h402

by BitGPT

White Paper

Creating the new generation of machine commerce

Gain a comprehensive understanding of our technology, strategy, and vision; backed by industry research, data, & expert analysis.

May 5, 2025

Contents

Prologue

Thirty-five years after Tim Berners-Lee set aside the `402 Payment Required` response code, the web still lacks a native, programmable unit of value.

In 2025, an autonomous agent can spin up a GPU in 200 milliseconds, request a language-model inference in 80 milliseconds, or stream sensor data to a fleet of microservices with sub-second round-trips. Yet, to pay for those resources, it must traverse seconds- to minutes-long card-network paths, opaque batch processors, or ad-hoc crypto gateways.

Over the past decade, we've patched around this gap with API keys, SaaS metering dashboards, usage spreadsheets, prepaid credits, ad impressions, OAuth scopes, embedded Stripe buttons, Lightning paywalls, ERC-20 `approve()`/`transferFrom()` gymnastics, and a half-dozen proprietary "credits" abstractions. Each workaround solves a local problem while reinforcing a global one: value exchange on the web remains siloed, lossy, and hostile to automation.

Today, we are entering a Cambrian moment for autonomous agents, where three forces converge:

- **LLM-powered agency.** Language models embedded in scripts, IDEs, and browser extensions now orchestrate tasks across dozens of APIs without human clicks. Their economic loops must run at machine speed.
- **Meter-every-byte infrastructure.** Cloud providers and CDNs invoice in milliseconds, gigabytes, and request counts; cost-allocation tools map every function call to P&L lines. Fine-grained pricing is now table stakes.
- **Plural on-chain settlement.** Public ledgers, from Bitcoin and Lightning to high-TPS L2s and Solana, have proven irreversible low-value transfers at Internet latencies and pennies per transaction.

In this landscape, an agent should be able to quote, authorize, broadcast, validate, and settle for a kilobyte of data, a millisecond of compute, or the unlock of a media fragment, all inside the same HTTP dialogue that already conveys authentication and content negotiation.

1 Introduction: `h402` Protocol

`h402` is a thin, header-based payment primitive that integrates directly into any HTTP interaction, turning every request–response cycle into an instant, verifiable settlement channel.

From a product standpoint, it enables API teams and digital goods providers to expose pay-per-call, streamed, or subscription pricing, without standing up billing dashboards, PCI workflows, or bespoke crypto gateways. Agents and end-users experience a single round-trip that quotes a price, authorizes funds across Bitcoin, Lightning, Solana, or EVM treasuries, and unlocks the service once the transfer is provably in flight.

1.1 The Missing Monetary Primitive in HTTP

When HTTP/1.0 codified the `401 Unauthorized` and `403 Forbidden` status codes in 1996, it addressed access, not payment. Over time, the web layered on bearer tokens, cookie sessions, and OAuth scopes, but these mechanisms merely gate content. They do not describe how much to pay, when to debit, or where funds should settle.

As SaaS proliferated, vendors improvised. Usage dashboards issue API keys to track request counts; metered services throttle until a credit card batch job clears; consumer sites embed Stripe or PayPal widgets that bounce users through multi-second redirects.

None of these patterns sit naturally in the hot loop of autonomous software.

Workaround	**Where it lives**	**What it solves**	**Pain for machine-to-machine flows**
API key in header	Server's internal DB	Who is calling	No price discovery; manual refills
OAuth 2.0 scope	Identity provider	Permission granularity	Separate billing back-end; token churn
Credit-card token on file	PSP vault (PCI)	Human cardholder billing	Seconds-long auth + batches; high fees
Prepaid "credits" SKU	Vendor database	Budget cap	Requires manual purchase; no real-time settle
Ad-hoc crypto gateway	Custom endpoint	One-off paywall	Chain-specific, opaque off-chain state

Table 1: Patchwork payment methods in HTTP and their limitations

Because none of these workarounds encode price, payment route, authorization, and proof of settlement inside the HTTP conversation itself, autonomous agents must juggle side channels, tolerate multi-second round-trips, or rely on custodial intermediaries.

The absence of a native monetary primitive is now the primary bottleneck preventing HTTP from powering real-time, agent-to-agent commerce at Internet scale.

1.2 Lessons from Usage-Based SaaS, FinOps, and Web3

Modern SaaS economics have decisively shifted toward meter-everything pricing. Snowflake charges per second of warehouse runtime, Datadog per ingested gigabyte, and API-native firms like OpenAI expose token-level tariffs. Analysts estimate that around 61% of SaaS providers now use usage- or consumption-based billing, a figure that has risen sharply since 2020 and is accelerating further as AI workloads drive per-request compute costs upward.[1]

Investors reward the model's alignment between revenue and usage; operators demand granular, automated metering down to the function call.

This shift has also reshaped financial operations. FinOps teams now allocate cloud costs in near real-time, tagging every Lambda invocation, GPU pod, and egress byte to a specific business unit. According to the 2025 *State of FinOps* survey, 40% of teams already manage SaaS spend in the same dashboards as their cloud spend, with a trend toward 65% within a year.[2]

Meanwhile, on-chain micropayments have moved beyond proof-of-concept. Bitcoin's Lightning Network now handles roughly 14–15% of all Bitcoin payment flows at major processors, a tenfold increase since 2022, achieving sub-cent fees and near-instant finality for sub-dollar transactions.[3]

These three arcs, fine-grained SaaS metering, budget-aware FinOps, and scalable micropayments, converge on a shared requirement: a native HTTP primitive that quotes, authorizes, and settles value within the same request path.

`h402` aims to close that loop, letting usage events *become* payment events, without detours through monthly invoices, batch-card processors, or bespoke crypto gateways.

2 Genesis of `h402`

The seed of `h402` was planted in December 2024 when the BitGPT team partnered with various AI-agent development frameworks, united by a shared vision: to build a decentralized mesh of autonomous agents.

As we collaborated with the community, two major friction points emerged:

- **Discoverability.** Newly built agents lacked a uniform mechanism to be discovered, or to discover remote services and the tasks they could perform.
- **Monetization.** Running agents is expensive, both in terms of compute and tokens. Builders wanted a reliable way to earn for contributing to the network.

While many projects are tackling the discoverability challenge, we chose to focus on monetization.

Early experiments, such as hard-coded price endpoints, prepaid token buckets, and network-specific paywalls, solved isolated cases but failed to support general-purpose, inter-agent commerce. In January 2025, the team refocused on a protocol that would:

- Embed price discovery and settlement in every request–response round-trip.

[1] Source: BillingPlatform, Business Insider
[2] Source: ProsperOps, 2025 State of FinOps Survey
[3] Source: Best Bitcoin & Crypto Payment Processor, The Block

- Remain deterministic, no invisible off-chain state; every artefact auditable.
- Support multiple treasuries from day one (EVM, Solana, Bitcoin, Lightning).
- Slot cleanly into existing HTTP tooling without reinventing headers or status codes.

Out of this effort, `h402` was born. Rather than inventing a brand-new namespace, `h402` reuses the header names and JSON schemas emerging from the broader "402 family" (notably `x402` by Coinbase and `L402` by fewsats), allowing API gateways, doc generators, and client SDKs to converge instead of fork.

2.1 Design Tenets: Chain-Plural • Deterministic • Extensible

`h402` is architected around three non-negotiable principles that shape every header field, JSON artefact, and reference implementation.

Tenet	Definition	Practical Outcome
Chain-Plural	The protocol treats multiple settlement networks, EVM chains, Solana, Bitcoin L1, Lightning, as first-class options rather than "adaptors."	A single `Quote` can expose parallel price routes; a client selects the treasury that best meets its latency, fee, and regulatory constraints. No vendor lock-in, no chain maximalism.
Deterministic	Every step, `Quote`, `Authorize`, `Broadcast`, `Validate`, `Settle`, emits an immutable artefact whose content alone suffices for replay protection, audit, and dispute resolution.	Gateways require zero hidden session state. Retries and failovers are stateless. Auditors can verify a payment flow from stored JSON and on-chain proofs.
Extensible	New cryptosystems, compliance payloads, or pricing mechanisms can be added via optional headers and versioned JSON fields, without breaking clients.	Future additions, ZK receipts, CBDC rails, Travel Rule envelopes, can integrate seamlessly. Older clients ignore what they don't understand.

These tenets keep `h402` lean enough for edge functions, yet robust enough for enterprise treasuries, ensuring the protocol evolves with, not against, the rapidly diversifying landscape of networks, regulations, and agentic use cases.

2.2 Why Not `x402`?

We created `h402`, short for "HTTP 402", based on the open schema designs introduced with `x402` by Coinbase.

While we maintain full compatibility with the open schema, we're building an implementation that goes far beyond what x402 currently offers:

- **Speed and Autonomy.** A design philosophy focused on performance, flexibility, and production use, not corporate alignment

- **Infrastructure Requirements.** Integration with infrastructure like Redis queues, NBXplorer, and other systems critical for real-world crypto payments

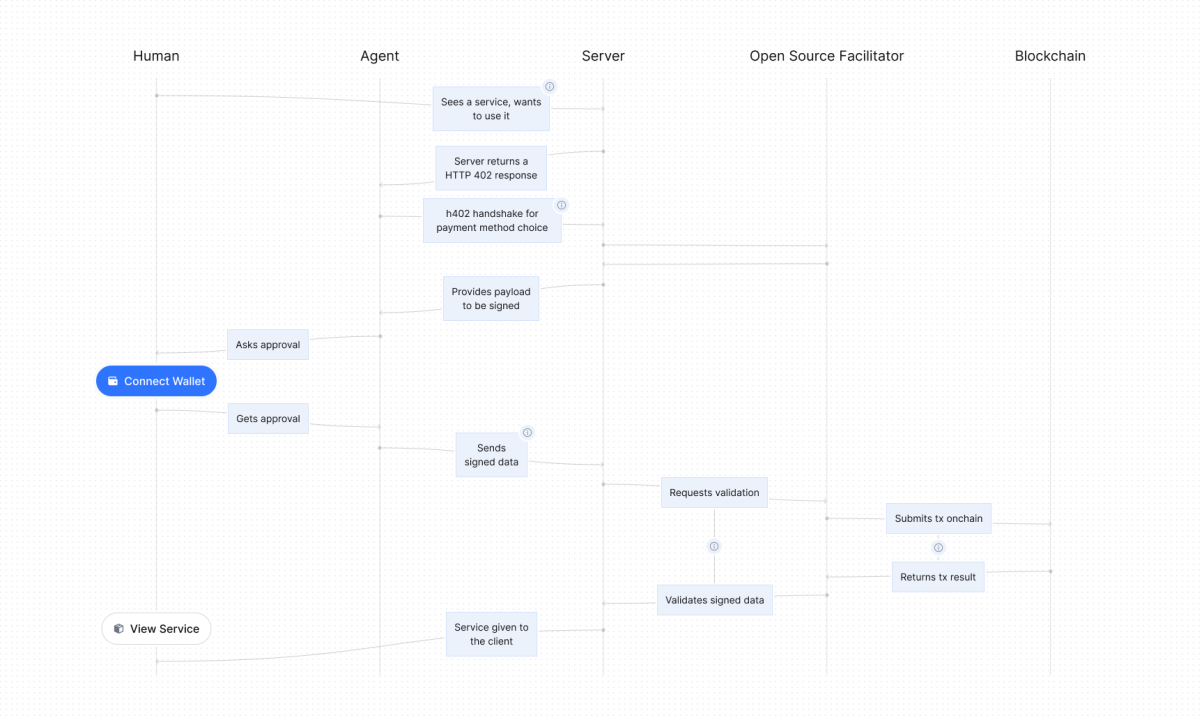
Moreover, we needed support for features that `x402` does not currently provide:

- **Non-permit Tokens.** `x402` assumes EIP-2612-style permit-based tokens, which USDC supports, but USDT and others do not.
- **Post-Broadcast Validation.** Essential for cryptocurrencies like Bitcoin, which require confirmation-based payment semantics.
- **Polling-based Systems.** Needed as fallback for payment providers and for setups based on standalone address verification, beyond signed payload + broadcast flows.

We fully respect the foundation x402 provides, but building h402 was a necessary move. Our approach is simple: preserve the protocol schema, but refuse to be limited by someone else's tech stack or product roadmap.

3 Payment Lifecycle: Quote → Authorise → *Broadcast → Validate → *Settle

The `h402` handshake distills every payment into five deterministic steps. Each step yields a machine-readable artifact that survives retries, failovers, and audits. The server ("facilitator gateway") never needs hidden session state, and the client, be it a human wallet, bot, or autonomous agent, can always resume from the last artifact on record.



h402 in action

From an autonomous AI agent's perspective, the handshake feels like a single-round-trip extension of the HTTP it already speaks. The agent hits an endpoint, receives a `402` response detailing the exact price (across several currencies), signs an authorization using

its wallet module, and rebroadcasts the request with the signature inline. Its networking loop then waits for a short "payment-settled" callback, either zero-confirmation for Lightning, or a few blocks for Bitcoin/EVM, before streaming the response payload onward.

Because each artifact (quote JSON, sig blob, tx hash, receipt) is deterministic, the agent can safely retry after crashes, migrate between hosts, or delegate settlement to another micro-service without losing budget or state. In effect, `h402` turns "fetch-then-pay" into "quote-and-settle inside fetch," enabling millisecond-latency commerce without bespoke billing APIs or human escrow.

#	Stage	Artifact (hash-stable)	Timeout	Retry-safe?
1	Quote	`quote.json` (price, expiry, treasury routes, nonce)	2–5s	Yes (same nonce)
2	Authorise	`auth.sig` (EIP-2612 permit, PSBT, delegated-spend)	10–30s	Yes (deterministic sig)
3	Broadcast	`txHash` or Lightning pre-image	Chain mempool TTL	Yes (chain rejects dupes)
4	Validate	`receipt.json` (confirmations, block height)	Configurable (0–3 conf)	Yes (idempotent lookup)
5	Settle	`settled.json` (status, FX details)	SLA-bound	Yes (stateless write)

3.1 Lifecycle Components

3.1.1 Quote

The client requests a resource. The gateway replies with HTTP `402` and a `Quote` header containing a JSON blob listing permitted payment routes (e.g., USDC/BSC, BTC/LN, SOL/Solana), unit price, expiry timestamp, and a unique nonce. The quote is cryptographically signed by the gateway so that any replica can later verify it offline.

3.1.2 Authorise

Using the quote's nonce, the client produces a deterministic authorization artifact:

- **EVM** → ERC-20 `Permit` signature
- **Bitcoin** → PSBT partial signature

- **Solana** → delegated-spend instruction

No funds move yet, this step simply proves intent and locks price/expiry. The gateway responds with `202 Accepted`, echoing the authorization hash so both sides share the same checkpoint.

3.1.3 *Broadcast

Either party may now broadcast the on-chain transaction. For permit flows, the gateway typically pushes the signed calldata; for PSBT or Lightning flows, the client does. The artifact of record is the `txHash` (or invoice pre-image), which can be independently verified.

*It's important to note that the Broadcast component is required only for post-broadcast payment validation that do not use permit-like logics.

3.1.4 Validate

`h402` requires proof of on-chain inclusion before service is delivered or rate limits are lifted. Confirmation depth is negotiable per quote (e.g., `"conf": 0` for optimistic L2s, `3` for BTC main chain). The gateway logs a `receipt.json` that binds `quoteHash`, `authHash`, and `txHash` to the observed block height.

This post-broadcast validation (not present in `x402`) enables support for Bitcoin and polling-based systems, without custodial hacks.

3.1.5 *Settle

Once the confirmation target is reached, or the timeout expires, the gateway emits a final `settled.json` (success, partial, or failed). On success, the resource is streamed, or the HTTP status is downgraded from `402` to `200 OK`.

*It's important to note that the Settle component is required only for permit-like payment validation that are not broadcasted before reaching the Validate step.

3.1.6 Timeouts & Retries

- **Quote expiry** prevents stale pricing; client simply re-requests.
- **Authorise idempotency** is guaranteed via nonce; replays return `409 Conflict`.
- **Broadcast timeout** (e.g., 60s) auto-cancels unpaid quotes.
- **Validation timeout** converts 0-conf optimism into fallback pessimism.

Because each artifact is self-describing and hash-stable, any component, edge lambda, origin server, or auditor, can reconstruct state from disk without trusting a live database. The result: a payment rail that mirrors HTTP's statelessness while satisfying chain-plural finality guarantees.

4 Current Schemas & Forward Evolution

`h402` serializes every artifact as a hash-stable JSON object (mirrored in the reference TypeScript types below). The only scheme currently implemented is `exact` (pay the precise amount once), but the envelope is designed to support future schemes, `capped`, `streamed`, `subscription`, which can reuse the same top-level fields while extending the `extra` payload.

4.1 Protocol-Level Types (Excerpt)

Core Quote & Response

```
type PaymentDetails = {
  scheme: "exact"; // will accept new schemes later
  namespace: string | null; // e.g. "evm", "bitcoin", "solana"
  networkId: string; // chain or LN network (e.g. "1", "mainnet", "testnet")
  amountRequired: bigint;
  amountRequiredFormat: "smallestUnit" | "humanReadable";
  payToAddress: string; // contract / LN invoice / SOL pubkey
  tokenAddress: string | null; // ERC-20, SPL, or native if null

  // opaque resource identifier
  resource: string;
  description: string;
  mimeType: string;

  outputSchema: object | null; // JSON Schema of the eventual payload
  requiredDeadlineSeconds: number; // seconds to settle
  extra: Record<string, any> | null; // scheme-specific extensions
};

type PaymentRequired = {
  version: 1;
  accepts: PaymentDetails[]; // multi-token, multi-chain quotes
  error: string | null;
};
```

Narrative: A gateway can return several `PaymentDetails` objects in a single quote, for example, USDC on Ethereum mainnet and BTC via Lightning, letting the client pick the optimal route. Everything needed for price discovery, risk scoring, and UX display is embedded; no side-channel lookups are required.

4.2 Authorization & Payload Wrapper

```
type PaymentPayload<T> = {
  version: 1;
  scheme: "exact";
  namespace: "evm" | "btc" | "svm" | "ln";
  networkId: string;
  payload: T; // chain-specific object (see below)
  resource: string;
};
```

`PaymentPayload` is a thin envelope that remains stable while the inner `payload` adapts to match each settlement network. This keeps gateway parsing logic consistent across new chains or compliance extensions.

4.3 EVM "Exact" Payloads (Today)

Native ETH Transfer

```
type NativeTransferPayload = {
  type: "nativeTransfer";
  signature: Hex; // EIP-1559 tx sig
  transaction: {
    from: Hex; to: Hex; value: bigint; nonce: number;
  };
};
```

ERC-20 Transfer

```
type TokenTransferPayload = {
  type: "tokenTransfer";
  signature: Hex;
  transaction: {
    from: Hex; to: Hex; value: bigint; data: Hex; nonce: number;
  };
};
```

ERC-2612 Permit (Gas-less Approval)

```
type AuthorizationPayload = {
  type: "authorization";
  signature: Hex;
  authorization: {
    from: Hex; to: Hex; value: bigint;
    validAfter: bigint; validBefore: bigint; nonce: Hex; version: string;
  };
};
```

These three variants already cover approximately 90% of EVM use cases, native fees, stablecoin payments, and meta-transactions. Each serializes deterministically, enabling the same hash guard to be reused across retries in edge functions, GPU jobs, or IoT flows.

4.4 Path to New Schemes & Networks

Coming Next	Schema Impact	Field Changes	Description
Capped	scheme: ”capped”	maxAmountRequired becomes mandatory	Quote advertises maximum spend; client locks funds once, server streams partial receipts.
Streamed	scheme: ”streamed”	extra:{unit:”token”, rate: ”100 wei”}	Gateway emits midstream meter ticks; settlement occurs when stream closes.
Subscription	scheme: ”subscription”	extra:{interval:”30d”, renewals:3}	One authorization covers recurring broadcasts; validation resets at each renewal cycle.

Each extension builds on `scheme`, `extra`, and other optional fields already present in `PaymentDetails`. Older clients ignore what they don't recognize, while newer clients unlock richer billing patterns, forward compatibility by design.

4.5 Facilitator & Verification Contracts

The facilitator utility types, `FacilitatorRequest`, `VerifyResponse`, and `SettleResponse`, remain unchanged across schemes. Only the validation logic changes (e.g., number of confirmations required, type of proof), ensuring that gateways, observability pipelines, and FinOps tools can support new payment styles through configuration, not rewrites.

5 Monetizing your first AI Agent

The quickest path to a working `h402` integration is a thin gateway that sits in front of your HTTP service, advertises a price, verifies the client's payment artifact, and (optionally) settles the transaction on-chain.

The example below uses Node + Express and the reference `@bit-gpt/h402` package. The same pattern translates to FastAPI, Go Fiber, or edge runtimes.

These implementations are subject to change. Please visit our GitHub repository to get up-to-date examples.

5.1 Install the SDK

```
npm install @bit-gpt/h402
```

5.2 Declare the Price Quote

```
// payment-config.ts
import { PaymentDetails } from "@bit-gpt/h402/types";

export const paymentDetails: PaymentDetails = {
  description: "AI-API Access",
  price: { currency: "USDT", amount: "0.10", network: "bsc" },
  recipient: "0xYourWalletAddress",
  expiresIn: 900          // quote valid for 15min
};
```

Everything the client needs, amount, token, chain, expiry, is captured in one deterministic object.

5.3 Wire Up Three Endpoints

```
// server.ts  (  50 lines in total)
import express from "express";
import { verify, settle } from "@bit-gpt/h402/facilitator";
import { isH402PaymentValid } from "@bit-gpt/h402/middleware";
import { paymentDetails } from "./payment-config";

const app = express();
app.use(express.json());

// 1  Verification: confirms sig & tx hash
app.post("/api/verify", async (req, res) => {
  const { payload } = req.body;
  const result = await verify(payload, paymentDetails);
  return res.json({ data: result });
});

// 2  Settlement: pushes tx on-chain if you hold the key
app.post("/api/settle", async (req, res) => {
  const { payload } = req.body;
  const result = await settle(payload, paymentDetails, process.env.PRIVATE_KEY);
  return res.json({ success: true, txHash: result.txHash });
});

// 3  Protected resource: auto-402 if unpaid
app.get("/api/premium-data", async (req, res) => {
  const paymentHeader = req.header("402-Payment");

  if (!paymentHeader)
    return res.status(402).json({ error: "Payment Required", details: paymentDetails });

  const ok = await isH402PaymentValid(paymentHeader, paymentDetails, {
    verifyEndpoint: "http://localhost:3000/api/verify"
  });

  if (!ok) return res.status(402).json({ error: "Invalid Payment" });

  return res.json({ data: "  Your premium content here" });
});
```

```
app.listen(3000);
```

Key idea: The resource route does not store any hidden session state, everything it needs is encoded in the client's `402-Payment` header plus the deterministic quote.

5.4 Client Flow

```
import { createPayment } from "@bit-gpt/h402";

const walletClient = /* Wagmi / Viem wallet instance */;

// 1. Produce header from the quote
const paymentHeader = await createPayment(paymentDetails, { evmClient: walletClient });

// 2. Call the premium API
const res = await fetch("http://localhost:3000/api/premium-data", {
  headers: { "402-Payment": paymentHeader }
});
console.log(await res.json());
```

5.5 Sequence Recap

- Server returns 402 + deterministic quote.
- Client signs and sends the `402-Payment` header.
- Server verifies and, optionally, settles on-chain.
- Content is streamed once `verify()` reports a valid proof.

This minimalist gateway demonstrates that pay-per-use crypto billing fits inside a single microservice, no PCI vault, no OAuth server, and no custom credit system required.

From here, you can plug the same facilitator into edge runtimes, add Prometheus hooks, or swap in Lightning and Solana adaptors without touching the protected route's core logic.

6 Agent Commerce: A New Market Layer for Code

For the first time since HTTP's birth, software entities, AI agents, can quote, pay, and consume a service without pausing for human approval. `h402` turns that capability into an economic substrate: an *agent commerce layer*, where every API call, data packet, or watt-second of energy can be priced, traded, and settled by code.

Below is a non-exhaustive map of what that unlocks today, and where it is already emerging.

6.1 Instant-Settlement Service Calls

What agents buy	Example price point	Why it matters
LLM inference bursts	$0.002 per 1K tokens	Agents compose specialised models on demand, paying only for the responses kept.
Real-time market data	$0.02 per options quote	Trading bots synthesise bespoke feeds instead of subscribing to full exchanges.
Per-frame video analytics	$0.005 per image	Drones pay edge nodes for object detection mid-flight, deciding routes in real time.

Today, these payments rely on pre-loaded credits or card vaults. `h402` collapses *quote → settle* into the request itself, allowing thousands of agents to spin workloads up and down every second.

6.2 Compute, Storage & DePIN Micro-Markets

Decentralised GPU marketplaces like Hyperbolic or io.net already expose $0.40–$0.70 per GPU-minute spot prices, up to 75% below hyperscaler on-demand rates. With `h402`:

- **Agent-driven spot bidding:** LLM swarms can auction off 30-second GPU slices, locking capacity the moment a Lightning or SOL payment hits the mempool.
- **Elastic storage:** Context-hungry agents pay per GB-hour as they expand vector stores, relinquishing chunks automatically when budget envelopes close.
- **DePIN rewards:** Edge devices stream telemetry and collect instant satoshi payouts via Lightning, already validated in today's DePIN networks.

6.3 Context & Content on Demand

Fine-tuned assistants thrive on proprietary knowledge but choke on blanket subscriptions. An `h402`-enabled content provider can expose:

- Per-document legal archives at $0.10 per filing
- Premium news paragraphs priced by the character, unlocking just the cited snippet
- Scientific datasets purchased by the feature column, not the whole CSV

Academic studies and Lightning pilots show frictionless sub-cent micropayments can increase conversion rates and reduce subscription fatigue.[4]

6.4 Device-to-Device & Physical-World Autonomy

- EV charging robots negotiate spot energy with charging pads, paying per kilowatt as the car parks.
- Smart factories let CNC machines rent tool heads from neighbouring cells, settling usage-based wear costs every few minutes.
- Logistics drones stream air-corridor tolls to urban UTM servers, paying per metre flown; fees adjust dynamically with weather and traffic.

[4] See relevant discussion on arXiv and Medium pilot summaries.

These interactions demand millisecond quotes and deterministic receipts, conditions `h402` was purpose-built to meet.

7 Conclusion

The web's original blueprint left a monetary slot unfilled. Three decades later, the rise of autonomous software makes that omission impossible to ignore.

`h402` is our answer: a stateless, chain-plural, and extensible payment primitive that nests natively inside HTTP. By collapsing price discovery, authorisation, broadcast, validation, and settlement into a single, deterministic handshake, the protocol frees agents, humans, and services from the slow detours of card vaults, dashboards, and proprietary credit systems.

What follows is more than cheaper payments. `h402` lays the tracks for agent commerce: LLM swarms leasing GPUs by the second, drones bartering airspace in real time, knowledge bases selling paragraphs instead of subscriptions, and micro-factories clearing invoices machine-to-machine every few minutes.

Each of these interactions inherits the protocol's auditability, compliance hooks, and pick-your-chain optionality, attributes that let enterprises adopt it with confidence while innovators extend it toward yet-unimagined rails.

Every new integration brings us closer to a web where value moves at the same speed, and with the same openness, as information.

h402

by BitGPT