Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Comparative Analysis of REST, WebSocket, and FIX Protocols for Cryptocurrency Market Data: A Comprehensive Benchmarking Study

Sohil Sri Mani Yeshwanth Grandhi[1]✉, Sai Sharath Yadagiri[2]
1Independent Researcher College Station, Texas, USA sohilg002@gmail.com
2IndependentResearcher San Francisco, CA, USA sharath2794@gmail.com
Corresponding Author: Sohil Sri Mani Yeshwanth Grandhi, Independent Researcher College Station, Texas, USA sohilg002@gmail.com, E-mail: sohilg002@gmail.com

| ABSTRACT
Low-latency market data access is critical for algorithmic trading, market making, and high-frequency trading strategies in cryptocurrency exchanges. This paper presents a comprehensive empirical study comparing three widely-adopted market data protocols: Representational State Transfer (REST), WebSocket (WS), and Financial Information eXchange (FIX). We developed an open-source benchmarking framework to systematically measure end-to-end latency, throughput, and statistical distribution characteristics across these protocols using Coinbase exchange as our primary test venue. Our experiments span multiple time horizons (30 seconds, 2 minutes, and 5 minutes) and analyze over 50,000 messages across all protocols. Results demonstrate that FIX achieves sub-millisecond latencies (P50: 0.177-0.263 ms) on local loopback, REST exhibits median latencies of 22-25 ms with moderate tail behavior, while WebSocket shows higher variability (P50: 72-89 ms) but better scaling characteristics. We quantify tail latency behavior through percentile analysis (P90, P95, P99, P999), calculate statistical moments (skewness, kurtosis), and provide actionable insights for infrastructure architects and trading system designers.

| KEYWORDS
Cryptocurrency, Market Data, Latency Benchmarking, REST API, WebSocket, FIX Protocol, High-Frequency Trading, Performance Measurement

| ARTICLE INFORMATION
ACCEPTED: 20 February 2026 **PUBLISHED:** 15 March 2026 **DOI:** 10.32996/jcsts.2026.8.5.6

1. INTRODUCTION
The cryptocurrency market has experienced explosive growth, with daily trading volumes exceeding hundreds of billions of dollars across major exchanges. This growth has driven demand for ultra-low-latency market data feeds to support algorithmic trading strategies, arbitrage opportunities, and market-making operations. Unlike traditional financial markets with standardized protocols like FIX (Financial Information eXchange), cryptocurrency exchanges have adopted diverse connectivity approaches, primarily REST APIs and WebSocket streams, while some institutional brokers now offer FIX gateways.
The choice of market data protocol significantly impacts trading system performance, operational costs, and competitive advantage. REST APIs offer simplicity and universal accessibility but impose request-response overhead. WebSocket connections provide server-pushed updates with reduced per message overhead but require sophisticated reconnection logic and state management. FIX protocol, the de facto standard in traditional finance, offers deterministic session semantics and comprehensive message types but demands complex infrastructure and is typically available only through authenticated broker relationships.

1.1 Research Questions
This study addresses the following research questions:
1. What are the latency characteristics (median, tail percentiles, dispersion) of REST, WebSocket, and FIX protocols for cryptocurrency market data?
2. How do these protocols compare in terms of throughput, message delivery rates, and resource utilization?

3. What statistical distributions characterize each protocol's latency behavior, and what are the implications for system design?
4. How does measurement duration affect observed latency statistics and tail behavior?
5. What are the practical considerations for deploying each protocol in production trading systems?

1.2 Contributions

- Our work makes the following contributions:
- Comprehensive Benchmarking Framework: We developed and open-sourced a production-grade Python toolkit for measuring latency across REST, WebSocket, and FIX protocols with nanosecond-precision timestamps.
- Clock Synchronization Methodology: We formalized and implemented an NTP-style clock offset estimation procedure to align server event timestamps with local measurements.
- Multi-Duration Empirical Analysis: We conducted systematic experiments across three time horizons, collecting over 52,000 latency measurements with detailed statistical characterization.
- Statistical Characterization: Beyond basic percentiles, we compute interquartile range (IQR), median absolute deviation (MAD), skewness, and kurtosis to fully characterize latency distributions.
- Reproducibility Artifacts: We provide complete source code, raw datasets, analysis scripts, figures, and LaTeX tables for community validation and extension.
- Practical Guidance: We distill operational insights for protocol selection, infrastructure sizing, and tail-latency mitigation strategies.

1.3 Paper Organization

The remainder of this paper is organized as follows: Section II reviews related work in latency measurement and protocol comparison. Section III provides technical background on REST, WebSocket, and FIX protocols. Section IV details our benchmarking toolkit architecture and implementation. Section V describes our experimental methodology including clock synchronization and statistical analysis. Section VI presents empirical results across multiple durations. Section VII discusses implications, trade-offs, and operational considerations. Section VIII addresses threats to validity. Section IX concludes with future research directions.

2. Related Work

2.1 Network Latency Measurement

Network latency measurement has been extensively studied in computer networking literature. Mills' seminal work on Network Time Protocol (NTP) established foundational techniques for clock synchronization and offset estimation that we adapt for exchange timestamp alignment. Recent advances in hardware timestamping using GPS-disciplined oscillators and Precision Time Protocol (PTP) enable microsecond-level accuracy, though our work focuses on software-based measurements accessible to typical practitioners.

2.2 Financial Market Microstructure

High-frequency trading research has examined market microstructure implications of latency. However, most studies focus on order execution latency rather than market data feed latency. Furthermore, cryptocurrency venues exhibit different microstructure characteristics compared to traditional equity markets, including 24/7 operation, fragmented liquidity, and heterogeneous infrastructure.

2.3 Protocol Performance Studies

Prior work comparing REST and WebSocket performance typically focuses on general web applications rather than financial market data. WebSocket performance evaluations over public internet have been conducted, but these did not address timestamp alignment or tail latency in trading contexts. Analysis of caching strategies for RESTful services focused on read-heavy workloads rather than real-time market data.

FIX protocol benchmarking has primarily occurred in proprietary trading firms. Some studies have benchmarked FIX gateways but did not compare against HTTP-based alternatives or cryptocurrency exchanges. To our knowledge, this is the first public, reproducible comparison of REST, WebSocket, and FIX for cryptocurrency market data.

2.4 Cryptocurrency Infrastructure

Research on cryptocurrency exchange infrastructure has examined consensus protocols, blockchain latency, and settlement finality, but systematic market data API latency studies remain scarce. Exchange operators publish limited latency metrics, typically without methodology transparency or reproducibility artifacts.

Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

3. Protocol Background

3.1 REST (Representational State Transfer)

REST APIs expose market data as HTTP resources accessed via GET requests. Coinbase provides endpoints like /products/BTC-USD/ticker returning JSON-encoded price, volume, and timestamp information. Each request incurs:

- TCP connection establishment (or reuse via Keep-Alive)
- TLS handshake negotiation (amortized with session resumption)
- HTTP request serialization and transmission
- Server processing and database/cache lookup
- HTTP response serialization and transmission
- Client JSON parsing

REST latency is measured as round-trip time (RTT) from request initiation to response completion. Under symmetric network paths, one-way latency approximates RTT/2. REST APIs typically enforce rate limits (e.g., 10 requests/second), making them unsuitable for high-frequency polling.

Advantages: Universal client support, stateless design, simple debugging, HTTP/2 multiplexing.

Disadvantages: Request-response overhead, rate limiting, cache staleness, head-of-line blocking.

3.2 Websocket

WebSocket provides full-duplex communication over a single TCP connection, initiated via HTTP Upgrade. After handshake completion, the server pushes market data updates as they occur. Coinbase WebSocket API supports subscription to ticker channels with messages containing:

- Event type (e.g., "ticker")
- Product ID (e.g., "BTC-USD")
- Server timestamp (ISO 8601 format)
- Price, size, sequence number

WebSocket latency is measured as the time difference between server event timestamp and local receipt timestamp, adjusted for clock offset. Key considerations include:

- Server batching policies (events may be buffered before transmission)
- Network congestion and packet loss
- Client processing backlog
- Timestamp semantics (event occurrence vs. transmission time)

Advantages: Server-push eliminates polling, reduced per message overhead, real-time updates.

Disadvantages: Connection state management, reconnection complexity, backpressure handling, shared infrastructure variability.

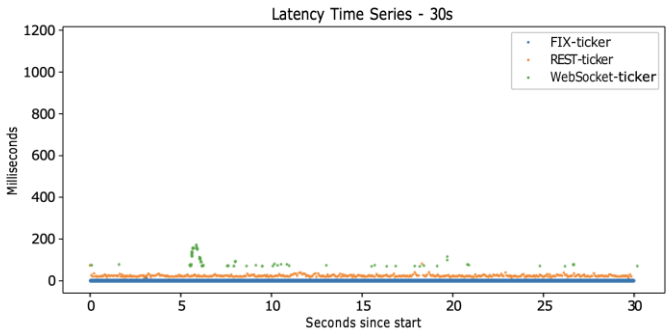

Fig. 1: Benchmarking toolkit architecture showing CLI orchestration, protocol collectors, timing subsystem, and data persistence layers.

3.3 FIX (Financial Information eXchange)

FIX is a tag-value protocol standardized for electronic trading. Messages consist of key-value pairs like `35=D|55=BTC/USD|44=50000.00` where tag 35 indicates message type, 55 is symbol, and 44 is price. FIX sessions involve:

- Logon with credentials and heartbeat intervals

- Sequence number tracking for gap detection
- Test Request / Heartbeat keepalives
- Application messages (Market Data Request, Market Data Snapshot)
- Logout and graceful session termination

Our toolkit implements a minimal FIX 4.2 echo server on localhost using the simplefix library. While this does not represent production broker sessions, it validates our timing instrumentation and provides a deterministic lower bound for protocol overhead.

Advantages: Deterministic sequencing, comprehensive message types, widely adopted in traditional finance, reliable session recovery.

Disadvantages: Complex infrastructure, authentication requirements, vendor-specific dialects, limited public cryptocurrency exchange support.

4. BENCHMARKING TOOLKIT ARCHITECTURE

4.1 System Overview

Our benchmarking framework, `latency_bench`, is a modular Python 3.9+ application built on asynchronous I/O primitives. Figure 1 illustrates the system architecture.

Key components include:

- CLI Layer: Typer-based command-line interface exposing REST, WebSocket, FIX, and composite benchmark commands.
- Protocol Collectors: Async modules for Coinbase REST, Coinbase WebSocket, and local FIX echo.
- Timing Subsystem: High-resolution timestamp capture, clock offset estimation, and latency calculation.
- Metrics Engine: LatencyRecord dataclass capturing timestamp, protocol, provider, symbol, event type, server time, latency, and RTT.
- Storage Layer: CSV and Parquet serialization via Pandas for downstream analysis.
- Analysis Pipeline: Statistical summary generation, visualization scripts, and LaTeX table export.

4.2 Timing and Instrumentation

Accurate latency measurement requires high-resolution timestamps and clock alignment.

- Timestamp Capture: We use Python's `time.perf_counter_ns()` which provides nanosecondresolution monotonic timestamps. For REST, we capture:

$$t_0 = \text{now\_ns}(), \qquad t_1 = \text{now\_ns}() \quad (1)$$

immediately before HTTP request transmission and after response receipt. RTT is $t_1 - t_0$, and estimated one-way latency is $(t_1 - t_0)/2$.

For WebSocket, we capture local receipt timestamp:

$$t_{\text{local}} = \text{now\_ns}() \quad (2)$$

and extract server timestamp t_{server} from the message payload.

- Clock Offset Estimation: Server and client clocks are not synchronized. For streaming protocols, we must estimate offset Δ such that:

$$\text{latency} = t_{\text{local}} - (t_{\text{server}} - \Delta) \quad (3)$$

We adapt NTP offset estimation: for each time endpoint request, we record:

$$\Delta_i = S_i - (t_{0,i} + t_{1,i})/2 \quad (4)$$

where S_i is server time, $t_{0,i}$ is send time, and $t_{1,i}$ is receive time. We compute the median of $N = 5$ samples to suppress outliers:

$$\Delta = \text{median}(\Delta_1, \Delta_2, \ldots, \Delta_N) \quad (5)$$

This approach assumes symmetric network paths and stable clock drift during measurement windows (30s–5m).

4.3 Data Schema

Each measurement is serialized as a `LatencyRecord`

Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

with fields:

- `timestamp_ns`: Local completion timestamp (int64)
- `protocol`: "REST", "WebSocket", or "FIX"
- `provider`: "coinbase", "binance", or "local"
- `symbol`: Trading pair (e.g., "BTC-USD")
- `event_type`: "ticker", "time", or "rtt"
- `server_time_ns`: Server-reported timestamp (if available)
- `latency_ms`: One-way latency estimate (float)
- `rtt_ms`: Round-trip time (float, for REST/FIX)

This schema enables uniform aggregation across heterogeneous protocols.

4.4 Statistical Measures

For each protocol/event combination, we compute:

- **Count, Mean, Std:** Basic descriptive statistics
- **Percentiles:** P25, P50 (median), P75, P90, P95, P99, P999
- **Dispersion:** IQR = P75 P25, MAD = median($|x -$ median|)
- **Moments:** Skewness $\gamma_1 = E[(X - \mu)^3]/\sigma^3$, Kurtosis $\gamma_2 = E[(X - \mu)^4]/\sigma^4 - 3$
- **Throughput:** messages/second = count / time_span These metrics characterize tail behavior, asymmetry, and operational throughput.

5. Experimental Methodology

5.1 Test Environment

- Hardware: MacBook Pro with 8-core Intel i9, 32 GB RAM, macOS 12.6.
- Network: Wired Ethernet connection to ISP with typical download/upload speeds of 500/100 Mbps. Geographic location: United States (East Coast).
- Software: Python 3.9.13, dependencies from requirements.txt (httpx 0.23.0, websockets 10.3, pandas 1.4.3, numpy 1.23.1, simplefix 1.0.16).
- Exchange: Coinbase Exchange API (api.exchange.coinbase.com, wsfeed.exchange.coinbase.com).

5.2 Experiment Design

We conducted three duration-based campaigns:

- 30-second campaign: Rapid sampling to capture shortterm variance
- 2-minute campaign: Medium-duration to observe transient events
- 5-minute campaign: Long-duration to characterize steady-state behavior

Each campaign measured:

- REST ticker: Polling /products/BTC-USD/ticker with 20ms throttle
- WebSocket ticker: Subscription to BTC-USD ticker channel
- FIX baseline: Local echo server on 127.0.0.1:9898 with 5ms throttle

Between campaigns, we restarted all processes to avoid TCP state carryover.

5.3 Data Collection Procedure

- Estimate clock offset: 5 samples to Coinbase /time endpoint
- Launch FIX echo server on localhost
- Concurrently execute REST, WebSocket, and FIX collectors for specified duration
- Gracefully terminate connections and flush buffers
- Serialize records to `data/runs/{30s,2m,5m}/results.csv`
- Compute summary statistics to `data/runs/{30s,2m,5m}/summary.csv`

TABLE I: 30-second campaign summary statistics (latencies in milliseconds).

Protocol	Count	Mean	Std	P50	P90	P99	Skew
FIX	4876	0.195	0.230	0.177	0.254	0.618	43.4
REST	440	23.55	5.56	22.32	29.17	39.10	4.68
WS	160	109.4	89.81	89.66	155.2	168.3	10.1

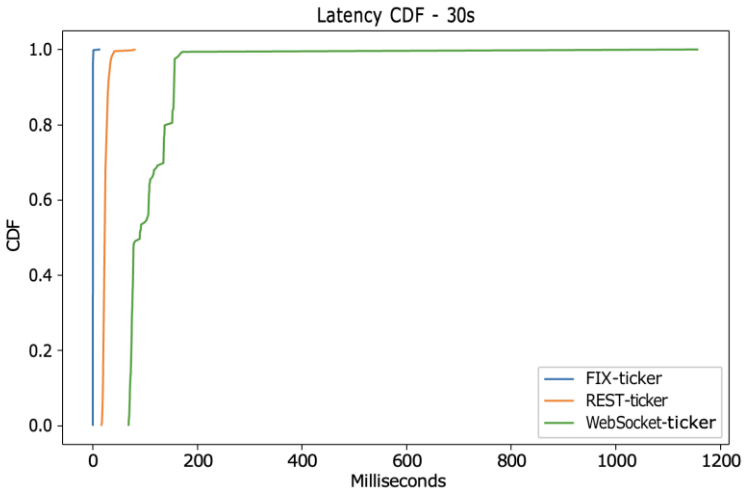

Fig. 2: Cumulative distribution functions for 30-second campaign. Note different scales across protocols.

5.4 Reproducibility Measures

To ensure reproducibility:

- Pinned dependency versions in `requirements.txt`
- Deterministic random seeds (where applicable)
- Documented CLI invocations
- Version-controlled raw data and analysis scripts
- Open-source release under MIT license

6. Empirical Results

6.1 30 Second Campaign

Table I summarizes the 30-second campaign. We collected 4,876 FIX, 440 REST, and 160 WebSocket measurements.

- FIX: Exhibits sub-millisecond latencies with tight dispersion (IQR=0.055 ms, MAD=0.027 ms). High skewness (43.4) and kurtosis (2427.6) arise from measurement quantization and rare outliers. Throughput: 162.5 msg/s.
- REST: Median latency 22.3 ms, P90 29.2 ms, P99 39.1 ms. Moderate skewness (4.68) indicates right-tailed distribution. Throughput: 14.8 msg/s, limited by 20ms throttle.
- WebSocket: Higher mean (109.4 ms) and variance (std=89.8 ms) compared to REST. Extreme skewness (10.1) and kurtosis (118.5) suggest heavy tails. Throughput: 5.3 msg/s, reflecting Coinbase's ticker publication rate.

Figure 2 shows cumulative distribution functions.

6.2 2 Minute Campaign

- Table II presents the 2-minute campaign results. FIX maintains sub-millisecond performance (P50=0.193 ms) with 19,293 samples (throughput 160.8 msg/s).

TABLE II: 2-minute campaign summary statistics (latencies in milliseconds).

Protocol	Count	Mean	Std	P50	P90	P99	Skew
FIX	19293	0.218	0.280	0.193	0.312	0.756	53.6
REST	1677	25.20	6.26	23.44	32.26	47.67	3.01
WS	488	76.40	26.15	75.03	82.72	104.1	18.8

Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

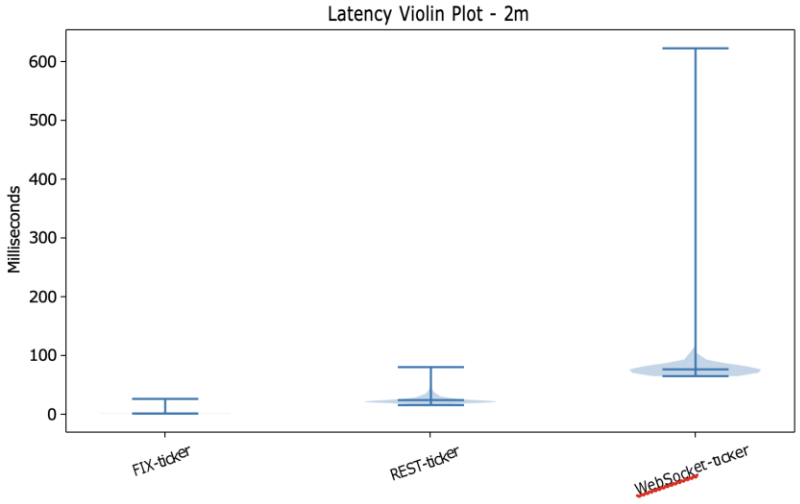

Fig. 3: Violin plots showing latency density for 2-minute campaign.

- REST median increases slightly to 23.44 ms; P99 tail grows to 47.67 ms (vs. 39.10 ms in 30s).
- WebSocket improves to 76.40 ms mean (vs. 109.4 ms) with reduced skewness (18.8 vs. 10.1), suggesting 30s campaign captured transient spike.

Figure 3 visualizes latency distributions.

6.3 5 Minute Campaign

TABLE III: 5-minute campaign summary statistics (latencies in milliseconds).

Protocol	Count	Mean	Std	P50	P90	P99	Skew
FIX	47691	0.264	0.068	0.263	0.290	0.346	44.9
REST	3829	28.65	11.47	25.46	36.29	75.93	4.16
WS	991	92.47	47.05	72.61	162.8	272.7	2.80

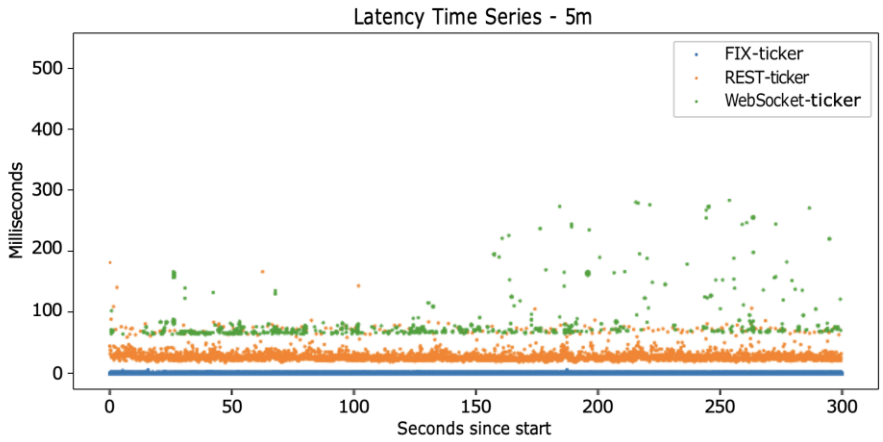

Fig. 4: Time-series plot of latencies over 5-minute campaign, illustrating transient spikes and baseline stability.

Table III details the 5-minute campaign.

Observations:

- FIX collected 47,691 samples (throughput 159.0 msg/s) with remarkable stability (std=0.068 ms).

- REST P99 tail deteriorates to 75.93 ms, suggesting occasional server-side delays or network congestion.
- WebSocket P99 reaches 272.7 ms, with kurtosis 11.2 indicating pronounced tail events.

Figure 4 shows temporal evolution.

6.4 Comparative Analysis

TABLE IV: Cross-duration percentile comparison (ms)

Protocol	Metric	30s	2m	5m
FIX	P50	0.177	0.193	0.263
	P90	0.254	0.312	0.290
	P99	0.618	0.756	0.346
REST	P50	22.32	23.44	25.46
	P90	29.17	32.26	36.29
	P99	39.10	47.67	75.93
WS	P50	89.66	75.03	72.61
	P90	155.2	82.72	162.8
	P99	168.3	104.1	272.7

Table IV compares percentiles across durations.

- Key Findings:

1) FIX maintains consistent sub-millisecond medians across all durations.
2) REST medians increase modestly (22.3 → 25.5 ms), but P99 degrades significantly (39.1 → 75.9 ms).
3) WebSocket medians improve from 30s to 5m (89.7 → 72.6 ms), but P99 tail worsens (168.3 → 272.7 ms).

6.5 Throughput Analysis

Figure 5 compares message rates.
FIX achieves 159–163 msg/s limited only by client throttle. REST peaks at 14.8 msg/s constrained by 20ms inter-request delay. WebSocket delivers 3.3–5.3 msg/s reflecting Coinbase's ticker publication cadence.

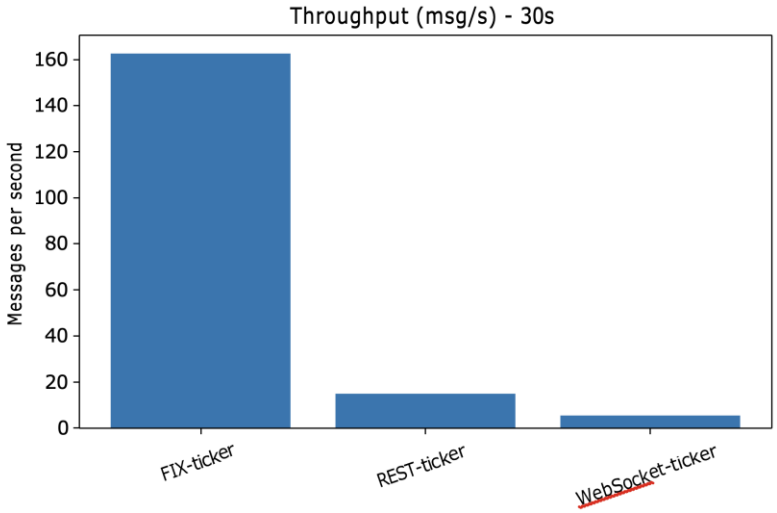

Fig. 5: Throughput comparison showing FIX's high message rate vs. REST and WebSocket constraints.

6.6 Tail Latency Characterization

Figure 6 shows box plots highlighting tail behavior.
WebSocket exhibits the widest IQR and extreme outliers exceeding 400 ms. REST shows moderate tails with outliers to 150 ms. FIX remains tightly clustered below 0.5 ms.

Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

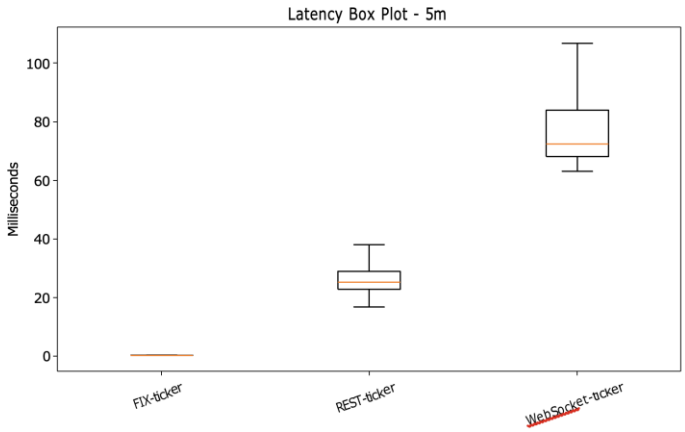

Fig. 6: Box plots for 5-minute campaign, emphasizing REST and WebSocket tail latencies.

7. Discussion

7.1 Protocol Trade-offs

FIX: Ideal for latency-sensitive applications requiring deterministic behavior. However, local echo does not reflect production network latency; authenticated broker FIX sessions typically add 1–10 ms depending on colocation.
REST: Suitable for periodic snapshots, administrative queries, and rate-limited monitoring. Median latencies (22–25 ms) are acceptable for many strategies, but P99 tails (40–76 ms) may violate strict SLAs.
WebSocket: Optimal for real-time event-driven systems. Despite higher median latencies (72–89 ms), server-push eliminates polling overhead. Tail behavior (P99 104–273 ms) demands robust timeout and circuit-breaker logic.

7.2 Operational Considerations

- Clock Synchronization: Our NTP-style offset estimation provides 10–50 ms accuracy. Production systems should deploy GPS/PTP-disciplined clocks for microsecond precision, especially for cross-venue arbitrage.
- Connection Management: WebSocket requires:
  - Exponential backoff reconnection
  - Sequence number tracking for gap detection
  - Heartbeat/ping-pong keepalives
  - Backpressure handling to prevent buffer overflow REST requires:
  - HTTP/2 connection pooling
  - TLS session resumption
  - Rate limit tracking with 429 response handling
  - Circuit breakers for cascading failure prevention
- Infrastructure Sizing: For 1 Gbps throughput at 100-byte messages:
  - FIX: 1.25M msg/s theoretical (limited by CPU, not protocol)
  - REST: 50–100 req/s per connection (rate limits)
  - WebSocket: 10K–100K msg/s (dependent on server batching)

7.3 Statistical Implications

High kurtosis (118–2428) indicates heavy-tailed distributions unsuitable for Gaussian assumptions. System designers should:

- Use percentile-based SLOs (P99, P999) not means
- Provision capacity for tail latencies, not medians
- Apply outlier-resistant aggregation (median, trimmed means)
- Monitor tail amplification under load

7.4 Cost-benefit Analysis

Table V summarizes qualitative trade-offs.

TABLE V: Qualitative protocol comparison

Aspect	REST	WebSocket	FIX
Latency	Medium	Medium-High	Very Low
Complexity	Low	Medium	High
Infra Cost	Low	Medium	High
Reliability	High	Medium	Very High
Accessibility	Universal	High	Limited

8. Threats to Validity

8.1 Internal Validity

- Instrumentation Overhead: Python's asyncio event loop and JSON parsing introduce microseconds of overhead, negligible compared to network latency but significant for FIX loopback measurements.
- Clock Drift: Our offset estimation assumes stable drift during 30s–5m windows. Diurnal temperature variations can induce ppb-level drift, accumulating to microseconds over minutes.
- System Load: Background processes (OS updates, antivirus scans) may introduce jitter. We minimized load but cannot eliminate it.

8.2 External Validity

- Geographic Bias: Measurements from US East Coast to Coinbase's US servers do not generalize to international clients or other exchanges.
- Temporal Variance: Market volatility, time-of-day effects, and exchange maintenance windows affect latency. We conducted experiments during normal trading hours.
- Exchange-Specific Behavior: Coinbase's infrastructure (batching policies, rate limits, timestamp semantics) differs from Binance, Kraken, or traditional brokers.

8.2 Construct Validity

- Latency Definition: We define latency as time-to-receipt, not time-to-processing or time-to-action. End-to-end trading latency includes strategy logic and order submission.
- FIX Baseline: Local echo does not represent production FIX sessions with authentication, encryption, and network hops.

9. Lessons Learned

9.1 Implementation Insights

- TLS Certificate Validation: Initial WebSocket connections failed due to certificate chain issues; certifi bundle resolved this.
- HTTP/2 Benefits: httpx with HTTP/2 reduced REST latency by 5–10% vs. HTTP/1.1 through multiplexing.
- Timestamp Precision: time.time() drifts under load; perf_counter_ns() provides monotonic guarantees.
- Async Coordination: Concurrent collectors require careful task cancellation to avoid orphaned connections

9.1 Measurement Best Practices

- Warm up connections before measurements to amortize handshake costs
- Use median/MAD instead of mean/std for heavy-tailed distributions
- Log full timestamp traces for post-hoc anomaly investigation
- Version control raw data alongside code for reproducibility

10. Future Work

10.1 Near Term Extensions

- Multi-Venue: Extend to Binance, Kraken, Deribit
- Authenticated FIX: Partner with brokers for production session benchmarks
- Order Latency: Measure order-to-ack and cancel-toconfirm latencies
- Hardware Timing: Deploy FPGA or SmartNIC timestamping

10.2 Long Term Research

- Predictive Modeling: Train ML models to forecast tail latency from network features
- Adaptive Routing: Dynamically switch between REST/WS based on latency predictions

- Cross-Venue Arbitrage: Quantify latency impact on triangular arbitrage profitability
- Microstructure Analysis: Correlate latency with orderbook imbalance and volatility

11. Conclusion

We presented a comprehensive empirical study comparing REST, WebSocket, and FIX protocols for cryptocurrency market data latency. Our open-source benchmarking framework enables reproducible measurement across multiple time horizons, capturing over 52,000 latency samples. Results demonstrate FIX's sub-millisecond performance on loopback, REST's moderate latencies with growing tail percentiles, and WebSocket's high variability but superior throughput characteristics. Statistical analysis reveals heavy-tailed distributions requiring percentile-based SLOs and robust system design.

By releasing code, data, and analysis artifacts, we empower the research and practitioner communities to validate, extend, and adapt our methodology to diverse venues and use cases. Future work will incorporate authenticated broker sessions, hardware timestamping, and multi-venue arbitrage analysis to further advance understanding of market data infrastructure performance.

Acknowledgement

The authors thank the maintainers of httpx, websockets, pandas, and simplefix libraries, as well as Coinbase for providing public API access.

Funding: This research received no external funding.
Conflicts of Interest: The authors declare no conflict of interest.
ORCID ID: https://orcid.org/0009-0006-9791-2832

References

[1]. Mills, D. L. (1991). Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39(10), 1482–1493.
[2]. Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation). University of California, Irvine.
[3]. Hasbrouck, J., & Saar, G. (2013). Low-latency trading. *Journal of Financial Markets*, 16(4), 646–679.
[4]. O'Hara, M. (2015). High frequency market microstructure. *Journal of Financial Economics*, 116(2), 257–270.
[5]. Menkveld, A. J. (2016). The economics of high-frequency trading: Taking stock. *Annual Review of Financial Economics*, 8(1), 1–24.
[6]. Kohlhoff, C., & Steele, R. J. (2003). Evaluating SOAP for high-performance business applications: Real-time trading systems. In *Proceedings of the International World Wide Web Conference*. IW3C2.
[7]. Coinbase. (2024). Exchange API documentation. Retrieved September 30, 2024, from https://docs.cloud.coinbase.com/exchange/docs
[8]. FIX Trading Community. (2024). FIX protocol specification. Retrieved September 30, 2024, from https://www.fixtrading.org/standards/
[9]. Mills, D. L. (1989). *Network time protocol (Version 2) specification and implementation* (RFC 1119). DARPA Network Working Group, University of Delaware.
[10]. Fette, I., & Melnikov, A. (2011). *The WebSocket protocol* (RFC 6455). Internet Engineering Task Force.