CONFIDENTIAL

UNITED KINGDOM INTELLECTUAL PROPERTY OFFICE

PATENT APPLICATION

Applicant

The Bitcoin Corporation Ltd

Inventor

Richard Boase

Date of Preparation

2 March 2026

Title of Invention

Autonomous Discovery, Evaluation, Acquisition, and Re-Serving of Tokenized Web Content by Networked Software Agents Using a URL-Embedded Monetary Signalling Convention

Field of the Invention

The present invention relates to systems and methods for autonomous software agents to discover, evaluate, acquire, and redistribute tokenized digital content on the World Wide Web. More particularly, the invention concerns a network of autonomous agents that crawl the open web for URLs containing a predetermined monetary signalling prefix character, autonomously evaluate whether the tokenized content at such URLs represents a profitable acquisition opportunity, acquire bearer tokens representing access rights and economic interests in the content, re-serve the content to downstream requesters for revenue, and propagate discovered token addresses to peer agents via a gossip protocol, thereby forming a self-organising

distribution network in which token holders are economically incentivised to serve content and earn dividends on downstream transactions.

Background of the Invention

Problem Statement

The World Wide Web lacks a native mechanism for content to signal its own economic availability. Existing approaches to web monetisation suffer from several deficiencies:

1. **No Pre-Request Price Discovery** — When a web resource requires payment, the client must make a request and receive a rejection (typically HTTP 402 Payment Required) before learning that payment is needed. There is no convention by which a URL itself signals, prior to any HTTP request, that the resource at that URL is economically active — that is, available for tokenized purchase, tradeable, and capable of generating returns for holders. Clients, whether human or autonomous software agents, cannot distinguish economically active URLs from ordinary URLs by inspection alone.
2. **No Autonomous Acquisition Decision-Making** — Even where payment-gated content exists, there is no mechanism for an autonomous software agent to evaluate whether acquiring a token for a given piece of content is economically rational. Existing systems assume a human user who decides whether to pay based on subjective interest. No system enables a software agent to make a quantitative acquisition decision based on projected revenue from re-serving the content to others — that is, to treat the token as an investment rather than a consumption expense.
3. **No Holder-as-Distributor Model** — Existing digital content distribution models separate the roles of consumer and distributor. A user who pays for content consumes it; distribution is the responsibility of the content creator or a centralised platform. There is no system in which acquiring a content token automatically enrols the acquirer as a node in a distribution network, such that the acquirer earns revenue by re-serving the content to subsequent requesters, and earns dividends proportional to their stake from all downstream transactions in perpetuity.
4. **No Distributed Content Discovery for Autonomous Agents** — Web crawlers (e.g., search engine crawlers) discover and index content for human search. No crawler or agent network is designed to discover content that is economically active — that is, content which can be acquired as a tradeable token, re-served for profit, and staked for dividends. There is no gossip protocol by which autonomous agents propagate discoveries of economically active content addresses to peer agents, enabling the

network to collectively build a distributed index of monetizable web resources without central coordination.

5. **Content as Expense Rather Than Investment** — All existing micropayment and content-gating systems treat the payment as a consumption expense: the user pays, receives the content, and the transaction is complete. There is no system in which the payment results in the acquisition of a bearer token that: (a) can be traded on secondary markets; (b) can be staked to earn dividends from future sales of the same content; (c) entitles the holder to re-serve the content and earn revenue from each serve event; and (d) appreciates in value as demand for the content increases, because the pricing model is algorithmic and supply-dependent.
6. **No Redundancy Through Economic Incentive** — Centralised content hosting creates single points of failure. Content delivery networks (CDNs) provide redundancy but are operated by centralised corporations. There is no mechanism by which content redundancy emerges naturally from economic incentives — that is, a system in which holding and serving a copy of content is profitable, and therefore rational self-interest causes multiple independent agents to hold and serve the same content, producing redundancy as an emergent property of the economic model rather than as an engineered infrastructure decision.

Prior Art Limitations

Web crawlers (Googlebot, Bingbot, CommonCrawl) discover and index web content but do not evaluate economic potential or acquire tokens. Web scrapers extract data but do not participate in token economies. Payment channel networks (Lightning Network) and layer-2 payment protocols reduce transaction costs but do not enable autonomous acquisition decisions or holder-as-distributor economics. Content delivery networks (CDNs) provide redundancy but through centralised infrastructure, not through economic incentives to individual agents. BitTorrent provides peer-to-peer content distribution but without economic incentives for discovery, quality evaluation, or sustained availability. The x402 protocol (Coinbase, May 2025) addresses payment responses but operates on account-based blockchains, does not define a URL signalling convention for pre-request discovery, does not enable autonomous acquisition decisions, and does not create holder-as-distributor economics. The $402 protocol (The Bitcoin Corporation Ltd) defines the URL signalling convention and token economics but does not claim the autonomous agent behaviours of discovery, evaluation, acquisition decision-making, re-serving, and gossip propagation that are the subject of the present invention.

Cloudflare Pay-Per-Crawl (2024–2025) – Cloudflare has proposed and begun implementing a system whereby website operators can charge AI crawlers for the right to access and train on

their content, using Cloudflare's infrastructure as an intermediary. This represents server-side access control and billing – the website operator sets a price and Cloudflare enforces it. The system does not define a URL-embedded signalling convention for pre-request discovery, does not involve autonomous economic evaluation by the crawling agent, does not result in bearer token acquisition, does not create holder-as-distributor economics, and does not enable re-serving or staking. Cloudflare Pay-Per-Crawl is a centralised toll booth; the present invention creates autonomous economic agents that discover, evaluate, acquire, and redistribute.

Theta Network (US Patent 10,911,498, "Methods and systems for a decentralized data streaming and delivery network", 2021; and related patents) – Theta Network provides a decentralised content delivery network where edge nodes cache and relay video streams, earning THETA/TFUEL tokens for bandwidth contribution. However, Theta's model is bandwidth-centric: nodes are compensated for relaying streams, not for discovering and economically evaluating content. Theta does not use a URL-embedded monetary signalling convention, does not enable autonomous acquisition of bearer tokens representing content ownership, does not implement holder-as-distributor economics where serving generates revenue proportional to economic stake, and does not provide peer-to-peer gossip propagation of economically active URLs. Theta decentralises CDN infrastructure; the present invention creates autonomous agents that discover and invest in content.

Microsoft AI Negotiation Agent (US Patent Application 20230351420, "Systems and methods for autonomous agent negotiation", 2023) – Microsoft has disclosed systems for AI agents that negotiate terms on behalf of users, including price negotiation for services and content. However, Microsoft's system addresses multi-party negotiation protocols – the agent negotiates bespoke terms with a counterparty. The present invention does not involve negotiation: prices are algorithmically determined and non-negotiable. The present invention's agents make unilateral acquire/skip decisions based on autonomous profitability evaluation, acquire bearer tokens via deterministic payment, and join distribution networks. Microsoft's agents negotiate; the present invention's agents evaluate, acquire, and distribute.

No existing system combines: a URL-embedded monetary signalling convention enabling pre-request identification of economically active content; autonomous agent evaluation of acquisition profitability; bearer token acquisition creating automatic distribution network membership; revenue generation through re-serving; dividend earning through staking; and peer-to-peer gossip propagation of discovered economically active addresses.

Summary of the Invention

The present invention provides a system and method for autonomous content discovery, evaluation, acquisition, and redistribution ("the Discovery Engine") comprising:

(a) A URL-scanning discovery mechanism — Autonomous software agents ("miners" or "nodes") crawl the open World Wide Web, inspecting URLs for the presence of a predetermined monetary signalling prefix character (the dollar sign "$") in URL path segments. The presence of this prefix character in a URL path segment signals that the resource at that URL is economically active: it is available for tokenized purchase, the resulting token is tradeable, and holding the token confers economic rights including dividend entitlements from future transactions. This convention transforms URL inspection into a form of economic discovery — the agent can identify monetizable content before making any HTTP request to the server.

(b) An autonomous evaluation engine — Upon discovering a URL containing the monetary signalling prefix, the agent evaluates whether acquiring the token is economically rational. The evaluation considers: the current token price (retrieved from the server's pricing endpoint or from cached gossip data), the projected revenue from re-serving the content to downstream requesters (estimated from observed request frequency, peer demand signals, and historical serve revenue for comparable tokens), the agent's available capital, and configurable risk parameters set by the agent's operator. The evaluation produces a binary acquire/skip decision, or a confidence score that the operator can threshold. This evaluation is performed autonomously without human intervention.

(c) An autonomous acquisition mechanism — When the evaluation engine determines that acquisition is profitable, the agent autonomously constructs and submits a payment transaction to acquire one or more bearer tokens representing access rights and economic interests in the content. The acquisition is executed programmatically: the agent reads the pricing terms from the server's 402 response or discovery endpoint, constructs a blockchain transaction paying the specified price to the specified address, signs the transaction using the agent's cryptographic keys, broadcasts the transaction, and receives bearer tokens in return. The acquired tokens confer: ongoing access to the gated content, the right to re-serve the content to other agents or users, and proportional dividend entitlements from future transactions involving the same content.

(d) A re-serving revenue mechanism — Having acquired tokens, the agent joins the distribution network for that content. When other agents or users request the content, the holding agent serves a copy and receives a payment for each serve event. The agent thus earns revenue by functioning as a distribution node. The more agents that hold tokens for popular content, the more redundant the distribution network becomes — content availability and

distribution quality are emergent properties of rational self-interest, not centrally planned infrastructure.

(e) A dividend staking mechanism — Token holders may stake their tokens, registering them as active distribution commitments. Staked tokens earn proportional dividends from all future transactions involving the same content — including both primary sales (new token acquisitions) and secondary transfers. Dividends are distributed to stakers proportionally to their stake. This creates a sustained economic incentive for long-term holding and continuous content availability, as stakers who maintain high uptime and serve reliability earn greater effective returns.

(f) A peer-to-peer gossip protocol for address propagation — Agents propagate discovered economically active addresses to peer agents via a gossip protocol. When an agent discovers a new URL containing the monetary signalling prefix, or when an agent successfully acquires a token and begins serving content, it announces the discovery to connected peers. Peers evaluate the announcement and may themselves crawl the URL, evaluate the opportunity, and acquire tokens. This gossip mechanism creates a distributed, self-organising discovery network: no central index or registry is required. The network collectively discovers, evaluates, and distributes tokenized content through emergent coordination rather than central direction.

(g) A continuous operation model — The system is designed for autonomous 24/7 operation. Agents run as daemon processes on dedicated hardware or general-purpose computers, continuously: scanning the web for new economically active URLs, evaluating newly discovered opportunities, managing their portfolio of acquired tokens, serving content to requesters, collecting revenue and dividends, and gossiping discoveries to peers. The continuous operation model, combined with the economic incentives for content availability, produces a network that maintains content redundancy and availability without human supervision.

Detailed Description of the Invention

1. The Discovery Loop

The core of the invention is a six-stage autonomous loop executed continuously by each participating agent:

1.1 Discover

The agent crawls the open World Wide Web using standard HTTP requests, inspecting URLs for the presence of the monetary signalling prefix character (the dollar sign "$") in URL path segments. The crawl may be:

- **Broad crawl**: The agent follows hyperlinks from seed URLs, inspecting every encountered URL for the prefix character. This is analogous to how web search engine crawlers discover new pages, but the agent is specifically scanning for economically active URLs rather than building a text index.
- **Targeted crawl**: The agent crawls specific domains or URL patterns known or suspected to host economically active content. Targeting may be informed by: gossip announcements from peers, historical data about domains that have hosted economically active content, DNS records indicating protocol participation (e.g., TXT records at `_x-protocol.<domain>` or CNAME records at `x402.<domain>`), or `.well-known` discovery endpoints (e.g., `/.well-known/$402.json`).
- **Registry crawl**: The agent queries known registries or indexing services that catalogue economically active content. This mode serves as a bootstrap mechanism for new agents joining the network, and as a cross-verification source against open-web discoveries.

The monetary signalling prefix is syntactically distinctive: the dollar sign "$" is a legal sub-delimiter character in URLs under RFC 3986 and RFC 1738, but it is rarely used in conventional URL paths. The vast majority of URLs on the open web do not contain dollar-prefixed path segments. This scarcity is a feature: the prefix constitutes a signal that is easy to detect and has an extremely low false positive rate, because the convention of using dollar-prefixed path segments has no pre-existing widespread use on the web.

Upon encountering a URL containing the prefix (e.g., `https://www.example.com/$article-name`), the agent records the discovery and proceeds to evaluation.

1.1.1 URL Scanning Algorithm — Technical Specification

The agent parses each encountered URL in conformance with RFC 3986 (Uniform Resource Identifier: Generic Syntax). Per RFC 3986 Section 3.3, the path component of a URI is defined as a sequence of path segments delimited by the forward slash character ("/"). Each path segment is a sequence of characters from the `pchar` production rule, which includes the sub-delimiters set containing the dollar sign "$". The scanning algorithm operates exclusively on the path component after URI decomposition, ensuring that dollar signs appearing in the

query component (after “?”), the fragment component (after “#”), or the userinfo subcomponent of the authority are not matched.

The URL decomposition and scanning procedure is as follows:

```
FUNCTION scan_url(raw_url: string) -> DiscoveryResult | null:

    // Step 1: Decompose per RFC 3986 Section 3
    // URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
    parsed = parse_uri_rfc3986(raw_url)
    path = parsed.path          // e.g., "/$article-name/comments"
    scheme = parsed.scheme      // must be "http" or "https"
    authority = parsed.authority // e.g., "www.example.com"

    IF scheme NOT IN ("http", "https"):
        RETURN null             // only scan web URLs

    // Step 2: Split path into segments on "/" delimiter
    // Percent-decode each segment before inspection
    segments = split(path, "/")
    FOR EACH segment IN segments:
        decoded_segment = percent_decode(segment)

        // Step 3: Apply the monetary signalling prefix detection rule
        // A segment is economically active if and only if:
        //   (a) its first character is "$"
        //   (b) the segment contains at least one additional character
after "$"
        //   (c) the characters following "$" match the token-name
production:
        //       token-name = ALPHA *(ALPHA / DIGIT / "-" / "_")
        //
        // Regex for a single decoded path segment:
        //   ^\\$([A-Za-z][A-Za-z0-9\\-_]*)$
        //
        // This regex rejects:
        //   "$"       — bare dollar sign, no token name
        //   "$123"    — leading digit, not a valid token name
        //   "$$foo"   — double dollar, not valid
        //   "price$5" — dollar not in prefix position

        IF regex_match(decoded_segment, "^\\$([A-Za-z][A-Za-z0-9\\-
_]*)$"):
            token_slug = regex_capture_group(1)
            RETURN DiscoveryResult {
                url: raw_url,
                authority: authority,
                path: path,
                token_slug: token_slug,
                matched_segment: decoded_segment,
                discovered_at: current_timestamp_utc()
```

```
        }

RETURN null   // no economically active segment found
```

Disambiguation rules: The algorithm avoids false positives through the following specific exclusions:

1. **Query parameters**: The string `?price=$5` contains a dollar sign, but this appears in the query component, not a path segment. The algorithm operates only on path segments extracted after RFC 3986 decomposition, so query-component dollars are never inspected.
2. **Fragment identifiers**: The string `#section-$intro` contains a dollar sign in the fragment. Fragments are stripped during URI decomposition and not scanned.
3. **Environment variable patterns in URLs**: URLs such as `https://api.example.com/config/$ENV_VAR` are syntactically indistinguishable from monetary signalling URLs at the scanning stage. The agent resolves this ambiguity at the evaluation stage (Section 1.2): after discovering the URL, the agent issues an HTTP HEAD request and inspects the response. A genuine $402 resource returns a `402 Payment Required` status with the headers `X-Price`, `X-Currency`, and `X-Address`. A non-$402 resource returns a different status code (typically 200, 301, or 404), causing the agent to discard the candidate.
4. **Percent-encoded dollar signs**: The percent-encoding of "$" is `%24`. The algorithm percent-decodes each segment before applying the regex, ensuring that both literal `$` and percent-encoded `%24` forms are detected. This prevents evasion by encoding.
5. **Path segments with internal dollar signs**: A segment such as `total$amount` does not match because the dollar sign is not in the prefix position (position 0 of the segment). The regex requires `^\\$` — the dollar must be the first character.

1.2 Evaluate

The agent evaluates whether acquiring the token associated with the discovered URL is economically rational. The evaluation comprises:

Price discovery: The agent retrieves the current token price from one or more sources: - The server's HTTP 402 response headers (specifically `X-Price`, `X-Currency`, and related headers as defined in the $402 protocol); - The server's `.well-known/$402.json` discovery endpoint, which contains the full token market details including pricing model, current supply, and current price; - Cached price data received via gossip from peers who have recently

queried the same URL; - On-chain data from the blockchain, where the pricing model is declared and the current state (supply, treasury) is publicly observable.

Demand estimation: The agent estimates the demand for the content, which determines the projected revenue from re-serving. Demand signals include: - Request frequency observed by the agent's own crawler (how many other agents or users are requesting the URL); - Gossip data from peers reporting their own request observations; - Token market data (number of existing holders, recent transaction volume, velocity of new acquisitions); - Content characteristics (content type, size, category) correlated against historical demand data for comparable content.

Revenue projection: The agent computes a projected return on investment:

```
projected_revenue = estimated_demand × serve_price × holding_period
projected_roi = (projected_revenue - acquisition_cost) /
acquisition_cost
```

Where `serve_price` is the per-serve payment the agent will receive, `estimated_demand` is the projected number of serve requests over the holding period, and `acquisition_cost` is the current token price.

Risk assessment: The agent applies configurable risk parameters: - Maximum capital allocation per single acquisition; - Minimum projected ROI threshold; - Maximum portfolio concentration in any single content category; - Verification status of the token (whether the token's on-chain data has been cross-verified against multiple independent sources).

The evaluation produces a decision: acquire, skip, or watch (add to a monitoring list for future re-evaluation as market conditions change).

1.2.1 Evaluation Engine — Complete Pseudocode

The evaluation engine takes a `DiscoveryResult` and the agent's current portfolio state as inputs and produces a scored decision. The complete algorithm is specified below:

```
STRUCT EvaluationInput:
    discovery: DiscoveryResult
    token_price: uint64              // current price in satoshis
    token_supply: uint64             // current circulating supply
    serve_price: uint64              // per-serve payment in satoshis
    holder_count: uint32             // number of current token holders
    content_size_bytes: uint64       // size of gated content
    content_type: string             // MIME type (e.g., "text/html",
"image/png")
    agent_balance: uint64            // agent wallet balance in satoshis
    portfolio: PortfolioState        // current holdings

STRUCT EvaluationOutput:
    decision: ENUM { ACQUIRE, SKIP, WATCH }
    confidence: float64              // 0.0 to 1.0
    projected_roi: float64           // annualised ROI as decimal
    reasoning: string                // human-readable explanation

FUNCTION evaluate(input: EvaluationInput, config: AgentConfig) ->
EvaluationOutput:

    // ---- PHASE 1: Demand Estimation ----
    // Compute demand using exponential moving average (EMA) of observed
    // request rates, weighted toward recent observations.
    //
    // The EMA gives higher weight to recent data points, allowing the
    // agent to respond to trending content while dampening noise.

    alpha = config.ema_smoothing_factor   // typically 0.3 (range 0.0-
1.0)
    observed_rates = get_request_rates(input.discovery.url)
        // returns array of (timestamp, requests_per_hour) tuples
        // from the agent's own crawler logs and gossip data

    IF length(observed_rates) == 0:
        // No direct observations — use content-type heuristic
        base_demand = config.default_demand_by_type[input.content_type]
            // e.g., {"text/html": 10.0, "image/png": 5.0, "video/mp4":
20.0}
            // requests per hour
    ELSE:
        // Compute EMA over observed rates, most recent first
        sort_descending_by_timestamp(observed_rates)
        ema = observed_rates[0].requests_per_hour
        FOR i = 1 TO length(observed_rates) - 1:
            ema = alpha * observed_rates[i].requests_per_hour + (1 -
```

```
alpha) * ema
        base_demand = ema

    // Adjust demand estimate using gossip-sourced peer observations
    peer_observations =
get_peer_demand_signals(input.discovery.token_slug)
    IF length(peer_observations) > 0:
        peer_avg = mean(peer_observations.requests_per_hour)
        // Blend own observation with peer consensus (70/30 split)
        estimated_demand = 0.7 * base_demand + 0.3 * peer_avg
    ELSE:
        estimated_demand = base_demand

    // Apply holder dilution factor: more holders means each holder
    // receives a smaller share of incoming requests
    effective_demand = estimated_demand / (input.holder_count + 1)
        // +1 to include the agent itself as a prospective holder

    // ---- PHASE 2: Revenue Projection ----
    holding_period_hours = config.holding_period_days * 24
    projected_serve_revenue = effective_demand * input.serve_price *
holding_period_hours

    // Estimate dividend income over holding period
    // New acquisitions generate dividend flow; estimate from token
velocity
    token_velocity = get_token_velocity(input.discovery.token_slug)
        // transactions per hour for this token, from on-chain or gossip
data
    dividend_rate = config.dividend_pool_fraction  // e.g., 0.10 (10% of
each tx)
    avg_tx_value = input.token_price               // approximate
    dividend_per_hour = (token_velocity * avg_tx_value * dividend_rate)
/ (input.holder_count + 1)
    projected_dividend_revenue = dividend_per_hour *
holding_period_hours

    total_projected_revenue = projected_serve_revenue +
projected_dividend_revenue
    acquisition_cost = input.token_price

    projected_roi = (total_projected_revenue - acquisition_cost) /
acquisition_cost
    annualised_roi = projected_roi * (8760 / holding_period_hours)
        // 8760 = hours in a year
```

```
    // ---- PHASE 3: Risk Parameter Checks ----
    risk_flags = []

    // Check 1: Capital allocation limit
    IF acquisition_cost > config.max_single_acquisition_satoshis:
        risk_flags.append("EXCEEDS_MAX_ALLOCATION")

    // Check 2: Wallet balance threshold — never spend more than X% of
balance
    IF acquisition_cost > input.agent_balance *
config.max_balance_fraction:
        risk_flags.append("INSUFFICIENT_BALANCE_MARGIN")

    // Check 3: Portfolio concentration — max % in any single content
category
    category = classify_content(input.content_type, input.discovery.url)
    category_exposure =
input.portfolio.total_value_in_category(category)
    total_portfolio = input.portfolio.total_value()
    IF total_portfolio > 0:
        new_concentration = (category_exposure + acquisition_cost) /
(total_portfolio + acquisition_cost)
        IF new_concentration > config.max_category_concentration:
            risk_flags.append("CONCENTRATION_LIMIT")

    // Check 4: Minimum ROI threshold
    IF annualised_roi < config.min_annualised_roi:
        risk_flags.append("BELOW_ROI_THRESHOLD")

    // Check 5: Token verification — cross-reference on-chain
inscription
    IF NOT verify_token_onchain(input.discovery.token_slug):
        risk_flags.append("UNVERIFIED_TOKEN")

    // ---- PHASE 4: Decision ----
    // Confidence score combines ROI strength with absence of risk flags
    roi_score = clamp(annualised_roi / config.target_roi, 0.0, 1.0)
    risk_penalty = length(risk_flags) * 0.25
    confidence = clamp(roi_score - risk_penalty, 0.0, 1.0)

    IF length(risk_flags) > 0 AND any_flag_is_blocking(risk_flags):
        // Blocking flags: INSUFFICIENT_BALANCE_MARGIN, UNVERIFIED_TOKEN
        decision = SKIP
    ELSE IF confidence >= config.acquire_confidence_threshold:  // e.g.,
0.6
        decision = ACQUIRE
```

```
    ELSE IF confidence >= config.watch_confidence_threshold:    // e.g.,
0.3
        decision = WATCH
    ELSE:
        decision = SKIP

    RETURN EvaluationOutput {
        decision: decision,
        confidence: confidence,
        projected_roi: annualised_roi,
        reasoning: format_reasoning(risk_flags, estimated_demand,
projected_roi)
    }
```

The `AgentConfig` structure contains all operator-tunable parameters:

```
STRUCT AgentConfig:
    ema_smoothing_factor: float64          // default 0.3
    holding_period_days: uint32            // default 30
    max_single_acquisition_satoshis: uint64 // default 100,000 (0.001
BSV)
    max_balance_fraction: float64          // default 0.05 (5% of
wallet)
    max_category_concentration: float64    // default 0.30 (30%)
    min_annualised_roi: float64            // default 0.50 (50%
annualised)
    target_roi: float64                    // default 2.0 (200%
annualised)
    acquire_confidence_threshold: float64  // default 0.6
    watch_confidence_threshold: float64    // default 0.3
    dividend_pool_fraction: float64        // default 0.10
    default_demand_by_type: map<string, float64>
```

1.3 Acquire

Upon a positive acquisition decision, the agent autonomously executes the purchase:

1. The agent reads the complete pricing terms from the server's 402 response or discovery endpoint, including: the exact price, the payment address, the currency denomination, and any additional conditions.
2. The agent constructs a blockchain transaction paying the specified amount to the specified address, using unspent transaction outputs (UTXOs) from the agent's wallet.

3. The transaction is signed using the agent's private cryptographic key, which is stored in secure hardware (in the case of a dedicated hardware device) or in an encrypted key store (in the case of a software agent).
4. The signed transaction is broadcast to the blockchain network.
5. Upon confirmation, the agent receives bearer tokens representing: access rights to the gated content, membership in the distribution network for that content, and proportional dividend entitlements from future transactions.
6. The acquired tokens are recorded in the agent's local portfolio database, along with the acquisition price, timestamp, and associated content metadata.

The entire acquisition process — from price retrieval through transaction construction, signing, broadcast, and token receipt — is executed autonomously without human intervention. The agent's operator configures the evaluation parameters (risk thresholds, capital allocation) but does not approve individual acquisitions.

1.3.1 Autonomous Acquisition Transaction — Technical Specification

The acquisition transaction follows a precise sequence of HTTP interactions and blockchain transaction construction:

Step 1 — Price retrieval from 402 response headers:

The agent issues an HTTP GET request to the discovered URL. The server responds with HTTP status `402 Payment Required` and the following headers (as defined by the $402 protocol):

```
HTTP/1.1 402 Payment Required
X-Price: 1000
X-Currency: BSV
X-Address: 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa
X-Token-Id: $article-name
X-Network: mainnet
X-Memo: token_purchase:$article-name
Content-Type: application/json
```

The agent parses these headers into a `PricingTerms` structure:

```
STRUCT PricingTerms:
    price_satoshis: uint64       // parsed from X-Price (value in
smallest unit)
    currency: string             // parsed from X-Currency (e.g., "BSV")
    payment_address: string      // parsed from X-Address (Base58Check
or bech32)
    token_id: string             // parsed from X-Token-Id
    network: string              // parsed from X-Network ("mainnet" or
"testnet")
    memo: string                 // parsed from X-Memo (OP_RETURN data)
```

The agent validates the payment address by decoding the Base58Check encoding, verifying the checksum, and confirming the network prefix byte matches the declared network (0x00 for mainnet, 0x6f for testnet).

Step 2 — Transaction construction:

The agent constructs a Bitcoin SV transaction with the following structure:

```
TRANSACTION STRUCTURE:
    Version: 1
    nLockTime: 0

    INPUTS (1 or more):
        For each required input UTXO from agent wallet:
            txid: <32 bytes, hash of previous transaction>
            vout: <4 bytes, output index in previous transaction>
            scriptSig: <to be filled during signing>
            nSequence: 0xFFFFFFFF

    OUTPUTS (2 or 3):
        Output 0 — Payment to creator:
            value: pricing_terms.price_satoshis
            scriptPubKey: OP_DUP OP_HASH160 <20-byte hash of
payment_address> OP_EQUALVERIFY OP_CHECKSIG

        Output 1 — OP_RETURN token receipt:
            value: 0
            scriptPubKey: OP_FALSE OP_RETURN <protocol_prefix:
0x24343032>
                          <token_id: UTF-8 bytes> <agent_address: UTF-8
bytes>
                          <timestamp: 8 bytes, uint64 big-endian, Unix
epoch seconds>
            // The protocol_prefix 0x24343032 is the ASCII encoding of
"$402"
            // This output inscribes the token acquisition on-chain

        Output 2 — Change (if applicable):
            value: sum(input_values) - price_satoshis - miner_fee
            scriptPubKey: OP_DUP OP_HASH160 <20-byte hash of
agent_change_address> OP_EQUALVERIFY OP_CHECKSIG
```

UTXO selection follows a smallest-first strategy: the agent selects the smallest UTXOs from its wallet whose combined value meets or exceeds `price_satoshis + estimated_miner_fee`. The miner fee is estimated at 1 satoshi per 2 bytes of transaction size (0.5 sat/byte), computed after transaction construction.

Step 3 — Signing:

For each input, the agent computes the SIGHASH: 1. Serialize the transaction with the input's corresponding UTXO scriptPubKey substituted into the scriptSig field (SIGHASH_ALL, hash type 0x41 for BSV with FORKID). 2. Double-SHA256 hash the serialized transaction. 3. Sign the hash using ECDSA with the secp256k1 private key corresponding to the UTXO's

locking address. 4. Construct the scriptSig: `<DER-encoded signature || 0x41> <compressed public key (33 bytes)>`.

Step 4 — Broadcast:

The signed transaction is serialized to raw hex and broadcast via one or more of: - Direct submission to a miner's merchant API (mAPI) endpoint via HTTPS POST. - Submission to the Bitcoin SV peer-to-peer network via the `tx` message to connected nodes. - Submission to a transaction broadcast service API.

The agent waits for acknowledgement (mAPI returns a `txid` and `returnResult: "success"`) or monitors the mempool for the transaction's appearance. The agent records the broadcast `txid` and polls for confirmation in subsequent blocks.

Step 5 — Token receipt verification:

After the transaction is confirmed (included in a block), the agent verifies the token receipt by: 1. Querying the blockchain for the transaction by `txid`. 2. Confirming that Output 0 pays the correct `price_satoshis` to the correct `payment_address`. 3. Confirming that Output 1 contains the OP_RETURN inscription with the correct `protocol_prefix`, `token_id`, and `agent_address`. 4. Recording the confirmed acquisition in the portfolio database (see Section 1.3.2).

1.4 Serve

Having acquired tokens, the agent becomes a distribution node for the associated content. The agent:

1. Downloads and caches the full content associated with the token.
2. Registers its availability to serve the content, either by: announcing on the peer-to-peer gossip network that it holds and can serve the content; responding to content requests received directly from other agents or users; or registering with a content routing service that directs requesters to available servers.
3. When a request is received, the agent verifies that the requester has presented valid payment or valid tokens, then serves the content.
4. For each serve event, the agent receives a per-serve payment. The payment may be: a direct micropayment from the requester; a deduction from the requester's prepaid token balance; or a protocol-level distribution from the token's treasury.
5. The serve event is recorded in the agent's audit trail and, where the agent participates in Proof of Indexing mining, contributes to the agent's work commitment for the next mining cycle.

The re-serving mechanism transforms token holders from passive consumers into active distributors. Each holder has an economic incentive to maintain high availability (uptime) and fast response times, because these factors determine how many serve requests they receive and thus how much revenue they earn.

1.4.1 Serve Verification — Challenge-Response Protocol

Before accepting served content, a requesting agent verifies that the serving agent actually holds the token by executing a challenge-response protocol. This prevents agents from fraudulently claiming to hold tokens and serving unauthorized copies.

The protocol proceeds as follows:

```
REQUESTER                                        SERVER (claimed token
holder)
    |                                                |
    |  1. SERVE_CHALLENGE                            |
    |  {                                             |
    |    token_id: "$article-name",                  |
    |    nonce: <32 bytes, cryptographically         |
    |           random>,                             |
    |    requester_address: "1Req...",               |
    |    timestamp: <uint64, Unix epoch>             |
    |  }                                             |
    | ---------------------------------------------> |
    |                                                |
    |                          2. Server looks up    |
    |                             its UTXO holding   |
    |                             the token.         |
    |                             Signs the nonce    |
    |                             with the private   |
    |                             key that controls  |
    |                             that UTXO.         |
    |                                                |
    |  3. SERVE_RESPONSE                             |
    |  {                                             |
    |    token_id: "$article-name",                  |
    |    nonce: <echoed nonce>,                      |
    |    utxo_txid: <32 bytes>,                      |
    |    utxo_vout: <uint32>,                        |
    |    holder_pubkey: <33 bytes, compressed>,      |
    |    signature: <DER-encoded ECDSA sig of        |
    |      SHA256(nonce || token_id || timestamp)    |
    |      using holder's private key>,              |
    |    content_hash: <32 bytes, SHA256 of          |
    |      content payload>                          |
    |  }                                             |
    | <--------------------------------------------- |
    |                                                |
    |  4. Requester verifies:                        |
    |     a) nonce matches the one sent              |
    |     b) signature is valid for holder_pubkey    |
    |     c) holder_pubkey hashes to the address     |
    |        that owns utxo_txid:utxo_vout           |
    |     d) utxo_txid:utxo_vout is unspent          |
    |        (confirmed via blockchain query)        |
    |     e) the UTXO's transaction contains an      |
    |        OP_RETURN output with the $402          |
    |        protocol prefix and matching token_id   |
```

```
|     f) timestamp is within acceptable skew  |
|        (± 300 seconds)                      |
|                                             |
|  5. If all checks pass:                     |
|     Requester sends payment and receives     |
|     content. Verifies content_hash matches.  |
|                                             |
|  6. If any check fails:                     |
|     Requester aborts, optionally broadcasts  |
|     a DISPUTE message via gossip.            |
```

The nonce is a 32-byte value generated from a cryptographically secure random number generator. Its purpose is to prevent replay attacks: a signature over a stale nonce is rejected because the requester never issued that nonce. The timestamp adds a secondary freshness check.

The UTXO liveness check (step 4d) is performed by querying the blockchain (or a cached UTXO set) to confirm the referenced output has not been spent. This ensures that an agent that has transferred or sold its token cannot continue to claim holder status using a stale UTXO reference.

1.5 Earn Dividends

Token holders may stake their tokens to earn dividends:

1. **Staking registration**: The agent submits a staking transaction to the blockchain, registering specific tokens as staked. Staked tokens remain in the agent's wallet but are marked as committed to the distribution network.
2. **Dividend accrual**: As new transactions occur involving the same content — new token acquisitions by other agents, secondary market trades, serve payments — a configurable proportion of each transaction's value is allocated to the dividend pool for that content.
3. **Dividend distribution**: The dividend pool is distributed to all stakers proportionally to their stake. Distribution may occur: per-transaction (each transaction triggers an immediate micro-distribution), per-epoch (accumulated dividends are distributed at regular intervals), or on-demand (stakers claim accumulated dividends by submitting a claim transaction).
4. **Staking incentive alignment**: Stakers who maintain high availability and serve reliability earn preferential treatment in content routing, receiving more serve requests and thus more direct serve revenue in addition to their dividend income. This creates a dual incentive: dividends reward capital commitment, and serve revenue rewards operational quality.

5. **Unstaking**: Token holders may unstake at any time, though a configurable cooldown period may apply to prevent gaming (e.g., staking immediately before a large transaction and unstaking immediately after).

The dividend mechanism is the critical innovation that transforms content tokens from consumption expenses into investment instruments. A token acquired for one unit of currency may return many multiples of that unit over time through accumulated dividends, provided the underlying content continues to attract demand. Early acquirers benefit from lower acquisition prices and longer dividend accrual periods.

1.6 Gossip

Agents propagate discoveries and market intelligence to peers via a gossip protocol:

1. **New address announcement**: When an agent discovers a new URL containing the monetary signalling prefix, it broadcasts an announcement to connected peers comprising: the discovered URL, the token's identifier, the current price, and the agent's assessment of the opportunity (optional).
2. **Acquisition announcement**: When an agent acquires a token and begins serving the content, it broadcasts a service announcement comprising: the token identifier, the agent's network address, and a proof that the agent holds the token (e.g., a signed statement referencing the agent's token UTXO).
3. **Market data propagation**: Agents periodically broadcast summaries of their observations: which tokens they hold, serve request frequency for each token, revenue earned per token, and any price changes observed. This data enables peer agents to make more informed evaluation decisions.
4. **Dispute propagation**: If an agent detects a discrepancy — for example, a token's on-chain inscription data does not match the data advertised by the server — it broadcasts a dispute announcement, alerting peers to exercise caution.

The gossip protocol operates over a peer-to-peer network using encrypted connections (e.g., libp2p with Noise encryption). Messages are propagated using a publish-subscribe model (e.g., GossipSub), with topic-based channels for different message types. The protocol is designed to be resilient to partial network partitions and to converge on a consistent view of the network's collectively discovered content addresses.

1.6.1 Gossip Protocol Message Formats — Technical Specification

All gossip messages are serialized using a compact binary encoding (Protocol Buffers or equivalent). Each message has a common envelope and a type-specific payload. The

following specifies the exact message types, field layouts, and pub-sub topic names.

Common message envelope:

```
STRUCT GossipEnvelope:
    version: uint8                  // protocol version, currently 1
    message_type: uint8             // 0x01=NewAddress,
0x02=Acquisition,
                                    // 0x03=MarketData, 0x04=Dispute
    sender_peer_id: bytes[32]       // libp2p peer ID (SHA-256 of public
key)
    timestamp: uint64               // Unix epoch seconds, UTC
    ttl: uint8                      // time-to-live hop count (default
6, max 10)
    payload: bytes[]                // type-specific payload (see below)
    sender_signature: bytes[64]     // Ed25519 signature over
                                    // SHA256(version || message_type ||
sender_peer_id
                                    //        || timestamp || ttl ||
payload)
```

Message Type 0x01 — NewAddressAnnouncement:

Published to topic: `/discovery-engine/v1/new-address`

```
STRUCT NewAddressAnnouncement:
    url: string                     // full URL including scheme, e.g.,
                                    // "https://example.com/$article-
name"
    token_id: string                // extracted token slug, e.g.,
"article-name"
    price_satoshis: uint64          // observed price at time of
discovery
    currency: string                // e.g., "BSV" (max 8 bytes)
    content_type: string            // MIME type if known, else empty
    content_size_bytes: uint64      // content length if known, else 0
    discoverer_address: string      // BSV address of discovering agent
    discoverer_sig: bytes[64]       // ECDSA signature (secp256k1) over
                                    // SHA256(url || token_id ||
price_satoshis
                                    //        || timestamp)
                                    // using the discoverer's BSV
private key,
                                    // proving the discoverer controls
the
                                    // declared address
```

Message Type 0x02 — AcquisitionAnnouncement:

Published to topic: `/discovery-engine/v1/acquisition`

```
STRUCT AcquisitionAnnouncement:
    token_id: string                // token slug
    agent_address: string           // BSV address of acquiring agent
    acquisition_txid: bytes[32]     // transaction ID of the acquisition
    utxo_vout: uint32               // output index of the token receipt
    serve_endpoint: string          // URL or multiaddr where content
can be
                                    // requested, e.g.,
                                    //
"/ip4/203.0.113.5/tcp/8402/p2p/QmPeer..."
    holding_proof: bytes[]          // ECDSA signature over
                                    // SHA256(token_id || agent_address
                                    //        || acquisition_txid ||
timestamp)
                                    // using the private key that
controls
                                    // the UTXO at
acquisition_txid:utxo_vout
    content_hash: bytes[32]         // SHA-256 hash of the cached
content,
                                    // enabling requesters to verify
integrity
```

Message Type 0x03 — MarketData:

Published to topic: `/discovery-engine/v1/market-data`

```
STRUCT MarketData:
    token_id: string                // token slug
    observation_window_seconds: uint32  // period these stats cover
(e.g., 3600)
    serve_count: uint64             // number of serve events in window
    revenue_satoshis: uint64        // total serve revenue earned in
window
    price_observation: uint64       // last observed token price in
satoshis
    holder_count_estimate: uint32   // number of unique holders observed
                                    // via gossip and on-chain data
    reporter_address: string        // BSV address of reporting agent
    reporter_sig: bytes[64]         // ECDSA signature over
                                    // SHA256(token_id || serve_count
                                    //        || revenue_satoshis
                                    //        || price_observation ||
timestamp)
```

Message Type 0x04 — DisputeAlert:

Published to topic: `/discovery-engine/v1/disputes`

```
STRUCT DisputeAlert:
    token_id: string                // token slug
    dispute_type: uint8             // 0x01=inscription_mismatch,
                                    // 0x02=content_hash_mismatch,
                                    // 0x03=serve_failure,
                                    // 0x04=invalid_holding_proof
    accused_peer_id: bytes[32]      // peer ID of accused agent
    evidence_hash: bytes[32]        // SHA-256 of evidence data (stored
off-band)
    description: string             // human-readable description (max
512 bytes)
    reporter_sig: bytes[64]         // ECDSA signature over full payload
```

Pub-sub topic hierarchy:

Topic	Purpose	Typical publish rate
`/discovery-engine/v1/new-address`	New economically active URLs	Per-discovery (bursty)
`/discovery-engine/v1/acquisition`	Token acquisitions with serve endpoints	Per-acquisition
`/discovery-engine/v1/market-data`	Periodic market observations	Every 60 seconds per agent
`/discovery-engine/v1/disputes`	Dispute alerts	Rare
`/discovery-engine/v1/heartbeat`	Agent liveness signals	Every 30 seconds per agent

Messages are propagated via GossipSub with a mesh degree of D=6 (each node maintains 6 mesh connections per topic), D_low=4, D_high=12, and a heartbeat interval of 1 second. The TTL field in the envelope is decremented at each hop; messages with TTL=0 are not forwarded. Duplicate messages (same sender_peer_id + timestamp + payload hash) are dropped.

2. The Distribution Network as Emergent Property

A critical aspect of the invention is that the distribution network is not designed or centrally managed — it emerges from the individual rational decisions of autonomous agents. Each

agent independently:

- Discovers content through its own crawling and through gossip;
- Evaluates whether to acquire based on its own profit calculations;
- Serves content because doing so earns revenue;
- Stakes tokens because doing so earns dividends;
- Gossips discoveries because doing so improves the network's collective knowledge and, by extension, the agent's own opportunity set.

The result is a self-organising content distribution network with the following properties:

Redundancy: Popular content is held and served by many agents, because popular content is the most profitable to serve. Redundancy is proportional to demand — the more valuable the content, the more agents hold it, and the more resilient its distribution.

Availability: Agents operate 24/7 because content serve revenue is continuous. An agent that goes offline loses serve revenue to competitors. Economic incentives produce high availability without any centralised SLA enforcement.

Quality of service: Agents that serve content faster and more reliably receive more requests and thus more revenue. Quality competition is market-driven rather than contractually mandated.

Price discovery: The price of content tokens is set algorithmically based on supply and demand. As more agents acquire tokens for popular content, the price rises along the bonding curve, naturally moderating demand and rewarding early discoverers with capital appreciation.

Censorship resistance: Content is held by many independent agents across many jurisdictions. No single point of failure or control exists. Removing content from the network requires persuading or coercing all holding agents — a task that becomes exponentially harder as the number of holders grows.

3. Identity and Accountability

Agents participating in the distribution network are identifiable through the $401 identity protocol. Each agent's blockchain address is linked to its operator's verified identity chain. This accountability is essential for:

- **Dividend distribution**: Dividends must be routed to the correct recipients. The $401 identity chain provides a verified payout address.
- **KYC compliance**: In jurisdictions requiring Know Your Customer verification for financial transactions, the $401 identity chain provides graduated identity verification

levels (Level 1 through Level 4+) that can satisfy regulatory requirements.

- **Dispute resolution**: When disputes arise (e.g., an agent fails to serve content despite holding tokens, or serves corrupted content), the $401 identity provides accountability.
- **Reputation**: Over time, the on-chain history of an agent's behaviour — content served, uptime, dispute frequency — builds a verifiable reputation that influences peer trust decisions.

4. The Economic Innovation

The present invention's central economic innovation is the transformation of content access from a consumption model to an investment model:

Consumption model (prior art): User pays → receives content → transaction complete. The payment is an expense with no residual value.

Investment model (present invention): Agent acquires token → receives content access + distribution rights + dividend entitlements. The token is an asset with ongoing yield:

Revenue Stream	Source	Duration
Capital appreciation	Token price increases as demand grows	Until sale
Serve revenue	Per-serve payments from downstream requesters	While holding and serving
Dividends	Proportional share of all future transactions	While staked

An agent that acquires a token for content that subsequently becomes popular may earn many multiples of its acquisition cost through the combination of serve revenue and dividends, while simultaneously providing a valuable service (content distribution) to the network.

This economic model aligns incentives across all participants: - **Content creators** benefit from a distributed sales force (token holders) who are incentivised to promote and distribute the content. - **Early acquirers** benefit from lower prices and longer dividend accrual periods. - **Late acquirers** benefit from established distribution infrastructure and proven demand. - **The network** benefits from increasing redundancy, availability, and coverage as more agents discover and acquire tokens.

5. Integration with Existing Protocols

The Discovery Engine operates on top of existing protocol infrastructure:

- **$401 (Identity)**: Provides agent identity, KYC accountability, and payout address resolution.
- **$402 (Commerce)**: Provides the token market mechanics, algorithmic pricing, HTTP 402 response protocol, bearer token model, and DNS-based facilitator routing.
- **$403 (Securities)**: Provides programmable conditions for tokens that require compliance gating (e.g., securities tokens requiring accredited investor status).
- **Proof of Indexing**: The discovery, evaluation, and serving work performed by agents constitutes verifiable indexing work that can be committed to the Proof of Indexing mining mechanism, earning agents additional $402 utility tokens.

The present invention claims the autonomous agent behaviours — discovery, evaluation, acquisition, serving, dividend earning, and gossip — that operate on top of these protocol layers.

6. Implementation Architecture

A reference implementation comprises:

Agent daemon: A long-running process executing the six-stage discovery loop. The daemon manages: - A web crawler module for URL scanning; - An evaluation engine with configurable parameters; - A wallet module for autonomous transaction construction and signing; - A content cache for serving acquired content; - A gossip network client for peer communication; - A portfolio manager tracking acquired tokens, revenue, and dividends; - An audit logger recording all actions for Proof of Indexing work commitment.

Hardware form factor: The agent may run on: - A dedicated hardware device (e.g., the ClawMiner device described in related patent application GB [ClawMiner application number]) with secure element key storage; - A general-purpose computer running the agent as a background daemon; - A mobile device running a lightweight version of the agent; - A cloud virtual machine for operators seeking to run multiple agents.

Network topology: Agents connect to one another via a peer-to-peer network using libp2p, with Noise encryption for transport security, Yamux for connection multiplexing, and GossipSub for message propagation. Bootstrap nodes provide initial peer discovery; thereafter, agents discover peers through DHT (Distributed Hash Table) routing and mDNS (Multicast DNS) for local network discovery.

6.1 Portfolio Management Data Structures — Database Schema

Each agent maintains a local SQLite database (or equivalent embedded relational store) for tracking acquired tokens, revenue, dividends, and pending evaluations. The schema is

specified below:

```
-- Table 1: Acquired tokens (the agent's portfolio)
CREATE TABLE acquired_tokens (
    id                  INTEGER PRIMARY KEY AUTOINCREMENT,
    token_id            TEXT NOT NULL,          -- e.g., "article-name"
    token_url           TEXT NOT NULL,          -- full URL of the
        gated resource
    authority           TEXT NOT NULL,          -- domain, e.g.,
        "example.com"
    acquisition_txid    TEXT NOT NULL UNIQUE,   -- blockchain
        transaction ID (hex)
    utxo_vout           INTEGER NOT NULL,       -- output index of
        token receipt
    acquisition_price   INTEGER NOT NULL,       -- price paid in
        satoshis
    acquisition_ts      INTEGER NOT NULL,       -- Unix epoch seconds
        of acquisition
    content_hash        BLOB NOT NULL,          -- SHA-256 of cached
        content (32 bytes)
    content_type        TEXT,                   -- MIME type
    content_size_bytes  INTEGER,                -- size of cached
        content
    is_staked           INTEGER DEFAULT 0,      -- 1 if currently
        staked
    staked_at           INTEGER,                -- Unix epoch of
        staking tx
    staking_txid        TEXT,                   -- blockchain txid of
        staking tx
    is_spent            INTEGER DEFAULT 0,      -- 1 if UTXO has been
        spent (sold/transferred)
    spent_txid          TEXT,                   -- txid of spending
        transaction
    category            TEXT,                   -- content
        classification for concentration tracking
    created_at          INTEGER NOT NULL DEFAULT (strftime('%s', 'now'))
);

CREATE INDEX idx_tokens_token_id ON acquired_tokens(token_id);
CREATE INDEX idx_tokens_authority ON acquired_tokens(authority);
CREATE INDEX idx_tokens_staked ON acquired_tokens(is_staked) WHERE
        is_staked = 1;

-- Table 2: Revenue tracking (per-serve payment records)
CREATE TABLE serve_revenue (
    id                  INTEGER PRIMARY KEY AUTOINCREMENT,
    token_id            TEXT NOT NULL,          -- which token was
        served
    requester_address   TEXT NOT NULL,          -- BSV address of
        requester
    payment_txid        TEXT NOT NULL UNIQUE,   -- txid of serve
        payment received
```

```
    amount_satoshis     INTEGER NOT NULL,        -- payment amount
    served_at           INTEGER NOT NULL,        -- Unix epoch seconds
    requester_peer_id   BLOB,                    -- libp2p peer ID if
        known (32 bytes)
    response_time_ms    INTEGER,                 -- time to serve
        content in milliseconds
    FOREIGN KEY (token_id) REFERENCES acquired_tokens(token_id)
);

CREATE INDEX idx_revenue_token ON serve_revenue(token_id);
CREATE INDEX idx_revenue_ts ON serve_revenue(served_at);

-- Table 3: Dividend accruals
CREATE TABLE dividend_accruals (
    id                  INTEGER PRIMARY KEY AUTOINCREMENT,
    token_id            TEXT NOT NULL,           -- token earning the
        dividend
    source_txid         TEXT NOT NULL,           -- txid of the
        transaction that generated the dividend
    source_type         TEXT NOT NULL,           -- "primary_sale",
        "secondary_trade", or "serve_payment"
    accrued_satoshis    INTEGER NOT NULL,        -- dividend amount
        accrued
    accrued_at          INTEGER NOT NULL,        -- Unix epoch seconds
    claimed             INTEGER DEFAULT 0,       -- 1 if claimed via on-
        chain claim tx
    claim_txid          TEXT,                    -- txid of claim
        transaction
    claimed_at          INTEGER,                 -- Unix epoch of claim
    FOREIGN KEY (token_id) REFERENCES acquired_tokens(token_id)
);

CREATE INDEX idx_dividends_token ON dividend_accruals(token_id);
CREATE INDEX idx_dividends_unclaimed ON dividend_accruals(claimed) WHERE
        claimed = 0;

-- Table 4: Pending evaluations (watch list)
CREATE TABLE pending_evaluations (
    id                  INTEGER PRIMARY KEY AUTOINCREMENT,
    token_id            TEXT NOT NULL,           -- token slug
    url                 TEXT NOT NULL,           -- discovered URL
    authority           TEXT NOT NULL,           -- domain
    first_seen          INTEGER NOT NULL,        -- Unix epoch of first
        discovery
    last_evaluated      INTEGER NOT NULL,        -- Unix epoch of last
        evaluation
    evaluation_count    INTEGER DEFAULT 1,       -- number of times
        evaluated
    last_decision       TEXT NOT NULL,           -- "WATCH" or "SKIP"
```

```
    last_confidence     REAL NOT NULL,          -- confidence score
        0.0-1.0
    last_projected_roi  REAL NOT NULL,          -- annualised ROI at
        last eval
    last_price          INTEGER NOT NULL,       -- token price at last
        eval (satoshis)
    source              TEXT,                   -- "crawl", "gossip",
        or "registry"
    gossip_peer_id      BLOB,                  -- peer that announced
        it, if from gossip
    next_eval_after     INTEGER,               -- earliest time for re-
        evaluation
    UNIQUE(token_id, url)
);

CREATE INDEX idx_pending_next_eval ON
        pending_evaluations(next_eval_after);

-- Table 5: Wallet UTXOs (local UTXO set for the agent's wallet)
CREATE TABLE wallet_utxos (
    txid                TEXT NOT NULL,
    vout                INTEGER NOT NULL,
    amount_satoshis     INTEGER NOT NULL,
    script_pubkey       BLOB NOT NULL,
    address             TEXT NOT NULL,
    is_spent            INTEGER DEFAULT 0,
    spent_by_txid       TEXT,
    confirmed_at        INTEGER,                -- block timestamp
    block_height        INTEGER,
    PRIMARY KEY (txid, vout)
);

CREATE INDEX idx_utxo_unspent ON wallet_utxos(is_spent, amount_satoshis)
    WHERE is_spent = 0;

-- View: Portfolio summary
CREATE VIEW portfolio_summary AS
SELECT
    token_id,
    COUNT(*) AS token_count,
    SUM(acquisition_price) AS total_cost_satoshis,
    SUM(CASE WHEN is_staked = 1 THEN 1 ELSE 0 END) AS staked_count,
    (SELECT COALESCE(SUM(amount_satoshis), 0)
     FROM serve_revenue sr WHERE sr.token_id = at.token_id) AS
        total_serve_revenue,
    (SELECT COALESCE(SUM(accrued_satoshis), 0)
     FROM dividend_accruals da WHERE da.token_id = at.token_id) AS
        total_dividends,
    category
```

```
FROM acquired_tokens at
WHERE is_spent = 0
GROUP BY token_id;
```

The portfolio manager module queries these tables to compute real-time portfolio statistics for the evaluation engine: total portfolio value, per-category concentration, historical revenue rates for ROI calibration, and wallet balance from unspent UTXOs.

Brief Description of Drawings

The following drawings would accompany this application:

- **Figure 1** — System architecture diagram showing the Discovery Engine's six-stage loop: Discover → Evaluate → Acquire → Serve → Earn Dividends → Gossip, with data flows between stages.
- **Figure 2** — URL scanning mechanism showing an agent crawling the open web, identifying dollar-prefixed path segments, and classifying discovered URLs as economically active.
- **Figure 3** — Evaluation engine decision flow showing price discovery, demand estimation, revenue projection, risk assessment, and the acquire/skip/watch decision output.
- **Figure 4** — Autonomous acquisition flow showing the sequence from pricing term retrieval through transaction construction, signing, broadcast, and bearer token receipt.
- **Figure 5** — Distribution network topology showing multiple agents holding and serving the same content, with requesters being routed to available servers and payments flowing to serving agents.
- **Figure 6** — Dividend staking mechanism showing token staking, dividend pool accumulation from new transactions, and proportional distribution to stakers.
- **Figure 7** — Gossip protocol message types and propagation showing new address announcements, acquisition announcements, market data propagation, and dispute alerts flowing through the peer-to-peer network.
- **Figure 8** — Emergent redundancy diagram showing how content popularity drives agent acquisition decisions, which drives distribution node count, which drives content availability and resilience.
- **Figure 9** — Economic model comparison showing the prior art consumption model (pay → consume → done) versus the present invention's investment model (acquire → serve → earn → compound).

- **Figure 10** — Integration diagram showing the Discovery Engine operating on top of the $401, $402, and $403 protocol layers, with Proof of Indexing connecting agent work to token mining.

Initial Claims

Note: These claims are provided in sketch form for the purposes of establishing a priority date. Formal claims will be drafted and filed within 12 months in accordance with UKIPO rules.

Claim 1 — Autonomous Content Discovery via URL Monetary Signalling

A method of discovering monetizable content on the World Wide Web using autonomous software agents, the method comprising: (a) an autonomous software agent crawling the World Wide Web by following hyperlinks and inspecting encountered URLs; (b) identifying, from the URL structure alone and prior to making an HTTP request for the resource, URLs containing a predetermined monetary signalling prefix character in one or more path segments, the presence of said prefix character signalling that the resource is available for tokenized acquisition; (c) upon identification, retrieving pricing and token market data for the identified resource from one or more of: the server's HTTP response headers, a standardised discovery endpoint, cached data from peer agents, or on-chain blockchain records; (d) recording the discovered economically active URL in a distributed index maintained collectively by a plurality of agents via a gossip protocol; wherein the monetary signalling prefix character enables pre-request identification of economically active web resources, and the distributed index is constructed without central coordination through the collective crawling and gossiping activity of independent agents.

Claim 2 — Autonomous Token Acquisition Decision Engine

A method of autonomously evaluating and executing the acquisition of tokenized digital content by a software agent, the method comprising: (a) receiving, by the software agent, information about a tokenized web resource including: the current token price, the pricing model, and the current token supply; (b) estimating projected revenue from re-serving the content to downstream requesters, based on one or more of: observed request frequency, peer demand signals received via a gossip protocol, token market transaction volume, and historical revenue data for comparable content; (c) computing a projected return on investment by comparing the acquisition cost against the projected revenue over a configurable holding period; (d) applying configurable risk parameters including one or more

of: maximum capital allocation per acquisition, minimum return threshold, and maximum portfolio concentration; (e) upon a positive evaluation, autonomously constructing a blockchain payment transaction, signing the transaction using the agent's cryptographic key, and broadcasting the transaction to acquire bearer tokens; wherein the entire evaluation and acquisition process is executed autonomously by the software agent without human intervention for individual acquisition decisions.

Claim 3 — Holder-as-Distributor Content Network

A system for distributed content delivery in which content holders are economically incentivised to serve as distribution nodes, the system comprising: (a) a plurality of autonomous software agents, each capable of acquiring bearer tokens representing access rights and economic interests in tokenized web content; (b) a content serving mechanism whereby each agent that holds tokens for a given piece of content caches and serves that content to requesting agents or users in exchange for per-serve payments; (c) a dividend staking mechanism whereby token holders register tokens as staked and receive proportional dividends from future transactions involving the same content, including new token acquisitions and secondary market transfers; (d) a content routing mechanism that directs requesters to available serving agents; wherein the distribution network for any given piece of content is not centrally planned but emerges from the independent acquisition decisions of agents that have evaluated the content as profitable to serve, and wherein content redundancy and availability are proportional to demand because popular content attracts more acquirers who become additional distribution nodes.

Claim 4 — Peer-to-Peer Gossip Protocol for Economically Active Address Propagation

A method of propagating information about economically active web resources through a network of autonomous software agents, the method comprising: (a) a first agent discovering a URL containing a monetary signalling prefix character and broadcasting an announcement comprising the URL, token identifier, and current pricing data to connected peer agents via a gossip protocol; (b) a peer agent receiving the announcement, independently crawling the announced URL, evaluating the acquisition opportunity using its own evaluation parameters, and optionally acquiring tokens; (c) agents that have acquired tokens broadcasting service announcements indicating their availability to serve the associated content, the announcements comprising the token identifier, the agent's network address, and a cryptographic proof of token holding; (d) agents periodically broadcasting market intelligence comprising serve request frequency, revenue data, and price observations for tokens they hold; wherein the gossip protocol enables the network to collectively discover, evaluate, and

catalogue economically active web resources without central coordination, and wherein each agent's individual discoveries and market observations contribute to the network's collective intelligence.

Claim 5 — Content Token as Investment Instrument with Compound Returns

A method of generating compound economic returns from a digital content token, the method comprising: (a) acquiring a bearer token representing access rights and economic interests in a piece of tokenized web content, the token being acquired through an autonomous evaluation and purchase process; (b) earning serve revenue by caching and serving the associated content to downstream requesters in exchange for per-serve payments; (c) earning dividends by staking the token, the dividends comprising a proportional share of revenue from future transactions involving the same content; (d) realising capital appreciation from the algorithmic increase in token price as additional agents acquire tokens for the same content; wherein the token functions as an investment instrument with three independent return streams — serve revenue, staking dividends, and capital appreciation — rather than as a consumption expense, and wherein the holder's economic interest in the content creates a sustained incentive for the holder to maintain content availability and service quality.

Claim 6 — Self-Organising Redundant Content Distribution via Economic Incentive

A system for achieving content distribution redundancy through economic incentive rather than centralised infrastructure planning, the system comprising: (a) a tokenized content economy wherein acquiring a content token confers the right and economic incentive to re-serve the content; (b) a plurality of autonomous agents that independently discover, evaluate, and acquire content tokens based on projected profitability; (c) an algorithmic pricing mechanism wherein the token price for a given piece of content increases as more agents acquire tokens, reflecting increasing demand; (d) a revenue mechanism whereby agents earn payments for each content serve event; wherein the number of agents holding and serving any given piece of content is determined by market forces — agents acquire tokens when projected serve revenue exceeds the acquisition cost — and wherein content redundancy, availability, and geographic distribution emerge as natural consequences of rational economic behaviour by independent agents rather than as engineered infrastructure properties.

Abstract

A system and method for autonomous software agents to discover, evaluate, acquire, and redistribute tokenized digital content on the World Wide Web. Agents crawl the open web for URLs containing a predetermined monetary signalling prefix character (the dollar sign “$”) in path segments, which signals that the resource is available for tokenized purchase. Upon discovery, agents autonomously evaluate the acquisition opportunity by projecting revenue from re-serving the content against the acquisition cost, and execute purchases without human intervention when projected returns exceed configurable thresholds. Acquired bearer tokens confer ongoing access rights, distribution network membership, and proportional dividend entitlements from all future transactions involving the same content. Agents propagate discovered economically active addresses to peers via a gossip protocol, collectively building a distributed index without central coordination. The system transforms content tokens from consumption expenses into investment instruments with three return streams: serve revenue, staking dividends, and capital appreciation. Content distribution redundancy and availability emerge as natural consequences of rational economic behaviour by independent agents, rather than as centrally planned infrastructure.

Document prepared for UKIPO filing. Priority date to be established upon submission. Applicant: The Bitcoin Corporation Ltd Inventor: Richard Boase Date of preparation: 2 March 2026