Streaming the Agentic Economy

A WebSocket-Native x402 Facilitator Architecture for Data Providers

Manas Mudbari
manas.mudbari@turboline.ai

Chandan Bhagat
chandan.bhagat@turboline.ai

April 2026

Version: 1.0
Related work: Mudbari & Bhagat (2026), "Adaptive Memory for LLM-Based Time Series Analysis," https://doi.org/10.31224/6603

Abstract

The x402 protocol provides a native HTTP payment primitive for AI agents consuming digital resources: a client requests a gated endpoint, receives a payment instruction, pays in USDC, and retries with proof. Over 100 million payments have been processed since the protocol launched in May 2025, with weekly transaction volume growing 492% through December 2025. This adoption confirms that agents are operational economic participants at scale. However, the x402 protocol is designed for HTTP request-response and has no mechanism for gating WebSocket streaming connections. The majority of high-value data that agents require, including blockchain event streams, order book feeds, and real-time price signals, agents receive over persistent WebSocket sessions, not discrete HTTP calls. This paper defines the structural boundary between the two protocols, maps the revenue loss it creates for data providers, and presents a WebSocket-native x402 facilitator architecture that resolves the gap through a pre-auth HTTP 402 handshake, JWT session token issuance, proxy-layer balance metering via Redis, and graceful `CLOSE` frame termination. The architecture is fully x402-compliant and requires no protocol modification.

Contents

1. Executive Summary

The economy is acquiring a new class of participant. AI agents are moving beyond passive tool use and into active economic roles: they consume APIs, pay for data, allocate compute resources, and make decisions autonomously on behalf of their operators. The infrastructure layer serving these agents must be designed for machines. It cannot require human account registration, monthly subscription commitments, or credit card billing flows. It must accept payment at the moment of consumption, at the granularity of individual resource accesses, with no human in the loop. The x402 protocol, an HTTP-native payment standard developed by Coinbase and now governed by the x402 Foundation, provides this primitive for HTTP request-response interactions. It has processed over 100 million payments since its launch in May 2025, with weekly transaction volume growing 492% through December 2025 [1]. The agentic economy is not a forecast. It is operational infrastructure, and it is scaling.

The x402 protocol does not address WebSocket streaming connections. This is not a minor omission. The majority of high-value data that AI agents require, including blockchain event streams, order book feeds, real-time price signals, and continuous market data, agents receive over persistent WebSocket connections, not discrete HTTP calls. WebSocket and HTTP are structurally different protocols. Once a WebSocket connection upgrades from HTTP, the HTTP semantics that x402 depends upon cease to apply. HTTP 402 cannot be issued inside an open WebSocket session. There is no x402 mechanism for gating a WebSocket upgrade, metering balance across a persistent connection, or gracefully terminating a stream when a session's USDC balance reaches zero. Data providers operating WebSocket endpoints face three outcomes today: they require human account registration that autonomous agents cannot complete, they offer subscriptions that agents cannot commit to, or they open feeds without authentication and earn nothing from agent consumption. In all three cases, providers either block agent-driven consumption or serve it for free.

This paper presents a formal analysis of the streaming gap, quantifies its revenue cost to data providers, and describes a WebSocket-native x402 facilitator architecture that resolves the gap without modifying the x402 protocol. The architecture uses a standard x402-compliant HTTP 402 handshake before the WebSocket upgrade to issue a signed JSON Web Token (JWT) session token, then proxies authenticated WebSocket connections with atomic balance metering via Redis and graceful `CLOSE` frame termination on balance exhaustion. We present this architecture as one component of a streaming infrastructure layer for the agentic economy. The following sections establish the technical and commercial case for deploying it.

2. The Agentic Data Economy

The economic role of software has shifted. For most of the history of networked computing, software was a passive instrument: a human issued instructions, software executed them, and a human interpreted the result. APIs were interfaces for humans writing code. Subscriptions were contracts between a business and a human accountable for the commitment. The entire billing apparatus of the internet, including credit cards, monthly statements, and account verification flows, was designed around the assumption that a person sat at the other end of every transaction.

That assumption no longer holds. AI agents are now active economic participants. They do not wait for human instruction on each step of a task. They receive a goal, decompose it into subproblems, identify the data and compute resources required, acquire those resources programmatically, and act on the results. An agent managing a DeFi position does not ask a human which price feed to query; it queries the feed directly. An agent executing a multi-step research task does not ask a human to purchase the relevant data; it pays for the data at the moment of consumption. The distinction between a software tool and an economic actor has collapsed, and the infrastructure layer that serves these agents must be built accordingly.

The data appetite of AI agents is specific and demanding. Agents operating in financial markets require continuous on-chain event streams: block confirmations, smart contract state changes, token transfer events, liquidity pool updates. Agents participating in trading or market-making require order book feeds that update in milliseconds and cannot tolerate the latency of polling an HTTP endpoint. Agents monitoring macro conditions require news feeds, sentiment streams, and macro data pipelines delivered as they arrive. None of these data types are static. None can be batched and delivered at the end of a day. They are live, continuous, and time-sensitive, and the transport mechanism that best serves continuous data delivery is the persistent WebSocket connection, not the discrete HTTP request.

The billing infrastructure built for human developers is structurally incompatible with this consumption model. Subscription tiers assume predictable monthly usage by an accountable human who can commit to a plan, respond to billing notifications, and manage renewals. An AI agent has none of these capabilities natively. It has no email address for account verification. It has no credit card in its runtime context. It cannot navigate a human-facing onboarding flow to obtain an API key. When an agent's pre-purchased API credits expire mid-task, the agent does not escalate to a billing portal; it fails silently, errors out, or produces degraded results that the human operator may not discover until downstream. The failure is not a billing dispute. It is a data access gap that the current infrastructure was not designed to detect or recover from.

The x402 protocol provides a direct solution to the HTTP request-response component of this problem. An agent with a funded wallet can pay for an API call autonomously, without human intervention, at the moment of consumption, at the granularity of a single request [2, 3]. The protocol eliminates the account registration requirement, the subscription commitment, and the

billing cycle entirely for HTTP-based data access. Adoption confirms that agents are using this mechanism at scale: the x402 protocol processed over 100 million payments in its first six months of operation, with weekly transaction volume growing 492% through December 2025, reaching approximately 156,000 weekly transactions [1]. These are not human developers making exploratory API calls. This is automated agent infrastructure consuming data programmatically and paying per request.

The market context for this activity is large and accelerating. Galaxy Research projected in January 2026 that agentic commerce will represent $3 to $5 trillion in B2C revenue by 2030, with blockchain infrastructure serving as the settlement and verification layer operating beneath mainstream applications [4, 5]. Data providers who build for agent consumption now are not building for a niche; they are building the foundational data infrastructure for a category of economic activity that is already underway and is growing faster than any billing infrastructure built to serve it.

The x402 protocol addresses the payment primitive problem for HTTP request-response interactions. It does not address the WebSocket streaming layer. This distinction is not minor. The majority of high-value, time-sensitive data that AI agents require arrives over persistent WebSocket connections: blockchain event streams, order book feeds, real-time price signals, and continuous market data. The agents consuming this data are willing and able to pay for it, as the HTTP x402 adoption numbers confirm. The infrastructure to accept that payment at the WebSocket layer does not yet exist. The following section defines the boundaries of this gap with technical precision.

3. The Streaming Gap

3.1 How x402 Works

The x402 protocol repurposes an HTTP status code that has existed in the specification since 1996 but was never implemented at scale. HTTP 402, "Payment Required," was reserved by the original HTTP/1.1 authors for future use. Coinbase formalized a complete payment protocol around this status code in May 2025, and the x402 Foundation, with membership from Coinbase, Cloudflare, Google, and Visa, now governs the specification [5].

The protocol flow proceeds as follows. A client, typically an AI agent operating autonomously, sends an HTTP GET or POST request to a protected resource endpoint. The resource server returns HTTP 402 with structured payment instructions in response headers, specifying the required amount in USDC, the recipient wallet address, the settlement blockchain, and the URL of a facilitator service authorized to verify payment. The client constructs a signed USDC payment payload and retries the original HTTP request, this time including the payment proof in the `X-PAYMENT` header. A facilitator service verifies that the on-chain settlement has occurred and signals authorization to the resource server. The server grants access and includes a payment confirmation header in its response [2, 3].

This flow is stateless, synchronous, and semantically complete within a single HTTP request-response cycle. The client sends a request. The server responds with a price. The client pays. The client re-requests with proof. The server delivers. Each step maps cleanly to the request-response model that HTTP was designed to support. The V2 specification extends this with a session token that allows clients to reuse a single payment proof for multiple subsequent requests to the same resource, reducing on-chain transaction frequency for repeat access [1]. In V2, modernized header names, specifically `PAYMENT-SIGNATURE`, `PAYMENT-REQUIRED`, and `PAYMENT-RESPONSE`, standardize the payment envelope across implementations [1].

The facilitator plays a specific and bounded role in this flow. It abstracts blockchain complexity away from the resource server, handling transaction submission, gas management, chain routing, and settlement confirmation [3]. The resource server never touches the blockchain directly. It issues payment instructions and accepts verification signals. This separation of concerns is a core design feature of the protocol and is directly relevant to the facilitator architecture presented in Section 5.

3.2 How WebSocket Connections Work

The WebSocket protocol, standardized in RFC 6455 [6], establishes a fundamentally different communication model from HTTP. Where HTTP is stateless and request-response, WebSocket is stateful, persistent, and bidirectional. Understanding the lifecycle of a WebSocket connection is prerequisite to understanding why x402 cannot address it.

A WebSocket connection begins with an HTTP upgrade handshake. The client sends a standard HTTP GET request with two additional headers: `Upgrade: websocket` and `Connection: Upgrade`, along with a `Sec-WebSocket-Key` used for the handshake challenge. The server responds with HTTP 101 Switching Protocols, completing the upgrade. At this point, the upgrade discards the HTTP layer. The connection transitions to the WebSocket OPEN state: a persistent, full-duplex TCP connection over which both parties can send and receive messages at any time.

In the OPEN state, data flows continuously in both directions without any request-response structure. The server pushes blockchain events, order book updates, market signals, or other time-series data frames to the client as they occur. The client may send subscription messages, filter updates, or acknowledgements. Neither party initiates a new "request" in the HTTP sense. The connection remains in the OPEN state until either party transmits a CLOSE frame, at which point a closing handshake completes and the TCP connection terminates.

This lifecycle has a critical implication: the HTTP semantics that x402 relies upon have no valid insertion point inside an open WebSocket session. HTTP 402 is an HTTP response status code. It exists on the HTTP layer, which is discarded after the upgrade handshake completes. An HTTP 402 response cannot be sent to a client inside an open WebSocket session because the session is no longer operating on HTTP. The only way to communicate payment requirements over an open WebSocket connection is through WebSocket application-layer messages or WebSocket close codes,

neither of which the x402 protocol specifies or addresses.

This is not a gap that future versions of x402 can close by adding a new HTTP header. The gap is structural. It follows from the protocol boundary between HTTP and WebSocket.

3.3 The Formal Gap

Table 1 maps the capabilities of the x402 protocol as specified against the requirements of WebSocket streaming access.

Table 1: x402 Protocol Capability Gap Analysis

x402 protocol handles	**x402 protocol does not handle**
HTTP GET and POST to protected endpoints	WebSocket upgrade payment gating
Per-request stablecoin micropayments in USDC	Per-stream or per-minute billing inside a persistent connection
Stateless payment verification via facilitator	Balance metering across an open session
Session tokens for repeated HTTP access (V2)	Graceful stream termination on balance exhaustion
Facilitator-mediated on-chain settlement	Multi-stream session management
Automatic endpoint discovery (V2)	Payment renegotiation inside an open WebSocket connection

This gap is not a theoretical edge case. It is the operational reality that data providers encounter today when agents attempt to access their live data feeds.

Consider three scenarios that describe the current state for a data provider operating WebSocket endpoints, such as QuickNode or GetBlock, when an AI agent attempts access.

In Scenario A, the provider requires an API key to authenticate WebSocket connections, and that API key is issued through a human account registration flow: email verification, payment method on file, plan selection. The AI agent operating autonomously cannot complete this flow. It has no email address, no credit card, and no capacity to navigate a human-facing onboarding process. The WebSocket feed is effectively inaccessible to the agent regardless of the agent's ability and willingness to pay.

In Scenario B, the provider offers a tiered subscription and requires a monthly or annual commitment for WebSocket access above a free tier. The AI agent has no mechanism to authorize a monthly subscription, manage renewal, or respond to billing notifications. Its runtime has no persistent identity in the provider's billing system. If an agent is deployed, uses data, and is then redeployed from a different environment, the provider's subscription system has no way to connect the two sessions. Feed access requires human intervention at every account lifecycle event.

In Scenario C, the provider has recognized that subscription friction prevents developer adoption and has opened the WebSocket feed without authentication. Developers can connect freely to explore the data before committing to a paid tier. AI agents can connect freely by the same mechanism. The provider has resolved the access problem by eliminating the revenue model entirely. The provider earns nothing from agent consumption of its WebSocket feeds.

In all three scenarios, the provider either loses the agent customer entirely or serves the agent for free. There is no current mechanism that allows a data provider to monetize autonomous agent consumption of WebSocket feeds at the granularity of individual streams or sessions. This is the streaming gap.

3.4 Why This Matters Now

The x402 protocol processed over 100 million payments in its first six months of operation, with transaction volume growing 492% through December 2025, reaching approximately 156,000 weekly transactions [1]. These numbers represent agent-driven activity. Human users do not issue 156,000 payments per week through a raw HTTP payment protocol. The agents executing these payments are also consuming data feeds. A meaningful fraction of that consumption occurs over WebSocket connections, and providers monetize none of it at the streaming layer.

The infrastructure gap is becoming economically material precisely as the market for agent data consumption is growing fastest. Galaxy Research projected in January 2026 that agentic commerce will represent $3 to $5 trillion in B2C revenue by 2030 [4, 5]. Data providers are positioned at the foundation of this economy. The agents that will drive that commerce require live, streaming data to operate. The providers who close the streaming monetization gap now establish infrastructure-level relationships with agent runtimes at the moment those relationships are being formed.

The facilitator architecture described in the following sections addresses the streaming gap directly, without modifying the x402 protocol, and without requiring data providers to build or operate blockchain infrastructure.

4. What Data Providers Lose

The streaming gap is not an abstract technical problem. It has a direct revenue consequence for every data provider that operates WebSocket endpoints and has seen autonomous agent traffic, or could see it if a payment mechanism existed. The revenue model mismatch runs deeper than a missing feature. Subscription tiers were designed for a specific type of customer: a human developer with a monthly budget, a predictable usage pattern, and the organizational capacity to sign up, commit to a plan, and manage the account over time. AI agents share none of these characteristics.

Agents consume in bursts. A market surveillance agent may open twenty simultaneous WebSocket connections for thirty minutes during a volatile trading session and require nothing for the next

four hours. A blockchain indexing agent may process millions of events in a single run and then be dormant until its operator deploys a new task. These consumption patterns do not fit monthly subscription buckets. They generate unpredictable peaks that subscription tiers either over-serve (charging the provider for capacity the agent never uses) or under-serve, throttling the agent at precisely the moment it most needs throughput. Neither outcome benefits the provider or the agent operator.

Agents also bring no commitment horizon. A human developer who purchases a subscription at the start of a project has an incentive to remain on the plan for the duration of that project. An AI agent runtime has no equivalent incentive structure. It accesses data when it needs it and stops when the task completes. The operator of a fleet of agents may have no advance knowledge of how many streams those agents will open in a given month. Subscription-based billing asks the operator to predict and commit in advance to something that is inherently variable.

The three dominant WebSocket data providers targeting blockchain developers illustrate the current state of agent billing precisely. QuickNode offers WebSocket access as part of its Scale plan, priced at $499 per month, with usage billed against a monthly credit allocation [7]. An AI agent that needs 10 WebSocket streams for 2 hours on a Tuesday cannot purchase 2 hours of QuickNode access. It must either operate under a human-held subscription, which misattributes cost and consumption, or trigger an account registration flow it cannot complete. GetBlock operates a subscription model with shared and dedicated node tiers, offering no per-stream or per-session billing mechanism for autonomous clients [8]. Allnodes, which provides validator hosting and node access for Solana and other chains, offers subscription-based access with no agent-native billing path [9]. None of these providers has built a billing model that accepts payment from an autonomous agent at the granularity of a single WebSocket session.

The cost model quantifies what providers are leaving uncaptured. Under a facilitator model priced at $1.00 per stream with a 1% facilitator cut and Base gas of $0.007 per settlement, a provider earns $0.983 per stream net. At 100 agent streams per month, provider net revenue is $98.30. At 1,000 streams per month, provider net is $983.00. At 10,000 streams per month, provider net reaches $9,830.00. Every one of these streams is currently generating zero dollars, because the payment infrastructure to collect that revenue does not exist at the WebSocket layer.

The agent consumer segment has properties that make it particularly attractive once the billing infrastructure exists. Agents do not churn the way human developers do. An AI agent that is configured by its operator to pay for a specific feed pays for that feed on every run, at exact consumption, with no evaluation overhead and no support ticket volume. When the balance is exhausted, the agent re-triggers the payment handshake and continues. There are no billing disputes because payment occurs at the moment of consumption with cryptographic settlement proof. There is no trial abuse because each stream is paid at the point of access. The incremental cost to the provider of serving an additional agent stream is the marginal infrastructure cost of the connection

itself, with no human customer success overhead attached.

The facilitator model creates a new revenue tier that operates beneath the existing subscription stack. It does not replace enterprise contracts or human developer plans. Enterprise customers who need dedicated infrastructure, SLA guarantees, and account management will continue on subscription arrangements. The facilitator captures the consumption below the subscription threshold: the burst agent workloads, the autonomous fleet deployments, and the programmatic data access that currently generates no revenue for the provider. At the scale that Galaxy Research projects for agentic commerce by 2030, totaling $3 to $5 trillion in B2C activity [4, 5], the data provider that has built this billing capability owns a structural position in the agent data supply chain.

5. Facilitator Architecture

5.1 Overview

The WebSocket-native x402 facilitator architecture addresses the streaming gap through a design that is fully compliant with the x402 protocol specification and requires no modification to the protocol itself. The architecture inserts a standard x402 HTTP 402 handshake before the WebSocket upgrade, converts the successful payment into a signed JWT session token, and then proxies authenticated WebSocket connections with stateful balance metering and graceful termination. The resource server continues to operate its WebSocket endpoint without direct blockchain exposure. The facilitator handles all settlement logic. The agent client gains autonomous access to streaming data feeds.

5.2 Four-Component Architecture

The four components operate as follows.

Agent client. The AI agent operating autonomously. It has a funded wallet, the ability to sign USDC payment payloads, and the ability to present a JWT in a WebSocket upgrade request. It does not interact with the blockchain directly during the streaming session.

Resource server. The data provider's WebSocket endpoint. It serves blockchain event streams, order book feeds, or other live data. It validates JWT session tokens before completing the WebSocket upgrade handshake. It has no blockchain infrastructure and requires no on-chain capability.

Facilitator. Our streaming infrastructure layer. It handles the x402-compliant payment handshake, issues signed JWT session tokens, proxies authenticated WebSocket connections, meters balance atomically via Redis, and sends graceful CLOSE frames when balance is exhausted. It settles USDC on-chain and relays confirmation to the resource server.

Blockchain settlement layer. Base or Solana, depending on provider preference. The facilitator submits and verifies USDC transfers. The agent's wallet funds the session deposit. The provider's

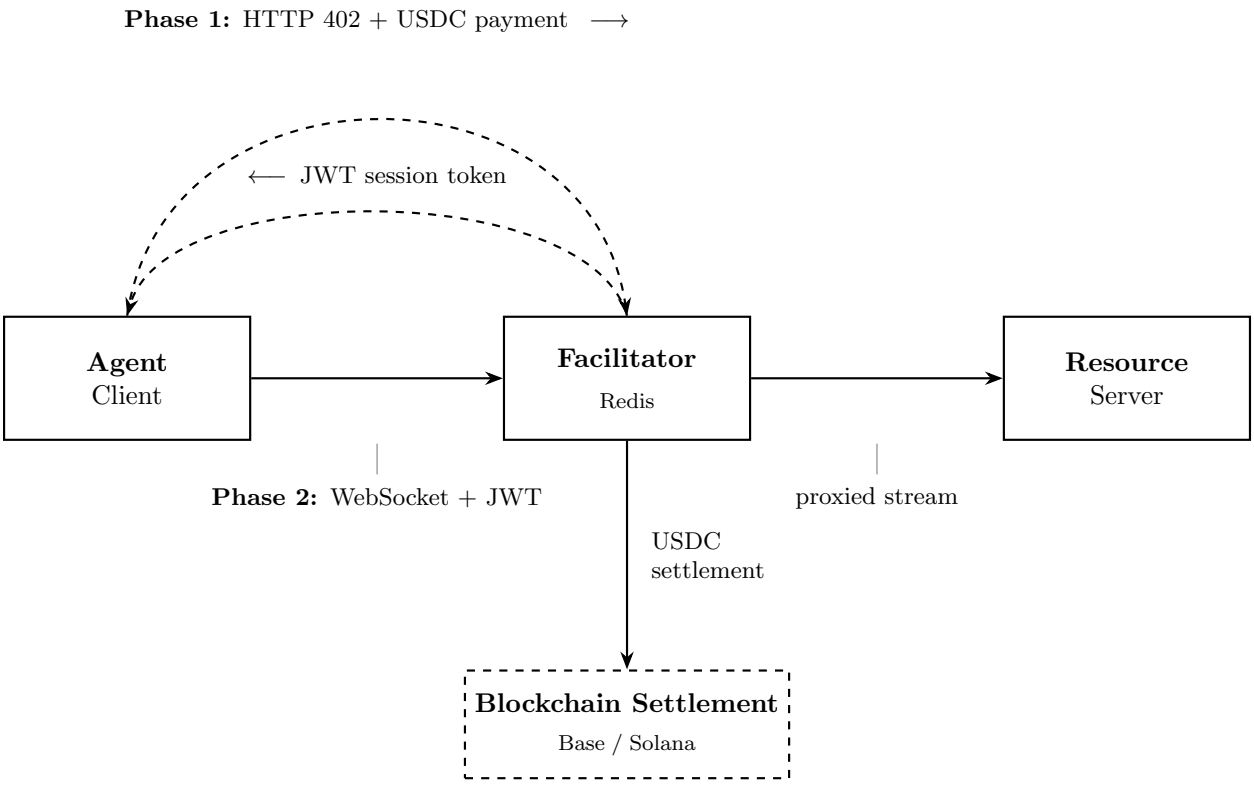

Figure 1: Four-component facilitator architecture. Phase 1 (dashed): x402 HTTP 402 handshake and JWT issuance. Phase 2 (solid): authenticated WebSocket proxy with Redis balance metering.

wallet receives net USDC after gas and facilitator fee.

The separation of concerns is deliberate. The resource server never touches the blockchain. The agent client never manages a persistent session state. The facilitator owns both the payment complexity and the session state, which is the appropriate allocation of responsibility in an infrastructure layer.

5.3 Phase 1: The Pre-Auth HTTP 402 Handshake

Before the WebSocket upgrade, the agent queries a designated HTTP endpoint on the resource server, or the facilitator acting as proxy, to initiate payment. This step is fully x402-compliant and uses the existing V2 protocol flow without modification [1].

The agent sends an HTTP GET to the pre-auth endpoint, specifying which WebSocket feed it intends to access. The server returns HTTP 402 with payment instructions: the USDC amount required for the session deposit, the recipient wallet address (the facilitator's settlement wallet), the settlement chain (Base or Solana), and the facilitator URL for verification [2, 3].

The agent constructs and signs a USDC payment payload for the specified deposit amount, for example $10.00 USDC covering 10 streams, and retries the HTTP request with the `X-PAYMENT` header containing the signed proof. The facilitator verifies the on-chain USDC transfer, confirming

settlement amount and recipient. On successful verification, the facilitator issues a signed JWT session token and returns it to the agent in the HTTP 200 response. This completes the x402 flow. The agent now holds a credential for WebSocket access.

5.4 JWT Session Token Structure

The session token is a JSON Web Token signed with the facilitator's private key. It contains the claims described in Table 2.

Table 2: JWT Session Token Claims

Field	Description
`sub`	Agent wallet address (subject identifier)
`feed`	Feed identifier specifying the authorized WebSocket endpoint
`deposited`	USDC amount deposited, e.g., "$10.00"
`streams_remaining`	Number of streams available for this session
`exp`	Expiry timestamp (Unix epoch); default 24 hours from issuance
`jti`	Unique token ID; used for replay protection
`iss`	Facilitator identifier
`chain`	Settlement chain (e.g., "base" or "solana")

The `jti` claim is a cryptographically random UUID generated at issuance time and stored in Redis with a 24-hour TTL. The token is opaque to the resource server, which validates the signature and claims without database access to the payment history.

5.5 Phase 2: The Authenticated WebSocket Upgrade

With a valid session token, the agent initiates a WebSocket upgrade to the provider's endpoint. The agent includes the JWT in the upgrade request, either as a query parameter (`?token=<jwt>`) or in the Authorization header (`Bearer <jwt>`). Both transport methods are supported by standard WebSocket client libraries.

The facilitator intercepts the upgrade request and performs three validations before completing the connection:

1. **Signature validation.** The facilitator verifies the JWT signature against its public key. An invalid signature returns HTTP 401.

2. **Claims validation.** The `exp` timestamp confirms the token has not expired. The `feed` claim confirms the token authorizes access to the requested endpoint. The `streams_remaining` field confirms at least one stream remains available. Any validation failure returns HTTP 403.

3. **Replay protection.** The facilitator queries Redis for the `jti` value. If the `jti` is not present in the active token registry, the facilitator rejects the request. Presenting a revoked or unknown `jti` returns HTTP 401.

On successful validation, the facilitator completes the WebSocket upgrade on behalf of the agent, opening a proxied connection to the upstream provider WebSocket endpoint. The agent receives the upgrade confirmation and begins receiving data. The resource server receives a validated, authenticated connection. Neither party is exposed to the full payment and session management logic.

5.6 Balance Metering Inside the Session

Once the WebSocket connection is open, the facilitator maintains balance state in Redis using an atomic counter keyed to the session token's `jti`. On each new stream that the agent opens within the session, the counter decrements by one. The decrement operation is atomic; concurrent stream opens cannot race to a negative balance.

When the `streams_remaining` counter reaches zero, the facilitator sends a WebSocket CLOSE frame to the agent with status code 4008 (a custom application-layer close code) and a human-readable reason string: "Session balance exhausted. Re-authorize via x402 to continue." The agent's runtime can parse this close code and automatically re-trigger the x402 pre-auth handshake to deposit additional USDC and obtain a new session token. The continuity of data access is preserved at the application layer without human intervention.

5.7 Gas Amortization via Session Deposit

The session deposit model reduces on-chain transaction costs for both agents and providers. Under single-settlement billing, each stream triggers one on-chain USDC transfer at a Base gas cost of $0.007. For a session covering 10 streams, total gas cost under single settlement is $0.070. Under the session deposit model, the agent submits one $10.00 USDC deposit covering all 10 streams, incurring one gas fee of $0.007. Gas cost per stream falls from $0.007 to $0.0007, a 90% reduction (see Appendix A, Analysis A.2).

The provider's net revenue improves accordingly. Under single settlement on Base, the provider receives $0.983 per stream. Under the session deposit model, the provider receives $0.9893 per stream, an improvement of $0.0063 per stream. At 10,000 streams per month, this difference compounds to $63.00 in additional provider net revenue from gas amortization alone.

5.8 Replay Protection

Each JWT's `jti` value is stored in Redis at issuance with a 24-hour TTL matching the token expiry. On any WebSocket upgrade request, the facilitator checks that the `jti` is present and has not been marked as consumed. Token theft does not yield a usable credential because the legitimate session has already registered the `jti`. Expired tokens are automatically evicted from Redis by the TTL mechanism, preventing accumulation of stale state.

5.9 Chain Selection

The facilitator architecture is chain-agnostic at the design level. Two settlement chains are currently supported.

Base offers established x402 tooling, Coinbase ecosystem alignment, and native USDC issuance by Circle, which launched on Base in September 2023 [10]. Base gas per USDC transfer is approximately $0.007 under normal network load [11]. Base is the natural choice for EVM-native data providers and those already operating within the Coinbase infrastructure ecosystem.

Solana offers lower gas costs: approximately $0.00125 per transaction including priority fee [12], and faster finality at approximately 400 milliseconds [12]. Solana is particularly well-suited for providers such as Allnodes, which hosts top Solana validators, where USDC liquidity on Solana is deep and the provider's customer base is Solana-native. The x402 protocol on Solana has already processed 35 million transactions and $10 million in volume [12].

Providers select their preferred settlement chain during feed registration. The facilitator handles chain-specific settlement logic transparently.

5.10 Implementation Note

The facilitator proxy layer is implemented in Rust for memory safety and performance at the connection layer, where thousands of concurrent WebSocket sessions must be managed with minimal per-connection overhead. Rust's ownership model eliminates the class of memory safety bugs that create exploitable vulnerabilities in long-running network services, and its async runtime delivers the throughput necessary for high-frequency streaming workloads without garbage collection pauses. The stateful session design in this architecture draws on adaptive memory and streaming architecture concepts developed in Mudbari and Bhagat (2026) [13], which examined structured memory management for continuous financial time series data.

6. Revenue Model for Data Providers

The facilitator's revenue model is a percentage of USDC settled through its infrastructure. Data providers pay nothing unless agents are paying them. This structure creates zero-risk adoption: a provider who integrates the facilitator and receives no agent traffic incurs no cost. A provider who receives agent traffic earns revenue proportional to that traffic, with the facilitator taking a 1% gross cut from each stream's settlement value. There is no setup fee, no monthly minimum, and no charge for periods of zero agent activity.

6.1 Per-Stream Economics

Tables 3 and 4 show the distribution of each $1.00 stream payment. All figures are drawn from the cost model in Appendix A.

Table 3: Per-Stream Payment Waterfall: Base (Single Settlement)

Item	Amount
Agent pays	$1.00000
Gas (Base, per settlement)	($0.00700)
Facilitator gross (1%)	($0.01000)
Facilitator infrastructure overhead	($0.00200)
Provider net per stream	**$0.98300**
Facilitator net per stream	**$0.00800**

Table 4: Per-Stream Payment Waterfall: Solana (Single Settlement)

Item	Amount
Agent pays	$1.00000
Gas (Solana, total with priority)	($0.00125)
Facilitator gross (1%)	($0.01000)
Facilitator infrastructure overhead	($0.00200)
Provider net per stream	**$0.98875**
Facilitator net per stream	**$0.00800**

6.2 Session Deposit Model

When agents deposit USDC in a session block rather than settling per stream, gas is incurred once per session rather than once per stream. Table 5 compares single settlement to session deposit for a 10-stream session on Base.

Table 5: Session Deposit vs. Single Settlement Comparison (10 Streams, Base)

Metric	Single Settlement	Session Deposit	Delta
Gas per stream	$0.00700	$0.00070	−90.0%
Provider net per stream	$0.98300	$0.98930	+$0.00630
Provider net (10 streams)	$9.83000	$9.89300	+$0.063
On-chain transactions	10	1	−9

6.3 Provider Revenue Projections

Table 6 shows provider revenue projections on Base at $1.00 per stream with a 1% facilitator cut. Full calculations appear in Appendix A, Analysis A.3.

6.4 Subscription Revenue Comparison

QuickNode's Scale plan costs $499 per month flat [7]. Under the facilitator model at $1.00 per stream and 1% facilitator cut on Base, a provider needs 508 streams per month to generate equivalent net revenue:

Table 6: Provider Revenue Projections (Base, Single Settlement)

Monthly Streams	Total GMV	Gas Cost	Facilitator Net	Provider Net
100	$100.00	$0.70	$0.80	$98.30
1,000	$1,000.00	$7.00	$8.00	$983.00
10,000	$10,000.00	$70.00	$80.00	$9,830.00

$$\text{Required streams} = \frac{\$499.00}{\$0.983} = 507.63 \rightarrow 508 \text{ streams/month}$$

Table 7 shows how fleet scenarios compare to the QuickNode Scale plan equivalent.

Table 7: Facilitator Model vs. Subscription Revenue Comparison

Scenario	Monthly Streams	Provider Net
QuickNode Scale equivalent	508	$499.36
10 agents × 25 streams	250	$245.75
10 agents × 100 streams	1,000	$983.00
10 agents × 1,000 streams	10,000	$9,830.00

This is not a replacement for enterprise subscription revenue. It is a new revenue tier, capturing agent consumption that currently generates zero dollars for every data provider operating WebSocket infrastructure in the agentic economy.

7. Ecosystem Positioning

The agent payment protocol landscape has developed rapidly in 2025 and 2026, producing several distinct layers that serve different functions in the payment stack. Table 8 maps the current landscape.

Table 8: Agent Payment Protocol Positioning Matrix

Player	HTTP x402	WebSocket Payments	Layer	Target
x402 HTTP facilitators	Yes	No	Application	API developers
Stripe MPP	No (fiat-first)	Partial (session)	Application	Merchant/e-commerce
Google AP2	Authorization only	N/A	Authorization	Enterprise orchestrators
This facilitator	Yes (pre-auth)	Yes (native)	Infrastructure	Data providers, node operators

Google AP2 operates above the payment execution layer. Its function is to define how agents prove user intent and enforce spending limits before payment is authorized [5]. It does not execute payments. Google integrated x402 as an execution layer beneath AP2 [5]. Our facilitator operates at the same execution layer, making AP2 and our facilitator architecturally compatible without requiring protocol modification on either side.

Stripe MPP launched March 18, 2026 [14]. It offers session-based payment aggregation for high-frequency agent interactions, supporting fiat and stablecoin hybrid settlement through Shared Payment Tokens (SPTs). Stripe MPP is well-suited for e-commerce agent use cases within Stripe's merchant ecosystem. It is a weaker fit for raw data feed infrastructure: its compliance stack assumes KYC-capable entities, and its merchant model assumes a human-configured payment relationship, overhead that the facilitator model eliminates for blockchain node providers.

x402 HTTP facilitators, including Coinbase CDP, PayAI, and Meridian, operate exclusively on HTTP request-response interactions [1, 2]. They are well-suited for API call monetization and per-inference billing. They do not address WebSocket streaming, as confirmed by the protocol's design and by third-party analysis [14]. Our facilitator does not compete with these HTTP facilitators; it serves a different transport layer.

Our facilitator is purpose-built for raw data feed providers operating persistent WebSocket endpoints. It operates at the infrastructure layer rather than the application layer. Data providers do not need to choose between x402 and this facilitator. A provider can operate HTTP x402 facilitators for its REST endpoints and our facilitator for its WebSocket endpoints simultaneously, using the same USDC wallet and the same agent payment flow from the agent's perspective. The x402 V2 extension model allows new functionality through plugins without breaking changes [1].

If a future x402 extension addresses WebSocket natively, our facilitator architecture provides the reference implementation. Until that extension exists, our facilitator is the only deployed solution for WebSocket-native x402 payment gating in production.

8. Integration Playbook

Integrating the WebSocket-native x402 facilitator into an existing data provider's infrastructure requires three steps. The provider requires no blockchain infrastructure. The provider needs no protocol expertise beyond standard HTTP middleware and WebSocket connection handling. The integration surface is minimal by design.

Step 1: Register the feed. The data provider specifies which WebSocket endpoints to gate, the price per stream in USDC, and the wallet address to receive settlements. Feed registration occurs through our facilitator's configuration interface. The provider sets the stream price as a USDC amount per individual WebSocket session open, for example $1.00 per stream, and selects the settlement chain: Base for EVM-native teams or Solana for providers whose customer base is Solana-native. The facilitator generates a feed identifier used in JWT session tokens for this endpoint. The provider receives no blockchain integration requirement and needs no on-chain infrastructure. Settlement USDC arrives in the specified wallet address on the provider's chosen chain.

Step 2: Add the pre-auth endpoint. The provider exposes a single HTTP endpoint that returns x402-compliant payment instructions when an agent requests a WebSocket upgrade without a valid session token. In practice, this is one middleware line in any modern HTTP server framework. The facilitator provides a middleware package for Node.js, Python FastAPI, Go `net/http`, and Rust Axum, each requiring only the feed identifier and the provider's settlement wallet address as configuration parameters. The middleware handles the 402 response construction, payment instruction serialization, and success callback routing automatically.

Step 3: Gate the WebSocket upgrade. The provider's WebSocket server validates the JWT session token issued by the facilitator before completing the upgrade handshake. The facilitator provides a verification SDK in Rust, Go, TypeScript, and Python. The SDK exposes a single function: `verify_session_token(token, feed_id)`, which returns a validated session context or an error. The provider's WebSocket handler calls this function at upgrade time and rejects connections with invalid or exhausted tokens. The facilitator accepts connections with valid tokens; they begin receiving data. The facilitator's proxy layer manages all ongoing balance metering and CLOSE frame delivery; the provider's WebSocket server treats the connection as a standard authenticated session once the upgrade is complete.

We operate a pilot program for data providers seeking to add agent-native billing to their WebSocket infrastructure. Please contact Manas Mudbari (manas.mudbari@turboline.ai) or Chandan Bhagat (chandan.bhagat@turboline.ai) to discuss feed configuration and settlement chain selection.

References

[1] E. Reppel, C. Roscoe, and J. Nickerson, "x402 V2," https://www.x402.org/writing/x402-v2-launch, December 2025, published December 11, 2025. x402 Foundation. Primary source for adoption statistics, V2 feature set, and session model.

[2] Cloudflare, Inc., "x402: HTTP payments for the agentic web," https://blog.cloudflare.com/x402/, 2025, accessed April 2026. Describes five-step x402 protocol flow and Cloudflare deferred payment extension.

[3] Allium, "x402 explained: The internet-native payments standard for APIs, data, and agent commerce," https://www.allium.so/blog/x402-explained-the-internet-native-payments-standard-for-apis-data-and-agent-commerce/, 2025, accessed April 2026. Technical protocol flow description and facilitator role explanation.

[4] Galaxy Research, "Agentic commerce market outlook," January 2026, cited secondarily via: Stellar Development Foundation, "x402 on Stellar," stellar.org, 2026. Projected agentic commerce at $3–5 trillion B2C revenue by 2030.

[5] Stellar Development Foundation, "x402 on Stellar," https://stellar.org/blog/foundation-news/x402-on-stellar, 2026, accessed April 2026. Source for Galaxy Research market sizing and x402 ecosystem membership.

[6] I. Fette and A. Melnikov, "The WebSocket protocol," IETF, RFC 6455, December 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6455

[7] QuickNode, "Pricing," https://www.quicknode.com/pricing, 2026, accessed April 2026. Scale plan priced at $499/month ($424/month billed annually); 950M API credits, 250 req/sec, 50 endpoints.

[8] GetBlock, "Pricing," https://getblock.io/pricing, 2026, accessed April 2026. Subscription plans at $0, $49, $199, $499, and $999/month (Compute Unit-based); dedicated nodes from $1,000/month. No per-stream or per-session billing for autonomous clients.

[9] Allnodes, "Pricing," https://www.allnodes.com/pricing, 2026, accessed April 2026. Node hosting on fixed subscription tiers (e.g., $320–$640/month for dedicated nodes); no per-stream or agent-native billing path.

[10] Circle, "USDC now available natively on Base," https://www.circle.com/blog/usdc-now-available-natively-on-base, September 2023, circle blog post confirming native USDC issuance on Base, replacing the bridged USDbC token.

[11] Base, "Network fees," https://docs.base.org/base-chain/network-information/network-fees, 2025, accessed April 2026. Base network fee documentation; used for L2 gas cost estimates.

[12] Solana Foundation, "What is x402?" https://solana.com/x402/what-is-x402, 2025, accessed April 2026. Source for Solana transaction cost $(0.00025 base) and finality time (400 ms)$.

[13] M. Mudbari and C. Bhagat, "Adaptive memory for LLM-based time series analysis: A case study on Bitcoin regime detection," *engrXiv*, March 2026, published March 9, 2026. [Online]. Available: https://doi.org/10.31224/6603

[14] WorkOS, "x402 vs. Stripe MPP: How to choose payment infrastructure for AI agents and MCP tools in 2026," https://workos.com/blog/x402-vs-stripe-mpp-how-to-choose-payment-infrastructure-for-ai-agents-and-mcp-tools-in-2026, 2026, accessed April 2026. Third-party comparison confirming x402 lacks WebSocket and streaming support.

A. Cost Model

Full calculation detail for all figures cited in this paper. All arithmetic shown.

A.1: Per-Stream Economics (Single Settlement)

Inputs: $1.00 stream price; Base gas $0.007; Solana gas $0.00125; facilitator cut 1% ($0.010); facilitator infrastructure overhead $0.002.

On Base

```
Agent pays:                  $1.00000
Less gas (Base):            -$0.00700
Less facilitator cut (1%):  -$0.01000
Provider net per stream:     $0.98300

Facilitator gross:           $0.01000
Less infra overhead:        -$0.00200
Facilitator net per stream:  $0.00800

Verification: $0.98300 + $0.00800 + $0.00200 + $0.00700 = $1.00000 OK
```

On Solana

```
Agent pays:                  $1.00000
Less gas (Solana):          -$0.00125
Less facilitator cut (1%):  -$0.01000
Provider net per stream:     $0.98875

Facilitator gross:           $0.01000
Less infra overhead:        -$0.00200
Facilitator net per stream:  $0.00800

Verification: $0.98875 + $0.00800 + $0.00200 + $0.00125 = $1.00000 OK
```

A.2: Session Deposit Model vs. Single Settlement (10 Streams, Base)

Single Settlement (10 separate on-chain transactions)

```
Total payment:          10 x $1.00  = $10.00000
Total gas:              10 x $0.007 =  $0.07000
Total facilitator net:  10 x $0.008 =  $0.08000
Total provider net:     10 x $0.983 =  $9.83000
```

Session Deposit ($10.00 USDC, one transaction)

```
Total payment (deposit):  $10.00000
Gas (one transaction):    -$0.00700
Facilitator gross (1%):   -$0.10000
Facilitator infra:        -$0.02000
Facilitator net:           $0.08000
Provider net:              $9.89300
Provider net per stream:   $0.98930

Verification: $9.893 + $0.08 + $0.02 + $0.007 = $10.000 OK
```

A.3: Provider Revenue Projection (Base, Single Settlement)

Provider net per stream: $0.983

```
100 streams:
  GMV = $100.00 | Gas = $0.70 | Facilitator net = $0.80 | Provider net =
     $98.30

1,000 streams:
  GMV = $1,000.00 | Gas = $7.00 | Facilitator net = $8.00 | Provider net =
      $983.00

10,000 streams:
  GMV = $10,000.00 | Gas = $70.00 | Facilitator net = $80.00 | Provider
     net = $9,830.00
```

A.4: Subscription Revenue Comparison

QuickNode Scale plan: $499.00/month flat [7].

```
Streams to match $499.00 net:
  $499.00 / $0.983 = 507.63 -> 508 streams/month

Fleet scenario (10 agents x 25 streams):
  250 streams x $0.983 = $245.75/month
```

*All citations resolved. No **[CITATION NEEDED]** flags remain.*