Independent Submission
Internet-Draft
Intended status: Informational
Expires: 23 October 2026

A. Ganji
Independent Researcher

23 April 2026

Fluid Agentic DeFi Protocol (FADP/1.0): HTTP-Native Micropayment Authentication for Autonomous AI Agents
draft-fluid-fadp-01

Abstract

This document defines the Fluid Agentic DeFi Protocol (FADP), version 1.0. FADP is an application-layer protocol layered atop HTTP that enables autonomous AI agents to pay for access to web resources using on-chain cryptocurrency transfers, with cryptographic proof of payment embedded directly in HTTP headers.

FADP extends HTTP 402 (Payment Required) with two new header fields: X-FADP-Required, which a server uses to communicate payment terms to an agent, and X-FADP-Proof, which an agent uses to supply a verifiable on-chain payment receipt. A nonce-challenge mechanism prevents proof replay. An optional verification endpoint allows third-party on-chain confirmation without requiring the verifying party to run blockchain infrastructure.

FADP is designed for agent-to-server and agent-to-agent payment flows operating at sub-dollar granularity (micropayments), where credit-card or OAuth-based billing is impractical. The reference implementation targets USDC on Base (an Ethereum Layer 2), though the protocol is token- and chain-agnostic.

About This Document

This note is to be removed before publishing as an RFC.

This document is submitted as an independent submission to the IETF datatracker. It does not represent the consensus of any IETF working group. The author welcomes discussion on the GitHub repository (https://github.com/fluidbase9/fadp) and via the contact address above.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

Autonomous AI agents — software processes that browse the web, call APIs, and complete tasks on behalf of users without per-action human approval — are an emerging class of HTTP client. Unlike human-operated browsers, these agents cannot authenticate themselves to paid services via credit-card billing flows, OAuth consent screens, or similar interactive mechanisms. The practical result is that most commercial APIs are inaccessible to fully autonomous agents, forcing developers to hard-code credentials that cannot be scoped, limited, or revoked per-task.

HTTP has carried a 402 (Payment Required) status code since RFC 7231 [RFC7231], but that specification explicitly reserved the code for future use and defined no standard payment handshake. Numerous proprietary schemes have since emerged, but none has achieved broad adoption, and none addresses the specific needs of machine-to-machine micropayments at the scale and frequency that autonomous agents require.

Blockchain-based stablecoins — particularly USDC on high-throughput, low-cost Layer 2 networks — have made sub-cent on-chain transfers economically viable for the first time. A transaction settling 0.001 USDC (one-tenth of one cent) on Base costs roughly USD 0.0001 in gas, making per-request billing at very fine granularity practical.

FADP defines a minimal, HTTP-native handshake that combines these two properties: the well-understood 402 status code as a payment prompt, and an on-chain transfer as the payment instrument. The protocol

requires no browser, no OAuth server, no API-key issuance flow, and no pre-existing billing relationship between the agent and the server. An agent that has a funded wallet and a Fluid agent key can pay for any FADP-gated resource it discovers, at runtime, without human intervention—provided the requested amount is within the agent's pre-configured spending limit.

1.1. Design Goals

* *HTTP-native*: the entire payment handshake is expressed in standard HTTP status codes and header fields. No new transport layer is required.

* *Stateless from the agent's perspective*: the server supplies all information needed to pay in the 402 response; the agent needs no prior knowledge of the server's payment address or preferred token.

* *Replay-safe*: a nonce bound to each payment challenge ensures that a valid payment proof cannot be reused across requests or across sessions.

* *Verifier-decoupled*: servers can delegate on-chain verification to a third-party service, removing the need to run or maintain blockchain node infrastructure.

* *Token- and chain-agnostic*: while USDC on Base is the reference currency, FADP carries token and chain identifiers in every message and imposes no restriction on supported assets.

* *Human-in-the-loop friendly*: spending limits and out-of-band approval flows are outside protocol scope but are natural affordances of agent runtime environments built on FADP.

1.2. Non-Goals

* FADP does not specify how agents acquire, store, or rotate their wallet credentials.

* FADP does not define a streaming or subscription billing model. Each request-response pair is an independent payment event.

* FADP does not mandate a specific blockchain or token.

* FADP does not replace existing API authentication schemes; it may operate alongside them.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Agent An autonomous software process that issues HTTP requests on behalf of a user or another automated system, without interactive human approval for each request.

Server An HTTP origin server that gates one or more resources behind a FADP payment requirement.

Payment Challenge A 402 HTTP response carrying an X-FADP-Required header that specifies the payment terms the agent must satisfy.

Payment Proof A JSON object carried in the X-FADP-Proof header of a subsequent request, asserting that the agent has executed an on-chain transfer matching the payment challenge.

Nonce A cryptographically random, single-use token embedded in a payment challenge. The server MUST reject any proof whose nonce has already been consumed or has expired.

Verification Service An HTTPS endpoint that accepts a payment proof and independently confirms the corresponding on-chain transaction. May be operated by a third party.

On-chain Transfer A transaction recorded in a distributed ledger (blockchain) that transfers a specified amount of a digital asset from one address to another.

Transaction Hash (txHash) A unique identifier for an on-chain transfer, derived from the cryptographic hash of the transaction data. Immutable once the transaction is finalized.

3. Protocol Overview

A FADP interaction consists of three HTTP exchanges:

1. *Initial Request*: The agent sends an ordinary HTTP request to a protected resource, without any payment header.

2. *Payment Challenge*: The server responds with HTTP 402 and an X-FADP-Required header describing the payment terms. The agent executes an on-chain transfer matching those terms.

3. *Authenticated Request*: The agent repeats the original request, adding an X-FADP-Proof header that references the completed on-chain transfer and the challenge nonce. The server verifies the proof and, if valid, fulfills the request.

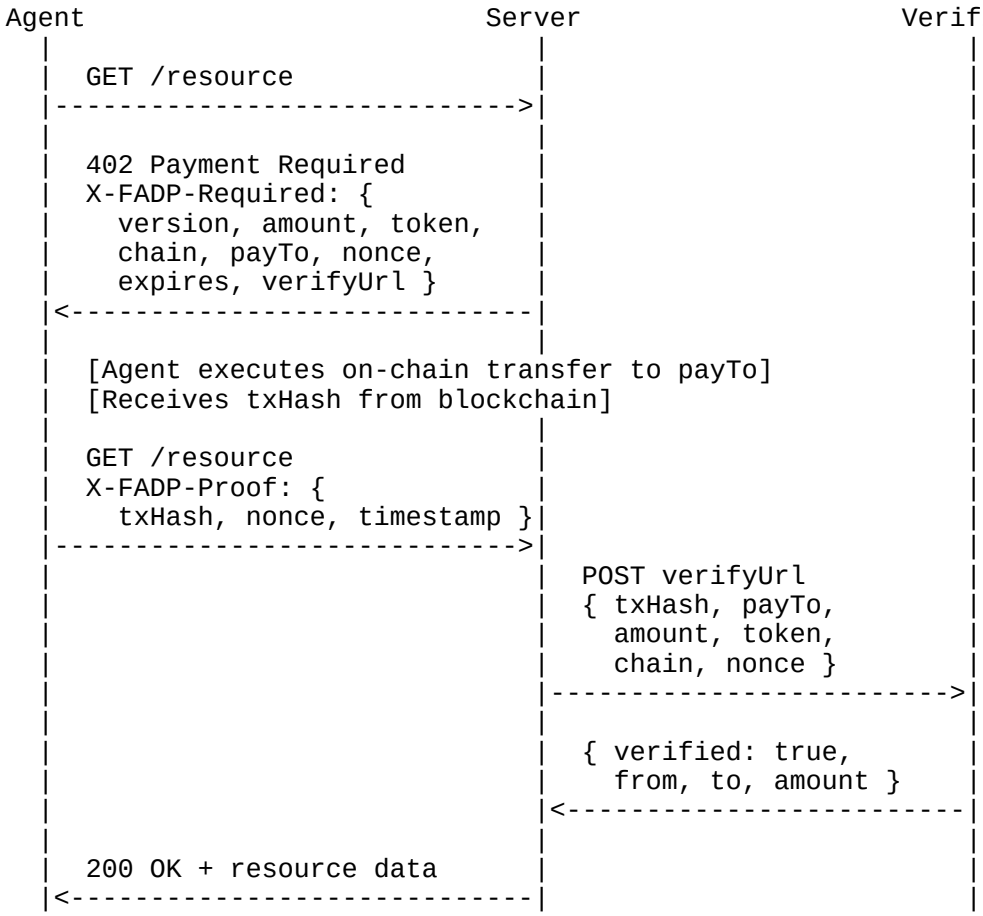

Figure 1: FADP Full Handshake

4. Header Fields

4.1. X-FADP-Required

The X-FADP-Required response header field is sent by the server in a 402 response. Its value is a JSON object with the following members:

version REQUIRED. String. MUST be "1.0" for this version of the protocol.

amount REQUIRED. String. The amount of the specified token that the agent MUST transfer, expressed as a decimal number (e.g., "0.001"). The server SHOULD use the minimal number of decimal places necessary to avoid ambiguity.

token REQUIRED. String. The symbol of the digital asset to be transferred (e.g., "USDC", "ETH", "USDT"). The symbol SHOULD conform to the canonical symbol registered with the issuer or a well-known token list.

chain REQUIRED. String. The identifier of the blockchain network on which the transfer must be executed. Registered values include "base" (Base Mainnet), "ethereum" (Ethereum Mainnet), and "solana" (Solana Mainnet). Implementors MAY define additional chain identifiers using reverse-DNS notation to avoid collision.

payTo REQUIRED. String. The blockchain address of the server operator's wallet. For EVM-compatible chains, this MUST be a checksummed Ethereum address (EIP-55).

nonce REQUIRED. String. A cryptographically random, single-use token generated by the server for this payment challenge. The server MUST ensure that each nonce is unique across all active challenges. The nonce MUST be at least 16 bytes of entropy, encoded as a lowercase hexadecimal string (minimum 32 characters).

expires REQUIRED. Number. A Unix timestamp (seconds since epoch) indicating when this payment challenge expires. The server MUST reject proofs referencing a nonce whose expiry has passed. The RECOMMENDED default TTL is 300 seconds (5 minutes).

description OPTIONAL. String. A human-readable description of the resource or service being paid for (e.g., "Premium market data API"). Intended for display in agent UIs and transaction logs.

verifyUrl OPTIONAL. String. An HTTPS URL identifying the verification service the server will use to confirm payment proofs. If omitted, the default is https://fluidnative.com/v1/fadp/verify. Servers operating their own verification infrastructure SHOULD set this field.

Example value (formatted for readability; actual header value MUST be a single-line JSON string):

```
{
  "version": "1.0",
  "amount": "0.001",
  "token": "USDC",
  "chain": "base",
  "payTo": "0xAbCd1234AbCd1234AbCd1234AbCd1234AbCd1234",
  "nonce": "a3f9c2b1d4e5f6a7b8c9d0e1f2a3b4c5",
  "expires": 1745001234,
  "description": "GPT-4o inference via OpenRouter proxy",
  "verifyUrl": "https://fluidnative.com/v1/fadp/verify"
}
```

4.2. X-FADP-Proof

The X-FADP-Proof request header field is sent by the agent in the authenticated request. Its value is a JSON object with the following members:

txHash REQUIRED. String. The transaction hash of the on-chain transfer executed by the agent in response to the payment challenge. For EVM-compatible chains, this is a 0x-prefixed 32-byte hex string.

nonce REQUIRED. String. The nonce value copied verbatim from the X-FADP-Required header of the payment challenge to which this proof responds. The server MUST verify that this nonce matches a known, unexpired, unconsumed challenge.

timestamp REQUIRED. Number. A Unix timestamp (seconds since epoch) representing the time at which the agent assembled this proof. The server SHOULD reject proofs where |currentTime - timestamp| > 300 seconds.

agentKeyPrefix OPTIONAL. String. A non-sensitive prefix of the agent's credential (e.g., the first 8 characters of a fwag_ key), used for logging and audit. Implementations MUST NOT include the full credential.

Example:

```
{
  "txHash": "0xabc1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcd",
  "nonce": "a3f9c2b1d4e5f6a7b8c9d0e1f2a3b4c5",
  "timestamp": 1745001200,
  "agentKeyPrefix": "fwag_a3f9"
}
```

5. Verification Endpoint

The verification endpoint is an HTTPS resource that the server (or its delegate) calls to confirm that a payment proof corresponds to a valid on-chain transfer. This section defines the request and response formats for that endpoint.

5.1. Verification Request

The server sends an HTTP POST to the verifyUrl with a JSON body containing:

txHash REQUIRED. String. The transaction hash from the agent's proof.

payTo REQUIRED. String. The wallet address that should have received the transfer.

amount REQUIRED. String. The amount that should have been transferred.

token REQUIRED. String. The token symbol.

chain REQUIRED. String. The chain identifier.

nonce REQUIRED. String. The nonce from the original challenge.

5.2. Verification Response

The verification service responds with a JSON object:

verified REQUIRED. Boolean. true if and only if all of the following conditions hold:

* The transaction identified by txHash exists on the specified chain.
* The transaction status is finalized (not reverted).
* The recipient address matches payTo.
* The transferred amount is greater than or equal to the required amount.
* The transferred token matches the required token.

txHash OPTIONAL. String. Echo of the verified transaction hash.

amount OPTIONAL. String. The actual transferred amount.

token OPTIONAL. String. The actual transferred token symbol.

chain OPTIONAL. String. The chain on which the transfer occurred.

from OPTIONAL. String. The sender’s wallet address.

to OPTIONAL. String. The recipient’s wallet address (SHOULD match payTo).

error OPTIONAL. String. A human-readable error message when verified is false.

Example successful response:

```
{
  "verified": true,
  "txHash": "0xabc123...",
  "amount": "0.001",
  "token": "USDC",
  "chain": "base",
  "from": "0xAgentWallet...",
  "to": "0xServerWallet..."
}
```

Example failed response:

```
{
  "verified": false,
  "error": "Transfer amount 0.0005 USDC is less than required 0.001 USDC"
}
```

6. Server Behavior

6.1. Issuing Payment Challenges

When a server receives a request for a FADP-protected resource without an X-FADP-Proof header, it MUST:

1. Generate a nonce of at least 16 bytes of cryptographically secure random data.

2. Record the nonce and its expiry time in durable server-side storage (memory, cache, or database).

3. Respond with HTTP 402.

4. Set the X-FADP-Required header to a JSON object as defined in Section 4.1.

5. Set Access-Control-Expose-Headers: X-FADP-Required to ensure cross-origin agents can read the header.

6.2. Verifying Payment Proofs

When a server receives a request with an X-FADP-Proof header, it MUST perform the following checks in order, returning an appropriate error response if any check fails:

1. Parse the header value as JSON; return 400 if malformed.

2. Verify that txHash, nonce, and timestamp are present; return 400 if any is absent.

3. Look up the nonce in server-side storage; return 402 if the nonce is unknown.

4. Verify the nonce has not expired; return 402 if expired (delete the nonce).

5. Verify that |currentTime - proof.timestamp| <= 300 seconds; return 402 if outside range.

6. Call the verification service at verifyUrl; return 402 if verified is false.

7. Atomically mark the nonce as consumed (delete it from storage) to prevent replay.

8. Proceed to handle the original request.

Step 7 (nonce consumption) MUST occur before the server fulfills the request. If the server cannot atomically consume the nonce and fulfill the request (e.g., due to a crash), it SHOULD err on the side of not consuming the nonce, accepting the risk of a duplicate fulfillment rather than silently accepting payment without delivering service.

7. Agent Behavior

An agent implementation SHOULD:

1. On receiving a 402 with X-FADP-Required, validate that the requested amount does not exceed the agent’s configured spending limit before proceeding.

2. Execute the on-chain transfer to the address and chain specified in the payment challenge, transferring at least the specified amount of the specified token.

3. Await sufficient blockchain confirmation before constructing the proof. The definition of "sufficient" is implementation-specific; at minimum, the transaction MUST appear in a block (status: success) before the proof is submitted.

4. Construct an X-FADP-Proof object containing the txHash, the nonce from the challenge, and the current Unix timestamp.

5. Retry the original request with the X-FADP-Proof header added.

6. If the server returns another 402 with a new nonce (e.g., because the previous nonce expired), restart the payment flow.

An agent MUST NOT reuse a txHash or nonce across separate request/response cycles. Each resource access requires a fresh payment and a fresh proof.

8. Error Responses

FADP servers SHOULD include a JSON body in all error responses with at least the following fields:

error String. A machine-readable error key.

protocol String. SHOULD be "FADP/1.0" to identify the protocol version.

detail String. OPTIONAL human-readable elaboration.

HTTP Status	Condition	Recommended error value
400	Malformed X-FADP-Proof header (not valid JSON)	invalid_proof_format
400	Missing required fields in proof	missing_proof_fields
402	No proof header present (initial challenge)	payment_required
402	Nonce unknown (never issued or already expired from store)	unknown_nonce
402	Nonce expired (TTL elapsed)	nonce_expired
402	Proof timestamp too old or too far in future	proof_timestamp_invalid
402	On-chain verification returned verified: false	payment_verification_failed
402	Transferred amount less than required	insufficient_payment
403	Nonce has already been consumed (replay attempt)	nonce_already_used

Table 1: FADP Error Conditions

9. Chain Identifiers

FADP uses short string identifiers for blockchain networks. The following values are defined by this specification:

Identifier	Network	Native Currency	EVM Chain ID
base	Base Mainnet (Coinbase L2)	ETH	8453
ethereum	Ethereum Mainnet	ETH	1
solana	Solana Mainnet Beta	SOL	N/A
injective	Injective Mainnet	INJ	N/A
base-sepolia	Base Sepolia Testnet	ETH	84532

Table 2: FADP Chain Identifier Registry

Implementors that wish to support chains not listed here SHOULD use identifiers of the form {namespace}-{networkname} to avoid collisions with future registered values.

10. Security Considerations

10.1. Replay Attacks

The nonce mechanism is the primary defense against replay attacks. A valid txHash from a past payment cannot be reused because:

1. Each payment challenge carries a unique nonce.
2. The proof MUST echo that nonce.
3. The server atomically consumes the nonce upon first successful verification.
4. Subsequent proofs referencing the same nonce are rejected with 403.

Server implementations MUST use cryptographically secure random number generation for nonces (e.g., /dev/urandom or equivalent). Predictable nonces would allow an attacker to pre-compute valid proofs.

The nonce store MUST be shared across all instances of a horizontally scaled server deployment. If a local per-process nonce store is used, the same nonce could be accepted by two different instances simultaneously, creating a replay window. A distributed cache (e.g., Redis) or a database with atomic compare-and-delete operations is RECOMMENDED for production deployments.

10.2. Overpayment

The verification service checks that the transferred amount is _at least_ the required amount. Servers SHOULD NOT issue refunds for overpayments, as the complexity of doing so on-chain outweighs the benefit for small amounts. Agents SHOULD transfer exactly the required amount to avoid unnecessary token loss.

10.3. Front-Running and Transaction Interception

In principle, an attacker observing the agent's on-chain transaction in the mempool could copy the txHash and attempt to submit a proof before the legitimate agent. This attack is mitigated by:

1. The nonce binding: the attacker must have obtained the nonce from a legitimate 402 challenge targeted at their own session. Nonces are per-session and cannot be forged.

2. The proof timestamp check: the server rejects proofs assembled more than 300 seconds after the challenge was issued.

Servers that require strong session binding MAY add an additional check verifying that the HTTP session initiating the proof matches the session that triggered the original challenge (e.g., via a session cookie or TLS channel binding), though this is outside the scope of the current protocol version.

10.4. Verification Service Trust

Servers that delegate verification to a third-party service are implicitly trusting that service's honesty and availability. A compromised or malicious verification service could return verified: true for invalid proofs, allowing unauthorized access.

Servers operating in high-value contexts SHOULD run their own verification infrastructure against a direct blockchain RPC endpoint. The verification endpoint API defined in Section 5 is designed to be self-hostable for this reason.

10.5. Blockchain Finality

On chains with probabilistic finality (including Ethereum and Base), a transaction that appears in a block may be reorged out of the canonical chain. Servers accepting payments for high-value resources SHOULD wait for a chain-specific number of confirmation blocks before accepting a proof as final. For low-value micropayments on Base (an optimistic rollup with fast soft-finality), a single block confirmation is typically sufficient.

Verification services MUST check that the transaction receipt status indicates success (i.e., it was not reverted), not merely that a transaction with the given hash exists.

10.6. Nonce Store Exhaustion

An adversary could generate large numbers of initial requests to force the server to allocate nonces, potentially exhausting memory if the nonce store grows without bound. Server implementations MUST periodically prune expired nonces and SHOULD rate-limit the issuance of payment challenges per IP address or session.

10.7. Header Injection

The X-FADP-Required and X-FADP-Proof header values are JSON strings. Implementations MUST parse these with a conformant JSON parser and MUST NOT construct them via string interpolation without proper escaping, to prevent injection attacks.

10.8. Transport Security

All FADP exchanges, including calls to the verification service, MUST be made over TLS (HTTPS). Transmitting payment challenges or proofs over plaintext HTTP exposes the nonce to network adversaries, potentially enabling race-condition replay attacks.

10.9. Agent Spending Controls

FADP does not define agent-side spending controls; these are the responsibility of the agent runtime. However, agent implementors MUST validate the requested amount in the payment challenge against a pre-configured limit before executing any transfer. Failing to do so could allow a malicious server to drain an agent’s wallet by issuing artificially large payment challenges.

11. IANA Considerations

11.1. HTTP Header Field Registration

This document requests registration of the following header fields in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" maintained at https://www.iana.org/assignments/http-fields/.

Header Field Name X-FADP-Required

Status provisional

Reference This document, Section 4.1

Header Field Name X-FADP-Proof

Status provisional

Reference This document, Section 4.2

11.2. Media Type

No new media types are registered by this document. All message bodies are application/json as defined in [RFC8259].

12. Implementation Notes

12.1. Reference Implementations

Reference implementations of the server middleware (fadpGate()) and client interceptor (fadpFetch()) are available as open-source software:

* *fluid-fadp* (npm): server middleware and standalone client interceptor for Node.js / TypeScript. Repository: https://github.com/fluidbase9/fadp

* *fluid-wallet-agentkit* (npm): full agent SDK incorporating FADP auto-pay via the agent.fetch() method. Repository: https://github.com/fluidbase9/fluidwalletbase

* *fluid-fadp-proxy* (npm): a ready-made reverse proxy that gates commercial AI service APIs (OpenRouter, RunPod, Vast.ai, Spheron) behind FADP payment walls. Repository: https://github.com/fluidbase9/fadp-proxy

12.2. Nonce Storage in Distributed Deployments

For single-process servers, an in-memory Map with periodic TTL pruning is sufficient. For multi-instance deployments, a shared store with atomic operations is required. A Redis SET NX EX command provides an atomic "set if not exists with expiry" primitive suitable for nonce issuance; a Redis DEL returning the count of deleted keys provides atomic consumption.

12.3. Amount Precision

Token amounts MUST be represented as decimal strings to avoid floating-point precision loss. On-chain, USDC uses 6 decimal places; the string "0.001" corresponds to on-chain value 1000 (in the smallest unit). Implementations MUST convert between decimal string and on-chain integer representation using fixed-point arithmetic, not IEEE 754 floating-point.

12.4. Cross-Origin Resource Sharing

Browser-based agent implementations require that X-FADP-Required be listed in the Access-Control-Expose-Headers response header for the 402 response. Servers MUST set this header when cross-origin clients are expected.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/rfc/rfc7231>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8259] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/rfc/rfc8259>.

[RFC9110] Fielding, R., Nottingham, M., and J. Reschke, "HTTP Semantics", RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

13.2. Informative References

[BASE-L2] Coinbase, "Base: An Ethereum L2 for the Next Billion Users", Technical overview, base.org, 2023. Base Mainnet is the reference deployment chain for FADP's reference implementation.

[EIP-20] Vogelsteller, F. and V. Buterin, "ERC-20 Token Standard", Ethereum Improvement Proposal 20, November 2015. Defines the fungible token interface implemented by USDC and most payment tokens used with FADP.

[EIP-55] Buterin, V., "Mixed-case checksum address encoding", Ethereum Improvement Proposal 55, January 2016. Defines the checksummed Ethereum address format used in FADP address fields.

[x402] Coinbase Developer Platform, "x402: An Open HTTP Payment Protocol", github.com/coinbase/x402, 2025. A contemporaneous HTTP 402-based payment protocol from Coinbase. FADP shares the 402 status code and on-chain settlement approach but differs in nonce mechanism, header naming, and protocol extensibility design.

Appendix A: Complete Exchange Example

The following illustrates a complete FADP exchange over HTTP/1.1.

Step 1 — Initial Request (no proof):

```
POST /v1/chat/completions HTTP/1.1
Host: proxy.example.com
Content-Type: application/json

{"model":"openai/gpt-4o","messages":[{"role":"user","content":"Hello"}]}
```

Step 2 — Payment Challenge:

```
   HTTP/1.1 402 Payment Required
   Content-Type: application/json
   X-FADP-Required: {"version":"1.0","amount":"0.001","token":"USDC",
     "chain":"base","payTo":"0xAbCd1234AbCd1234AbCd1234AbCd1234AbCd1234",
     "nonce":"a3f9c2b1d4e5f6a7b8c9d0e1f2a3b4c5","expires":1745001534,
     "description":"GPT-4o inference","verifyUrl":"https://fluidnative.com/v1/fadp/verify
"}
   Access-Control-Expose-Headers: X-FADP-Required

   {"error":"Payment required","protocol":"FADP/1.0"}

   *Step 3 — Agent executes on-chain USDC transfer on Base, receives
   txHash.*

   *Step 4 — Authenticated Request:*

   POST /v1/chat/completions HTTP/1.1
   Host: proxy.example.com
   Content-Type: application/json
   X-FADP-Proof: {"txHash":"0xabc123...","nonce":"a3f9c2b1d4e5f6a7b8c9d0e1f2a3b4c5",
     "timestamp":1745001480}

   {"model":"openai/gpt-4o","messages":[{"role":"user","content":"Hello"}]}

   *Step 5 — Successful Response (after on-chain verification):*

   HTTP/1.1 200 OK
   Content-Type: application/json

   {"choices":[{"message":{"role":"assistant","content":"Hello! How can I help?"}}]}
```

Acknowledgements

The author thanks the teams building open agent infrastructure across the Ethereum, Base, and Solana ecosystems whose work made practical micropayment settlement possible. The design of FADP was informed by the x402 protocol from Coinbase Developer Platform, the Model Context Protocol (MCP), and early feedback from developers building autonomous agents on the Fluid Wallet platform.

Author’s Address

Abhijeeth Ganji
Independent Researcher (Research Project: Fluid Wallet)
Email: fluidbase9@gmail.com
URI: https://fluidnative.com