault, following the standard rule that a BFT system requires at least $3f + 1$ nodes to tolerate f Byzantine faults. In addition to the standard node components, L1 nodes maintain an immutable blockchain ledger, providing tamper-proof storage of confirmed transactions.

D. Layer 2 (L2): Interactive Transaction Layer

The second layer represents the core innovation of our architecture. L2 functions as a transaction buffer, performing simulation consensus and batching operations for efficient commitment to L1. Each node in L2 employs the same service registry and service handlers both in the web server and consensus components. Unlike L1, this layer is not required to fulfill the strict BFT node count requirements; instead, it inherits fault tolerance properties from the underlying L1 network.

The separation between layers enables L2 to prioritize fast responsiveness over native BFT, effectively optimizing system performance for interactive applications while still validating transactions according to system rules. Furthermore, L2 is designed to operate with a minimal number of nodes, allowing consensus simulation to complete faster and reducing the latency introduced during transaction processing.

Algorithm 1 Unified Consensus Process (Both L1 and L2)

```
procedure VALIDATETRANSACTION(transaction)
    Layer 1: Verify transaction is well-formed and properly signed
    Layer 2: Check transaction against session state and business rules
    Decrypt and parse transaction content
    if transaction format is invalid then
        return FALSE                          ▷ Malformed transaction
    end if
    if transaction violates state constraints then
        return FALSE                          ▷ Transaction is invalid
    end if
    Execute transaction against current state
    return TRUE                               ▷ Transaction is valid
end procedure
procedure PROCESSPROPOSAL(proposedBlock)
    Layer 1: Validators independently execute transactions and compare results
    Layer 2: Simulates validation by re-executing operations against local state
    if results from execution differ from proposed results then
        return REJECT     ▷ Byzantine behavior detected
    else
        return ACCEPT ▷ Proposed transactions is valid
    end if
end procedure
```

E. System Flow

Figure 2 illustrates the complete transaction lifecycle in our two-layer architecture. The process begins when a client sends a request to initiate a new session with a Layer 2 node. The receiving node, becoming the transaction originator, processes the request by using its service registry to identify the appropriate service handler. The handler processes the request locally and stores the session in the node's database. After processing, the service handler generates a response s, which together with the original request r and the node's identifier N_{id} forms a transaction $Tx(r, s, N_{id})$.

The transaction originator forwards this transaction to the L2 consensus layer, which performs simulation consensus by replicating the operation across other L2 nodes. As shown in Algorithm 1, each participating node processes the request through its own service registry and service handlers

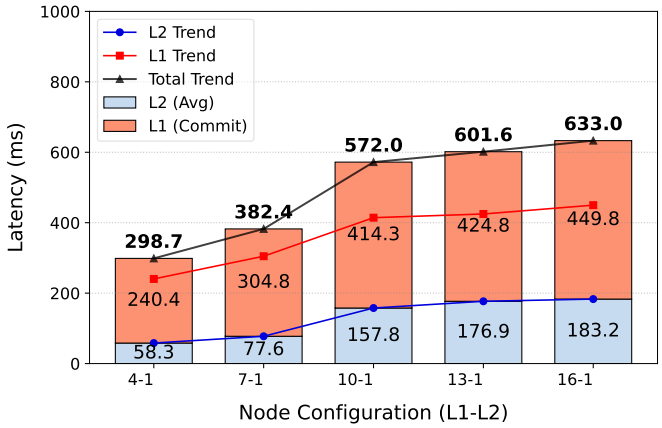

Fig. 3. Latency analysis of BFT node configurations with a single Layer 2 node, showing the trend of L2 operation latency, L1 commit latency, and total workflow latency across various L1 node configurations.

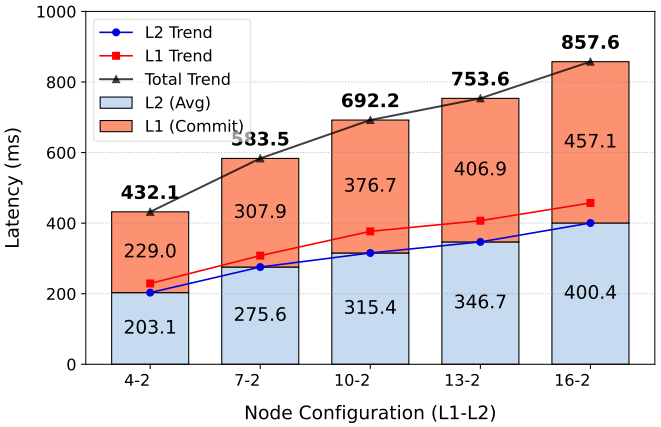

Fig. 4. Latency analysis of BFT node configurations with dual Layer 2 nodes, showing the trend of L2 operation latency, L1 commit latency, and total workflow latency across various L1 node configurations.

to verify response consistency. During the Process Proposal phase, nodes execute transactions independently and reject proposals if the results differ, detecting Byzantine behavior. If the deployment has only a single L2 node, it still executes consensus validation steps internally to ensure compliance with network rules.

After session creation, clients can execute a series of related operations within the same session context. Each subsequent request follows the same processing pattern as the initial session creation, with the important distinction that each operation is validated against the session's current state, following the validation logic in Algorithm 1.

The L1 layer operates a full BFT consensus mechanism. Similar to the L2 process, the transaction originator on L1 first replicates the complete session across validators. It generates a transaction containing all operations and submits it to the L1 consensus layer. The BFT consensus algorithm requires validation from at least $\lceil \frac{2n+1}{3} \rceil$ nodes to consider the transaction valid; otherwise, it is rejected.

Upon reaching consensus, the L1 network returns the result to the L2 node along with blockchain reference data such as block height and transaction hash. The L2 node then updates the session status accordingly, propagates this update to other L2 peers, and delivers the final result to the client. For verification purposes, clients can query transaction status either through an L2 node or directly from the L1 network, depending on the implementation.

This two-layer approach combines the immediate responsiveness of simulation consensus with the security guarantees of BFT consensus, creating a system that is both interactive and trustworthy.

IV. Performance Evaluation

Our proof-of-concept implementation demonstrates a supply chain management system that represents an ideal scenario where both high responsiveness and strong security guarantees are the required attributes. Supply chain operations demand fast and interactive processing to maintain throughput while simultaneously requiring tamper resistance to prevent fraud across organizational boundaries. As such, our use case demonstrates the need for session fault tolerance, step-by-step feedback without premature commitment, and a tamper-proof audit trail. The implemented workflow consists of five sequential stages: (1) session initiation, where an operator begins processing a package; (2) package scanning, which identifies and retrieves expected contents; (3) validation of the package's digital signature to authenticate its origin; (4) quality control inspection; and (5) shipping label generation and courier assignment. Once completed, the entire session can be committed as an atomic unit to the Byzantine fault-tolerant L1, demonstrating how our dual-consensus architecture effectively balances responsive user experience with strong security guarantees in multi-step business processes. It should be noted that while this workflow represents a simplified version of real-world supply chain operations, it captures the essential characteristics and challenges that our system is designed to address.

A. Proof-of-Concept Implementation

We utilized CometBFT as the consensus engine, providing customizable and robust BFT using the Tendermint consensus algorithm [17], [18] through its *Application BlockChain Interface* (ABCI). We implemented both the web server and service registry components in Go programming language to ensure optimal interoperability with CometBFT's native codebase.

Our primary goal is to enhance interactivity and responsiveness through the L2 implementation, so our evaluation focuses primarily on latency metrics. We define latency as the time it takes from request initiation to response reception, calculated as $t_{\text{res}} - t_{\text{req}}$, where t_{res} represents the response reception timestamp and t_{req} represents the request initiation timestamp.

We conducted comprehensive measurements for each step in the workflow sequence: from session creation through to

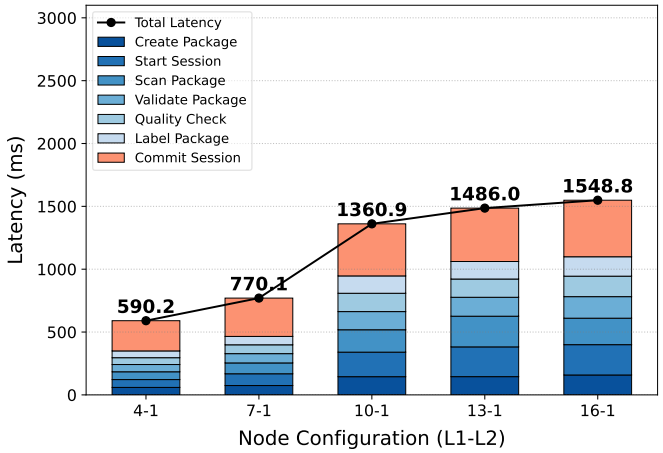

Fig. 5. Detailed endpoint latency breakdown for BFT configurations with a single L2 node, showing individual API endpoint contributions to total workflow latency.

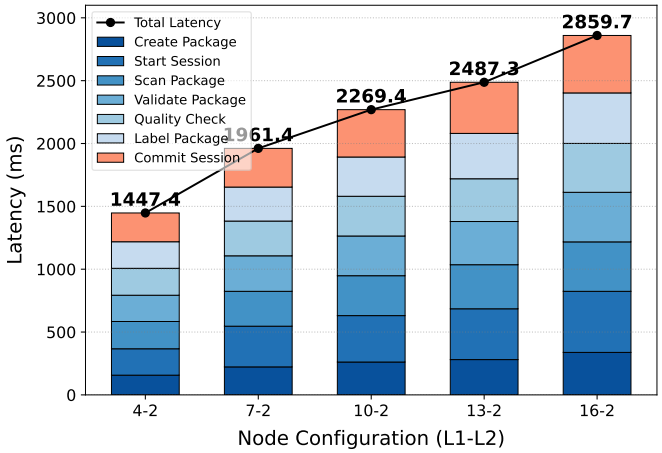

Fig. 6. Detailed endpoint latency breakdown for BFT configurations with dual L2 nodes, showing individual API endpoint contributions to total workflow latency.

blockchain commitment. To facilitate multiple test iterations, we implemented a package creation step that generates unique package identifiers for each test run. This approach enabled us to execute measurements systematically across varying node configurations. For each complete workflow, we performed 100 iterations to ensure statistical significance and reliability of our results.

For clarity in presenting our results, we use the notation $L1$-$L2$, where $L1$ represents the number of nodes in the BFT Layer 1 network and $L2$ represents the number of nodes in the simulation Layer 2 network. We systematically tested configurations with 1 and 2 L2 nodes while varying L1 nodes across five different cluster sizes: 4, 7, 10, 13, and 16. These specific node counts were selected to represent increasing Byzantine fault tolerance thresholds from 1 to 5, calculated according to the standard formula $n = 3f + 1$ [7], where n is the total number of nodes and f is the number of tolerable Byzantine faults.

The benchmarking framework was implemented as a standalone Go application that records performance metrics in CSV format. We then processed this data using Python to generate visualizations that highlight the performance characteristics across different system configurations.

B. Experimental Results

Our evaluation results demonstrate the significant performance advantages of the L2 session batching approach across various BFT cluster configurations.

For configurations with a single L2 node, we observed clear performance patterns as the number of L1 nodes increased. Figure 3 illustrates that L2 operations (Create Package, Start Session, Scan Package, Validate Package, Quality Check, and Label Package) maintained relatively low latencies even as the BFT cluster size grew. Specifically, with the 4-1 configuration, L2 operations averaged just 58.3ms, while the consensus-requiring Commit operation took 240.4ms, resulting in a total workflow latency of 298.7ms.

As the L1 BFT cluster expanded to 16 nodes, the average L2 operation latency increased to 183.2ms, while the Commit operation latency rose to 449.8ms, yielding a total workflow latency of 633.0ms. This demonstrates that while larger BFT clusters introduce increased latency, the interactive operations remain highly responsive, with L2 operations consistently completing in under 200ms, while the entire workflow still completes in under 700ms.

To better represent realistic consensus simulation, we tested configurations with two Layer 2 nodes. As shown in Figure 4, adding a second L2 node introduces additional coordination overhead but provides a more robust simulation environment. With the 4-2 configuration, L2 operations averaged 203.1ms, while the Commit operation took 229.0ms, resulting in a total workflow latency of 432.1ms.

When scaling to 16 L1 nodes (16-2 configuration), the average L2 operation latency increased to 400.4ms, with Commit operations requiring 457.1ms, bringing the total workflow latency to 857.6ms. These latencies are significantly higher than their single-L2-node counterparts, and notably, the L2 operation latency approaches the L1 commit operation latency, diminishing the supposed advantage of the two-layer architecture in this configuration.

Figure 5 provides a more detailed breakdown of individual endpoint latencies across all tested configurations with a single L2 node. The total workflow latency increased from 590.2ms with the 4-1 configuration to 1548.8ms with the 16-1 configuration. This increase is primarily attributable to the growing complexity of achieving consensus across larger BFT networks.

The endpoint-specific breakdown in Figure 6 shows that total workflow execution time ranges from 1447.4ms with the 4-2 configuration to 2829.3ms with the 16-2 configuration. This significant increase reflects the added complexity of coordinating two L2 nodes while simultaneously scaling the BFT L1 cluster.

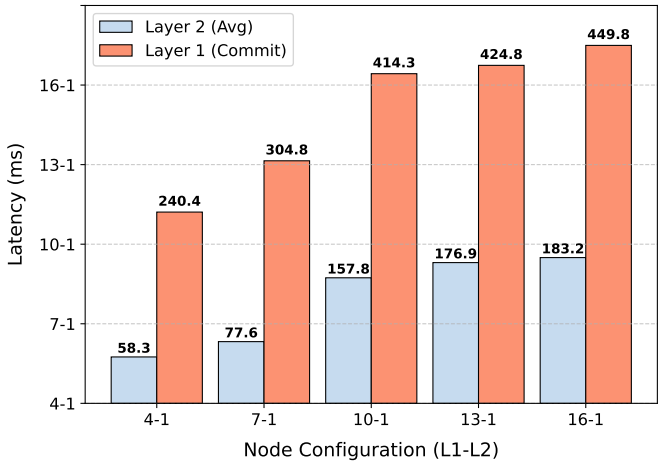

Fig. 7. Response time comparison between Layer 1 consensus commit operations and Layer 2 transaction simulation with L2=1 configurations, showing the significant decrease in latency at about 2.5-4.1× faster.

These results validate our two-layer approach: Layer 2 operations consistently achieve sub-second latencies suitable for interactive use, while the more expensive BFT consensus operations are consolidated into a single commit phase. Even with a 16-node BFT cluster, the interactive portions of the workflow maintain responsive performance.

While the latencies we observed are higher than those reported for traditional *request-compute-response* operations in DeWS (approximately 16-18ms for basic POST/GET operations) [12], the increased responsiveness in complex multi-step workflows justifies this trade-off. Our approach provides significant benefits in terms of session management, state consistency, and interactive user experience across the two-layer architecture.

Our findings suggest that smaller L1 configurations with a single L2 node offer the best balance of performance and fault tolerance for most applications. However, for scenarios requiring higher Byzantine fault tolerance thresholds, configurations with 10 to 16 L1 nodes remain viable, with total workflow latencies still acceptable for supply chain and similar applications. As shown in Figure 7, the comparison between L1 and L2 operations for single L2 node configurations demonstrates a significant latency decrease, with L2 operations approximately 2.5 to 4.1 times faster than L1 commits. The dual L2 node configurations, while theoretically providing more accurate simulation, introduce significantly higher latencies without proportional improvements in L2 operation responsiveness. This suggests that for applications prioritizing performance, either a single L2 node or direct L1 operations would be preferable to a multi-node L2 setup, as the added complexity does not yield substantial benefits in our tested scenarios.

V. Conclusion

This paper presented a two-layer architecture with a session-aware transaction buffer that addressed the challenge of providing interactive experiences in Byzantine fault-tolerant (BFT) web services. Our approach decoupled latency-sensitive operations from BFT consensus processes, maintaining sub-200ms response times for interactive operations even with 16-node BFT clusters. By buffering operations and deferring consensus until session completion, we achieved both the responsiveness needed for modern applications and the security guarantees of Byzantine fault tolerance. Our approach opened possibilities for BFT systems in domains previously impractical due to latency constraints, particularly for multi-step workflows like supply chain management.

References

[1] S. Mystakidis, "Metaverse," *Encyclopedia*, vol. 2, no. 1, pp. 486–497, 2022. [Online]. Available: https://www.mdpi.com/2673-8392/2/1/31

[2] J. H. Rony, R. H. Khan, J. Miah, and M. M. Syeed, "E-Commerce Application in Metaverse: Requirements, Integration, Economics and Future Trends," in *2024 IEEE CONECCT*, Jul. 2024, pp. 1–6.

[3] M. Hatami, Q. Qu, Y. Chen, H. Kholidy, E. Blasch, and E. Ardiles-Cruz, "A Survey of the Real-Time Metaverse: Challenges and Opportunities," *Future Internet*, vol. 16, no. 10, p. 379, Oct. 2024, number: 10 Publisher: Multidisciplinary Digital Publishing Institute.

[4] S. Yang and M. Li, "Web3.0 Data Infrastructure: Challenges and Opportunities," *IEEE Network*, vol. 37, no. 1, pp. 4–5, Jan. 2023.

[5] A. S. Rajawat, S. Goyal, A. Goyal, K. Rajawat, M. S. Raboaca, C. Verma, and T. C. Mihaltan, "Enhancing Security and Scalability of Metaverse with Blockchain-based Consensus Mechanisms," in *2023 15th Int. Conf. Electron., Comput. Artif. Intell. (ECAI)*, Jun. 2023, pp. 01–06.

[6] S. Isukapalli and S. N. Srirama, "A systematic survey on fault-tolerant solutions for distributed data analytics: Taxonomy, comparison, and future directions," *Computer Science Review*, vol. 53, p. 100660, Aug. 2024.

[7] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[8] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[9] M. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan, "Thema: Byzantine-fault-tolerant middleware for Web-service applications," in *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, Oct. 2005, pp. 131–140, iSSN: 1060-9857.

[10] W. Zhao, "BFT-WS: A Byzantine Fault Tolerance Framework for Web Services," in *2007 Eleventh International IEEE EDOC Conference Workshop*, Oct. 2007, pp. 89–96.

[11] C. Berger and H. P. Reiser, "Webbft: Byzantine fault tolerance for resilient interactive web applications," in *Distributed Applications and Interoperable Systems*, S. Bonomi and E. Rivière, Eds. Cham: Springer International Publishing, 2018, pp. 1–17.

[12] G. S. Ramachandran, T. T. L. Tran, and R. Jurdak, "DeWS: Decentralized and Byzantine Fault-tolerant Web Services," in *2023 IEEE ICBC*. Dubai, United Arab Emirates: IEEE, May 2023, pp. 1–9.

[13] R. Kotla, A. Clement, E. Wong, L. Alvisi, and M. Dahlin, "Zyzzyva: Speculative Byzantine Fault Tolerance – Communications of the ACM," Nov. 2008.

[14] M. Mandal, M. S. Chishti, and A. Banerjee, "Investigating Layer-2 Scalability Solutions for Blockchain Applications," in *2023 IEEE HPCC/DSS/SmartCity/DependSys*, Dec. 2023, pp. 710–717.

[15] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain Scaling Using Rollups: A Comprehensive Survey," *IEEE Access*, vol. 10, pp. 93 039–93 054, 2022.

[16] S. Calzavara, H. Jonker, B. Krumnow, and A. Rabitti, "Measuring Web Session Security at Scale," *Computers & Security*, vol. 111, p. 102472, Dec. 2021.

[17] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," Nov. 2019, arXiv:1807.04938 [cs].

[18] E. Buchman, R. Guerraoui, J. Komatovic, Z. Milosevic, D.-A. Seredinschi, and J. Widder, "Revisiting Tendermint: Design Tradeoffs, Accountability, and Practical Use," in *2022 52nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. – Suppl. Vol. (DSN-S)*, Jun. 2022, pp. 11–14.