Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 8 May 2026

A. Jurkovikj
4 November 2025

The Collaboration Tunnel Protocol
draft-jurkovikj-collab-tunnel-00

Abstract

This document specifies the Collaboration Tunnel Protocol, a method for efficient, verifiable content delivery between web publishers and automated agents. The protocol typically achieves up to 90% bandwidth reduction (83% median measured) through bidirectional URL discovery, template-invariant content fingerprinting, sitemap-first verification, and strict conditional request discipline.

Test Vector 2: Entity and Unicode - Input String (literal): Test & Unicode: caf迪- - Normalized String: test & unicode: caf迪- - SHA-256 (hex): f58639b586fac9cb70d4513c83a6b2954178a80f12f5c1069aad09d124ef7b24 - contentHash: sha256-f58639b586fac9cb70d4513c83a6b2954178a80f12f5c1069aad09d124ef7b24 - ETag: "sha256-f58639b586fac9cb70d4513c83a6b2954178a80f12f5c1069aad09d124ef7b24"

Table of Contents

1. Introduction

Automated agents (AI crawlers, search engines, content aggregators) increasingly consume web content at scale. Traditional HTML delivery designed for human browsers imposes unnecessary overhead: presentational boilerplate (navigation, footers, advertisements), large CSS/JavaScript bundles, and redundant fetches of unchanged content.

Existing approaches address portions of this problem:

* XML Sitemaps [XMLSitemaps] provide discovery but lack content fingerprints

* AMP [AMP] reduces HTML overhead but lacks synchronized hashing

* ResourceSync [ResourceSync] provides digest-based synchronization but lacks endpoint-level validator discipline

The Collaboration Tunnel Protocol (TCT, also referred to as "collab-tunnel") integrates these concepts into a cohesive system optimized for machine consumption while preserving human-readable canonical URLs for SEO and web compatibility.

1.1. Problem Statement

Current AI crawler behavior (2025) demonstrates:

1. *Bandwidth Waste*: Fetching full HTML documents when only core content is needed

2. *Token Overhead*: Processing boilerplate (navigation, footers) consumes 86% of tokens

3. *Redundant Fetches*: No efficient skip mechanism when content unchanged

4. *Lack of Verification*: No cryptographic proof of content delivery

Measured impact (from live deployments):

* HTML-only retrieval: 103 KB average (13,900 tokens)

* TCT JSON delivery: 17.7 KB average (1,960 tokens)

* *Savings: 83% bandwidth, 86% tokens*

1.2. Terminology

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

C-URL (Canonical URL): The human-readable URL of a web resource, typically serving HTML.

M-URL (Machine URL): A deterministically mapped endpoint serving machine-readable structured content for the same resource.

Template-Invariant Fingerprint: A cryptographic hash computed from normalized core content, stable across presentation changes.

Core Content: The primary informational content of a resource, excluding presentational boilerplate.

2. Protocol Overview

The Collaboration Tunnel Protocol consists of four coordinated mechanisms:

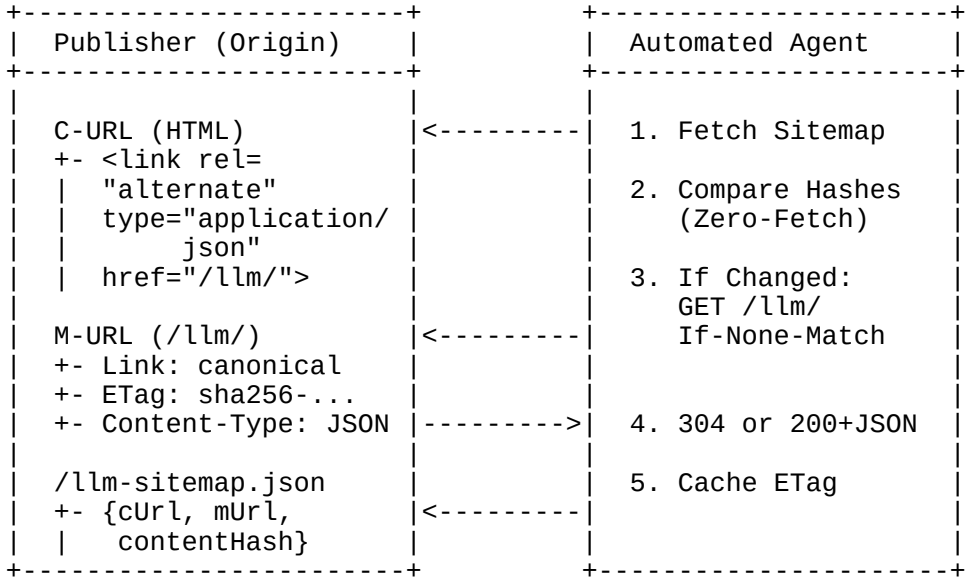
1. *Bidirectional Discovery*: Explicit C-URL <-> M-URL handshake preventing SEO conflicts

2. *Template-Invariant Fingerprinting*: Normalized content hashing for stable cache validation

3. *Conditional Request Discipline*: Strict If-None-Match precedence and 304 responses

4. *Sitemap-First Verification*: Zero-fetch skip logic when content unchanged

2.1. Architecture



3. Protocol Requirements

This section defines the normative requirements for TCT compliance.

3.1. MUST Requirements

Implementations MUST:

1. *Bidirectional Discovery*

   * C-URL HTML MUST include <link rel="alternate" type="application/json"> pointing to M-URL

   * M-URL response MUST include Link: <c-url>; rel="canonical" HTTP header

2. *Validators*

   * M-URL response MUST include ETag header

   * M-URL response SHOULD include Last-Modified header when available

   * Sitemap MUST include contentHash field for each URL

3. *Conditional Requests*

* M-URL responses MUST use strong ETags for template-invariant fingerprints: "sha256-<64 lowercase ASCII hex chars>"

* Server MUST honor If-None-Match header

* Server MUST return 304 Not Modified when ETag matches

* Server MUST give If-None-Match precedence over If-Modified-Since (see [RFC9110], Section 13.1.2)

* If-Range requires a strong validator. If the If-Range validator is weak or does not match, the server MUST ignore the Range header and send 200 OK with the full representation (per [RFC9110], Section 13.1.5)

* Servers MUST NOT assemble ranges across different encodings; ranges apply only to the selected representation variant.

4. *304 Response*

* Response MUST NOT include message body

* Servers MUST include ETag if the corresponding 200 would; otherwise SHOULD include ETag

* Servers SHOULD include Cache-Control (consistent with [RFC9110], Section 15.4.5)

5. *HEAD Support*

* Servers SHOULD support HEAD requests for all M-URLs and sitemaps

* HEAD responses MUST return the same validators and cache headers as GET and MUST NOT include a message body

* Body-dependent headers (e.g., Content-Length) MAY reflect the size of the corresponding GET response

* This behavior applies across HTTP/1.1, HTTP/2, and HTTP/3

* Implementations MUST avoid hop-by-hop headers on M-URLs and sitemaps and SHOULD NOT use transfer-codings or trailers on these responses

6. *Sitemap Parity*

- Sitemap contentHash value MUST equal M-URL ETag value (excluding quotes)
- Example: M-URL ETag "sha256-abc" -> Sitemap contentHash sha256-abc
- See "Template-Invariant Fingerprinting" and "ETag Generation" for computation and format details

7. *Canonical Verification (Agents)*

   - Agents MUST verify that the M-URL response includes Link: <c-url>; rel="canonical" and that the canonical URL matches the expected C-URL before processing
   - If the canonical link is missing or mismatched, agents SHOULD treat the endpoint as non-compliant and skip ingestion
   - If HTML <link rel="alternate"> discovery and the M-URL’s self-declared canonical conflict, agents SHOULD prefer the M-URL’s self-declared canonical and flag for operator review

8. *Protected Sitemaps*

   - Sitemaps that list protected M-URLs MUST NOT be served without access controls equivalent to those applied to the corresponding M-URLs

9. *Client Parity Behavior (Agents)*

   - Agents MUST NOT attempt to recompute the fingerprint from C-URL HTML to verify parity
   - Agents MUST verify parity by comparing the sitemap contentHash value to the M-URL ETag value (excluding quotes)
   - The comparison is a simple string equality check: contentHash === cleanETag(ETag)
   - Agents SHOULD NOT implement the normalization algorithm for compliance purposes; parity verification is sufficient

10. *Content Pages Only* - Publishers SHOULD NOT provide M-URLs for archive, category, tag, search, or date-based listing pages; where requested, servers SHOULD return 404 - Deployments with strong rationale MAY include such endpoints but MUST maintain parity semantics - Sitemaps SHOULD include only content pages (posts, articles, pages) - A homepage MAY be included only if it represents stable content

3.2. SHOULD Recommendations

Implementations SHOULD:

1. *Cache-Control*
    * Use: max-age=0, must-revalidate, stale-while-revalidate=60, stale-if-error=86400
    * Rationale: Enables revalidation with graceful stale serving
2. *Vary Header*
    * Include: Vary: Accept-Encoding
    * Rationale: Content varies by compression (gzip, br)
3. *Strong ETags*
    * Use strong ETags ("sha256-...") for TCT semantic fingerprints (per MUST requirement #3)
    * Rationale: Normalized content produces byte-identical JSON responses; strong validators ensure reliable cache compatibility
    * Note: Strong ETags MAY be used for representation-specific caching outside TCT parity (e.g., CDN byte-level caching)
4. *Content Pages Only*
    * Return 404 for M-URL requests to archive pages
    * Include only content pages in sitemap

3.3. MAY Extensions

Implementations MAY:

1. *Policy Descriptor*

- Servers MAY advertise a machine-readable policy via Link: </llm-policy.json>; rel="describedby"; type="application/json"
- The policy document format is informative and out of scope for the core protocol (see Publisher Policy Descriptor section)

2. *Additional Integrity*
   - Use Content-Digest header ([RFC9530])
   - Use HTTP Message Signatures ([RFC9421])
3. *Additional Formats*
   - Provide PDF JSON alternates
   - Provide receipt/proof systems

4. Bidirectional Discovery

Note: Path names in examples are non-normative. This specification does not require any specific URL paths. Example path "/llm/" is illustrative. Servers MAY choose alternative slugs or publish mappings. Agents MUST NOT assume a fixed path and SHOULD discover M-URLs via HTML rel="alternate", HTTP Link headers, or the JSON sitemap.

4.1. C-URL to M-URL Mapping

The C-URL MUST include an HTML <link> element in the document <head>:

```
<link rel="alternate"
      type="application/json"
      href="https://example.com/post/llm/">
```

Attributes:

- rel="alternate": Indicates alternate representation ([RFC8288])
- type="application/json": Machine-readable format
- href: Absolute or relative URL to M-URL

4.2. M-URL to C-URL Canonicalization

The M-URL response MUST include an HTTP Link header with rel="canonical":

Link: <https://example.com/post/>; rel="canonical"

This establishes bidirectional verification and prevents SEO duplication. The canonical link relation is registered by [RFC6596].

4.3. Deterministic Mapping

M-URLs SHOULD follow a deterministic pattern from C-URLs.

Non-normative examples: Append a slug to the C-URL path, such as /llm/.

Example: - C-URL: https://example.com/post/ - M-URL: https://example.com/post/llm/

Guidance: - These path patterns are examples, not protocol requirements. If a preferred slug collides with existing site routes, publishers MAY choose an alternate (e.g., /api/llm/, /content/llm/) or publish a mapping in a site-level manifest. - Agents MUST NOT assume a fixed path. Agents SHOULD discover M-URLs via HTML <link rel="alternate" type="application/json">, HTTP Link headers, or the JSON sitemap.

Migration Note: Publishers MAY use HTTP 308 Permanent Redirect to migrate from legacy paths to preferred endpoints and SHOULD list only the primary M-URL in the sitemap.

Sitemap Discovery:

Publishers SHOULD advertise the sitemap via one or both of:

1. *Link header* (RECOMMENDED) on the homepage or C-URL responses:

Link: </llm-sitemap.json>; rel="index"; type="application/json"

1. *Well-known URI* (OPTIONAL): Agents MAY check /.well-known/llm-sitemap.json. The .well-known URI convention follows [RFC8615].

Note: The well-known URI is not registered in IANA for -00; future versions may formalize this. Agents SHOULD try the Link header first, then fall back to well-known URI if needed.

4.4. Content Pages Only

Publishers SHOULD NOT provide M-URLs for archive, category, tag, search, or date-based listing pages; where requested, servers SHOULD return 404. Deployments with strong rationale MAY include such endpoints but MUST maintain parity semantics.

Rationale:

* Archive pages contain navigation and lists, not primary content

* Template-invariant fingerprinting is designed for stable content, not dynamic lists

* Archive pages change frequently as new content is published

Implementation:

* Publishers SHOULD return HTTP 404 for M-URL requests to archive pages

* Sitemaps SHOULD include only content page URLs, not archives

* Homepage MAY be included if it represents stable content

Dynamic Homepage Guidance:

If the homepage displays a dynamic content roll-up (e.g., recent posts, latest articles), publishers SHOULD prefer one of:

* Provide a synthesized stable overview representing the site (name, description, purpose)

* Include a stable "About" page as the first sitemap item instead of the homepage

Rationale: Dynamic homepages change frequently and may not provide valuable semantic content for automated agents. - CMS guidance (non-normative): For platforms like WordPress, exclude archive-like routes (e.g., category, tag, search, date, author) from M-URL handling and return 404 rather than 200 with empty payload. Ensure only singular content types (posts, pages, articles) emit M-URLs and sitemap entries.

Content page examples: - Blog posts: /blog/understanding-tct/ - Articles: /news/2025/protocol-launch/ - Static pages: /about/, /contact/

Archive page examples (should NOT have M-URLs): - Category archives: /category/technology/ - Tag archives: /tag/web-protocols/ - Date archives: /2025/10/ - Search results: /search/?q=protocol - Author archives: /author/john/

5. Template-Invariant Fingerprinting

5.1. Publisher Source of Content (Informative)

The M-URL JSON payload SHOULD be produced from the platform’s core content body (e.g., the WordPress post content field, a CMS article body), independent of the theme or template layer. This ensures the fingerprint is template-invariant.

Publishers MAY include additional semantic text in the content field to make the fingerprint sensitive to changes in those elements. For example:

* Title: Including the resource title ensures title changes produce new fingerprints

* Media descriptions: Including image alt text or <figcaption> content ensures accessibility metadata is tracked

* Deterministic order: If multiple elements are included, use a consistent order (e.g., title, blank line, main content)

Example reconstructed content:

Understanding the Collaboration Tunnel Protocol

The Collaboration Tunnel Protocol enables efficient content delivery. Diagram showing protocol flow. The protocol achieves 80-90% bandwidth reduction through conditional requests.

This approach balances template-invariance (content independent of presentation) with semantic completeness (title and media descriptions included). Publishers using this approach should ensure all included text goes through the same normalization pipeline defined in the next section.

Example of content Field Construction:

A publisher implementation might construct the content field by combining several data sources in a deterministic order.

Input Data: - Title: TCT Protocol Guide - Body Paragraph 1: The protocol is simple. - Image Alt: A flow diagram - Body Paragraph 2: It saves bandwidth.

Resulting content string in the JSON Payload:

TCT Protocol Guide

The protocol is simple.

[Image: A flow diagram]

It saves bandwidth.

This creates a readable representation that is also a stable and reliable input for the fingerprinting algorithm.

5.2. Normalization Algorithm

To generate the template-invariant fingerprint, the server MUST operate on the JSON payload’s content field (not on the C-URL HTML), applying the following steps in order:

1. Decode HTML Entities: Decode any HTML entities present in the content string (e.g., & -> &, — -> -)

2. Apply Unicode Normalization: Apply Unicode Normalization Form KC (NFKC) as defined in Unicode Standard Annex #15

3. Apply Unicode Case Folding: Convert to lowercase using the standard, locale-independent Unicode case-folding algorithm as defined in the Unicode Standard

4. Remove Control Characters: Remove all characters in the Unicode general category "Control" (Cc), which includes characters U+0000 through U+001F and U+007F through U+009F. Preserve only U+0009 (TAB), U+000A (LINE FEED), and U+000D (CARRIAGE RETURN) for subsequent whitespace collapsing.

5. Collapse Whitespace: Replace any sequence of one or more ASCII whitespace characters (U+0020 SPACE, U+0009 TAB, U+000A LINE FEED, U+000D CARRIAGE RETURN) with a single ASCII SPACE (U+0020)

6. Trim Whitespace: Remove any leading or trailing ASCII SPACE characters

7. Compute Hash: Compute the SHA-256 hash over the resulting string (encoded as UTF-8)

The strong ETag MUST be "sha256-<64 lowercase ASCII hex chars>" from this hash. The sitemap contentHash MUST be sha256-<64 lowercase ASCII hex chars> from the same hash (without the W/ prefix and quotes).

Example (pseudocode):

```
function generateFingerprint(contentString):
  normalized = contentString
    .decodeEntities()
    .unicodeNormalize('NFKC')
    .casefold()
    .removeControlChars()
    .collapseWhitespace()
    .trim()

  return "sha256-" + sha256(normalized)
```

Note: This normalization operates on the plain-text content field in the JSON payload, not on HTML from the C-URL.

Normalization uses Unicode NFKC [UAX15] and Unicode case folding [Unicode-CaseFolding]; named character references are decoded per the HTML Living Standard [WHATWG-HTML].

5.3. Deterministic JSON Serialization (Normative)

Implementations MUST use deterministic JSON serialization when generating M-URL responses to ensure that identical inputs yield byte-identical JSON.

Required Properties:

* *Stable object key order:* Lexicographic ordering by Unicode codepoint at every depth

* *Preserve array order:* Arrays MUST maintain element order as defined

* *UTF-8 encoding:* Without BOM; emit exactly one JSON document; no trailing newline

* *Compact serialization:* No pretty-print; no extraneous whitespace

* *Consistent escaping per [RFC8259]:*

  - Escape quotation mark (U+0022) and backslash (U+005C)

  - Do not escape solidus (U+002F)

  - Emit Unicode as UTF-8; use \uXXXX only where required by [RFC8259]

* *Numbers:* MUST be in minimal canonical form (no leading zeros, no "+", lowercase "e")

* *Non-finite numbers:* JSON numbers MUST NOT represent NaN or infinite values. Producers MUST NOT emit non-finite values; consumers encountering them MUST treat the payload as invalid. Values that cannot be portably represented SHOULD be encoded as strings with schema guidance

Deterministic field inclusion:

The set of fields included and their values for a given resource MUST be deterministic. Fields that vary independently of the normalized content MUST NOT be included unless they are a deterministic function of that content.

Recommendation:

Implementations SHOULD use [RFC8785] (JSON Canonicalization Scheme) or document an equivalent deterministic serialization profile to ensure cross-platform consistency.

5.4. Strong ETag and Parity (Normative)

Servers emitting strong validators MUST ensure that the ETag value changes whenever the final serialized JSON payload bytes change; equality of strong ETag values MUST imply byte-identical representations.

The 64 hexadecimal digits in the hash value MUST be lowercase ASCII.

ETag values MUST be sent as a quoted-string per [RFC9110].

5.4.1. Method A: Canonical JSON Strong-Byte (Recommended)

Computation:

1. Build the JSON response object WITHOUT the hash field

2. Canonicalize the JSON per the deterministic serialization requirements above

3. Compute F = SHA-256(canonical_json_bytes) as 64 hexadecimal characters

4. Set the hash value: hash_value = "sha256-" + F

5. Add the hash field to the JSON payload: payload.hash = hash_value

6. Set HTTP headers:

    * ETag: "sha256-" + F

    * Sitemap: contentHash: "sha256-" + F

7. Servers MUST canonicalize the final payload (now including the hash field) before sending, using the same deterministic serialization profile

Rationale:

This method guarantees strong ETag semantics even as the protocol evolves to add new fields. Any change to the JSON representation correctly changes the ETag, ensuring byte-identical validation per [RFC9110].

Computing the ETag over the canonical form of the payload without the hash field still satisfies strong validator semantics because the final payload bytes are a deterministic function of that canonical pre-hash payload and the ETag value.

Servers SHOULD compute F over identity-coded (uncompressed) canonical bytes and MAY reuse the same ETag across compressed variants; servers MUST set Vary: Accept-Encoding.

Example:

```
json_without_hash = {
  "profile": "tct-1",
  "canonical_url": "https://example.com/post/",
  "title": "Article Title",
  "content": "Normalized content text..."
}

canonical_bytes = canonicalize_json(json_without_hash)
F = sha256(canonical_bytes).hexdigest()  // 64 hex chars
hash_value = "sha256-" + F

json_with_hash = json_without_hash
json_with_hash["hash"] = hash_value

response.setHeader("ETag", '"' + hash_value + '"')
response.send(json_with_hash)
```

5.4.2. Method B: Content-Locked Strong-Content (Allowed with Restrictions)

Computation:

1. Extract and normalize content per the 6-step normalization algorithm
2. Compute F = SHA-256(normalized_content_utf8_bytes) as 64 hexadecimal characters
3. Set the hash value: hash_value = "sha256-" + F
4. Build the JSON payload deterministically from the normalized content
5. Set HTTP headers:
    * ETag: "sha256-" + F
    * Sitemap: contentHash: "sha256-" + F
    * JSON payload: hash: "sha256-" + F

Restrictions:

This method is ONLY valid if:

* The ENTIRE JSON representation is a deterministic function of the normalized content and fixed protocol constants
* NO field may vary independently of the normalized content
* Adding or changing any field REQUIRES recomputing the hash from updated content

Rationale:

If the final JSON bytes are strictly determined by content, then content-hashing produces the same result as JSON-hashing. This preserves template-invariance: same content text produces the same hash regardless of HTML/theme presentation.

Caution:

Future protocol versions that add metadata fields (e.g., language, author, published_date) independent of content text would violate strong semantics with this method. Such deployments MUST migrate to Method A.

5.5. Parity Rule (Normative)

Sitemap contentHash, JSON payload hash field, and M-URL ETag value MUST satisfy:

contentHash == clean(ETag) == payload.hash

Where clean(ETag) removes surrounding quotes from the ETag header value.

Example:

* HTTP Header: ETag: "sha256-2c26b46b68ffc68f..."

* Sitemap: contentHash: sha256-2c26b46b68ffc68f...

* JSON Payload: "hash": "sha256-2c26b46b68ffc68f..."

Verification:

Clients MUST verify parity through string equality and MUST NOT recompute hashes from HTML. Clients that detect parity violations SHOULD log a warning and MAY reject the response.

5.6. Template-Invariance

TCT’s template-invariance property means that HTML presentation changes (theme updates, CSS/JavaScript modifications, navigation restructuring) do not affect the protocol’s hash values, provided the core content (title + body text) remains unchanged.

With Method A (Canonical JSON Strong-Byte):

* HTML changes -> No effect on normalized content -> No effect on JSON -> ETag unchanged OK

* JSON field addition/change -> JSON bytes change -> ETag changes OK

* Result: Template-invariance preserved AND strong ETag semantics correct

With Method B (Content-Locked Strong-Content):

* HTML changes -> No effect on normalized content -> ETag unchanged OK

* Content changes -> Hash changes -> ETag changes OK

* Independent JSON field changes -> FAIL Would violate strong semantics (not allowed)

Summary:

Template-invariance addresses HTML/theme independence. Strong ETag semantics address JSON byte-identity. Both properties are compatible when JSON is deterministic.

6. Conditional Request Discipline

6.1. If-None-Match Precedence

When both If-None-Match and If-Modified-Since headers are present, servers MUST give If-None-Match precedence per [RFC9110], Section 13.1.2. This means:

1. Evaluate If-None-Match first

2. If ETag matches, return 304 Not Modified (ignore If-Modified-Since)

3. If ETag doesn’t match, process If-Modified-Since (if present)

Rationale: ETags provide stronger validation than modification dates, especially for semantic fingerprints.

6.2. 304 Not Modified Response

When the ETag matches the If-None-Match value:

1. Server MUST respond with 304 Not Modified

2. Response MUST NOT include a message body. Servers MUST include ETag if the corresponding 200 OK would include it; otherwise SHOULD include ETag. Servers SHOULD include Cache-Control (per [RFC9111]) and MAY include Last-Modified

Example:

```
HTTP/1.1 304 Not Modified
ETag: "sha256-2c26b46b68ffc68f..."
Last-Modified: Wed, 21 Oct 2025 07:28:00 GMT
Cache-Control: max-age=0, must-revalidate
Cache-Control: stale-while-revalidate=60
Cache-Control: stale-if-error=86400
Vary: Accept-Encoding
```

6.3. Replay and Stale Intermediaries

If an agent receives a 200 whose ETag is older than its cached parity while the sitemap contentHash matches the cached parity, it SHOULD treat the cached/sitemap state as authoritative unless the server signals otherwise (e.g., newer Last-Modified or sitemap value). The agent SHOULD revalidate end-to-end (e.g., Cache-Control: no-cache + If-None-Match) before adopting a regression.

6.4. Cache-Control Directives

M-URL responses SHOULD use:

```
Cache-Control: max-age=0, must-revalidate
Cache-Control: stale-while-revalidate=60
Cache-Control: stale-if-error=86400
```

Directives: - max-age=0: Require revalidation before serving from cache - must-revalidate: Do not serve stale without successful revalidation - stale-while-revalidate=60: Serve stale while revalidating in background (60s window) - stale-if-error=86400: Serve stale if origin unavailable (24h window)

Note: Avoid private on M-URLs and sitemaps (they are cacheable by shared caches).

6.5. Vary Header

Responses SHOULD include Vary: Accept-Encoding to indicate compression variance:

```
Vary: Accept-Encoding
```

M-URL responses primarily vary by compression (gzip, brotli), not by content type (always application/json).

Servers SHOULD NOT vary on User-Agent for M-URLs and sitemaps to preserve cacheability and reduce fragmentation.

6.6. HEAD Request Support

Servers SHOULD support HEAD requests for all M-URLs and sitemaps.

HEAD responses MUST: - Return same HTTP headers as equivalent GET request - NOT include a message body - Include all validators (ETag, Last-Modified, Cache-Control)

Example:

```
HEAD /post/llm/ HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
ETag: "sha256-abc123..."
Last-Modified: Wed, 21 Oct 2025 07:28:00 GMT
Cache-Control: max-age=0, must-revalidate, stale-while-revalidate=60
Vary: Accept-Encoding
Content-Type: application/json
Content-Length: 1234
```

This enables efficient validation without transferring the full response body.

7. Sitemap-First Verification

7.1. JSON Sitemap Format

Publishers SHOULD provide a machine-readable sitemap at a well-known location (e.g., /llm-sitemap.json).

Publishers MAY also include a Sitemap: directive in robots.txt pointing to the JSON sitemap; agents MAY use it as a discovery hint. If both XML and JSON sitemaps are present, agents that implement this protocol SHOULD prefer the JSON sitemap for TCT.

Schema:

```
{
  "version": 1,
  "profile": "tct-1",
  "items": [
    {
      "cUrl": "https://example.com/post/",
      "mUrl": "https://example.com/post/llm/",
      "modified": "2025-10-01T12:34:56Z",
      "contentHash": "sha256-2c26b46b68ffc68f..."
    }
  ]
}
```

Fields:

* version (integer): Sitemap format version (currently 1)

* profile (string, RECOMMENDED): Protocol version identifier (e.g., "tct-1"). Enables clients to detect protocol capabilities and maintain forward compatibility as the specification evolves.

* items (array): List of URL pairs

  - cUrl (string, required): Canonical URL

  - mUrl (string, required): Machine URL

  - modified (string, [RFC3339]): Last modification timestamp

  - contentHash (string, required): Template-invariant fingerprint (same as M-URL ETag)

Parity Rule:

The sitemap contentHash value MUST match the M-URL ETag header value, excluding quotes. This enables zero-fetch skip optimization: clients compare sitemap hash to cached ETag without fetching the M-URL.

Forward Compatibility:

Clients MUST ignore unknown fields in the sitemap JSON. Servers MAY add additional fields to support future protocol versions. Agents SHOULD read the profile field when present; unknown profile values SHOULD NOT cause ingestion failure but MAY be logged for analysis.

7.2. Sitemap Scalability

Publishers and agents SHOULD consider scalability for large sites:

* Publishers MAY split sitemaps into an index (a sitemap-of-sitemaps) to segment large URL sets (analogous to XML Sitemaps sitemapindex)

* Servers SHOULD support compression (e.g., Content-Encoding: gzip or br) for sitemap responses; agents SHOULD accept compressed responses

* Agents SHOULD use streaming JSON parsers for large sitemaps and enforce a maximum sitemap size (e.g., 100 MB)

* Servers SHOULD keep per-item objects compact (RECOMMENDED <= 2 KB). Servers MUST bound total sitemap size by an operator-configured budget. When budgets would be exceeded (RECOMMENDED <= 50,000 items per sitemap), servers SHOULD publish a sitemap index and split content

Sitemap Index (Large Sites):

For sites with thousands of URLs, publishers SHOULD segment sitemaps using a sitemap index (analogous to XML Sitemaps sitemapindex). A formal JSON sitemap index schema MAY be specified in future protocol versions. Agents SHOULD support consuming multiple sitemap files.

Homepage Handling:

Publishers SHOULD include the site homepage as the *first item* in the sitemap array. This provides automated agents with immediate access to site-level context (site name, description, purpose) before processing individual content pages.

For homepages that display dynamic content listings (blog roll, latest posts), publishers MAY synthesize stable content representing the site overview rather than the dynamic list.

Example with homepage first:

```
{
  "version": 1,
  "profile": "tct-1",
  "items": [
    {
      "cUrl": "https://example.com/",
      "mUrl": "https://example.com/llm/",
      "modified": "2025-10-15T08:00:00Z",
      "contentHash": "sha256-abc123..."
    },
    {
      "cUrl": "https://example.com/about/",
      "mUrl": "https://example.com/about/llm/",
      "modified": "2025-10-01T10:00:00Z",
      "contentHash": "sha256-def456..."
    }
  ]
}
```

Sitemap HTTP Response:

```
GET /llm-sitemap.json HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-Type: application/json
ETag: "sha256-sitemap-fingerprint"
Last-Modified: Wed, 21 Oct 2025 12:00:00 GMT
Cache-Control: max-age=0, must-revalidate, stale-while-revalidate=60
Vary: Accept-Encoding
Content-Length: 4567

{
  "version": 1,
  "profile": "tct-1",
  "items": [...]
}
```

Conditional Sitemap Fetch:

```
GET /llm-sitemap.json HTTP/1.1
Host: example.com
If-None-Match: "sha256-sitemap-fingerprint"

HTTP/1.1 304 Not Modified
ETag: "sha256-sitemap-fingerprint"
Cache-Control: max-age=0, must-revalidate, stale-while-revalidate=60
```

Clients SHOULD use conditional requests for sitemap to avoid unnecessary bandwidth when sitemap unchanged.

7.3. Error Handling (Informative)

Malformed sitemap: Agents SHOULD treat this as non-fatal; skip entries that fail structural validation; log and continue.

ETag/contentHash mismatch: Agents SHOULD treat the endpoint ETag as authoritative, process the change, and update caches. Publishers SHOULD publish sitemaps atomically or include Last-Modified.

M-URL unavailable: Agents MAY defer the fetch or fall back to the C-URL HTML as a last resort; publishers SHOULD return a 4xx/5xx rather than an empty 200.

HTTP Status Code Handling:

Agents SHOULD respect common HTTP status codes for retries and backoff. If a server responds with 410 Gone, the agent SHOULD treat the resource as permanently deleted. If a server responds with 429 Too Many Requests or 503 Service Unavailable, the agent SHOULD honor the Retry-After header if present.

7.4. Zero-Fetch Skip Logic

Automated agents SHOULD:

1. Fetch /llm-sitemap.json periodically
2. Compare contentHash values to locally cached hashes
3. *If hash unchanged*: Skip fetching both C-URL and M-URL (zero-fetch optimization)
4. *If hash changed*: Issue conditional GET to M-URL with If-None-Match

This enables 90%+ skip rate for unchanged content.

Example workflow:

```
Agent: Fetch /llm-sitemap.json
Agent: item.contentHash = "sha256-abc123..."
Agent: cachedHash = lookup(item.mUrl)

if (item.contentHash === cachedHash):
  // Zero-fetch: Content unchanged, skip all requests
  skip()
else:
  // Hash changed, fetch with conditional request
  GET item.mUrl
  Headers: If-None-Match: "sha256-abc123..."

  if (response.status == 304):
    // Still matched at endpoint, update cache
    cache(item.mUrl, item.contentHash)
  else:
    // Content changed, process new data
    process(response.body)
    cache(item.mUrl, response.headers['ETag'])
```

8. Publisher Policy Descriptor

This section is informative.

Publishers MAY provide a machine-readable policy descriptor at a well-known location (e.g., /llm-policy.json or /.well-known/llm-policy.json) to communicate usage terms, rate limits, and content licensing preferences to automated agents. Example paths are non-normative.

8.1. Policy Endpoint

The policy descriptor SHOULD be available at a stable URL. Example paths (non-normative):

https://example.com/llm-policy.json
https://example.com/.well-known/llm-policy.json

8.2. JSON Schema

```
{
  "profile": "tct-policy-1",
  "version": 1,
  "effective": "2025-10-01T00:00:00Z",
  "updated": "2025-10-15T12:00:00Z",

  "policy_urls": {
    "terms_of_service": "https://example.com/terms/",
    "payment_info": "https://example.com/pricing/",
    "contact": "https://example.com/contact/"
  },

  "purposes": {
    "allow_ai_input": true,
    "allow_ai_train": false,
    "allow_search_indexing": true
  },

  "requirements": {
    "attribution_required": true,
    "link_back_required": false,
    "notice_required": true
  },

  "rate_hints": {
    "max_requests_per_second": null,
    "max_requests_per_day": 10000,
    "note": "Advisory limits, honor system"
  }
}
```

Fields:

* profile (string): Policy schema version identifier (e.g., "tct-policy-1")

* version (integer): Policy revision number

* effective (string, RFC 3339): When policy took effect

* updated (string, RFC 3339): Last policy modification

* policy_urls (object): URLs to human-readable policy documents

  - terms_of_service: Legal terms URL

  - payment_info: Pricing/billing information URL (for paid access)

- contact: Publisher contact for licensing inquiries

* purposes (object): Usage permissions
  - allow_ai_input: Content may be used as AI input (RAG, context)
  - allow_ai_train: Content may be used for model training
  - allow_search_indexing: Content may be indexed for search
* requirements (object): Usage conditions
  - attribution_required: Must credit publisher when using content
  - link_back_required: Must link to canonical URL when republishing
  - notice_required: Must notify publisher of commercial use
* rate_hints (object): Advisory crawl rate limits (non-binding)
  - max_requests_per_second: Requests per second limit (null = no limit)
  - max_requests_per_day: Requests per day limit
  - note: Additional guidance

Rate hints are advisory only; enforcement, payment, and economic arrangements are out of scope for this specification. Vocabulary alignment with IETF AIPREF is expected as that work matures.

8.3. Discovery

The policy descriptor SHOULD be linked from the sitemap with a describedby Link header (example paths only):

Link: </llm-policy.json>; rel="describedby"; type="application/json"

Automated agents SHOULD:

1. Fetch /llm-policy.json before crawling
2. Honor stated usage restrictions
3. Respect rate hints to avoid overwhelming origin
4. Review terms before commercial use

This specification uses registered relations (alternate/index/ describedby). A dedicated relation for TCT sitemaps might be registered in the future; this document does not create new link relations.

8.4. Alignment with IETF AIPREF

This policy format is designed to complement the IETF AIPREF (AI Preferences) proposal, providing machine-readable expressions of publisher preferences for automated agent behavior.

9. M-URL Response Format

9.1. Content-Type

M-URL responses MUST set Content-Type: application/json:

Content-Type: application/json; charset=utf-8

9.2. JSON Payload Schema

Minimal required fields:

```
{
  "profile": "tct-1",
  "canonical_url": "https://example.com/post/",
  "title": "Article Title",
  "content": "Core article content...",
  "hash": "sha256-2c26b46b68ffc68f..."
}
```

Fields:

* profile (string, RECOMMENDED): Protocol version identifier (e.g., "tct-1"). Future versions (e.g., "tct-2") can introduce new fields while maintaining backward compatibility.

* canonical_url (string, required): The C-URL for this resource

* title (string, required): Resource title

* content (string, required): Plain-text core content (UTF-8). This field is the input to the normalization algorithm. The content field MUST NOT contain HTML tags or markup. Publishers MUST strip or escape any embedded HTML before inclusion. To ensure fingerprint stability, publishers SHOULD avoid including volatile, non-semantic elements (e.g., dynamic view counts, timestamps) in this string. Publishers SHOULD provide content independent of

template/theme presentation. Publishers MAY include semantic metadata (such as title or media captions) to create a more complete fingerprint.

* hash (string, required): Template-invariant fingerprint. MUST equal the M-URL ETag value excluding quotes. This is the same value as the sitemap contentHash field. Format: sha256-<64 hex>. Example: if M-URL ETag is "sha256-abc123", then hash is sha256-abc123 and sitemap contentHash is sha256-abc123.

Forward Compatibility:

Clients MUST ignore unknown fields in the JSON payload. Servers MAY add additional fields to support future protocol versions or domain-specific metadata. Agents SHOULD read the profile field when present; unknown values SHOULD NOT cause ingestion failure but MAY be logged.

Extended fields example:

```
{
  "profile": "tct-1",
  "canonical_url": "https://example.com/post/",
  "title": "Article Title",
  "language": "en-US",
  "published": "2025-10-01T10:00:00Z",
  "modified": "2025-10-15T14:30:00Z",
  "content": "Core article content...",
  "hash": "sha256-2c26b46b68ffc68f...",
  "structured_data": {
    "@context": "https://schema.org",
    "@type": "Article",
    "headline": "Article Title"
  }
}
```

9.3. Complete Response Example

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
ETag: "sha256-2c26b46b68ffc68f..."
Link: <https://example.com/post/>; rel="canonical"
Last-Modified: Wed, 15 Oct 2025 14:30:00 GMT
Cache-Control: max-age=0, must-revalidate
Cache-Control: stale-while-revalidate=60
Cache-Control: stale-if-error=86400
Vary: Accept-Encoding
Content-Length: 1234

{
  "profile": "tct-1",
  "canonical_url": "https://example.com/post/",
  "title": "Understanding the Collaboration Tunnel Protocol",
  "language": "en",
  "published": "2025-10-01T10:00:00Z",
  "modified": "2025-10-15T14:30:00Z",
  "content": "The Collaboration Tunnel Protocol enables...",
  "hash": "sha256-2c26b46b68ffc68f..."
}
```

10. Operational Considerations (Informative)

This section provides non-normative operational guidance for deployers and automated agents.

Migration and Path Collisions - If a preferred slug (e.g., /llm/) collides with existing routes, choose an alternate (e.g., /api/llm/, /content/llm/) and publish a 308 Permanent Redirect from legacy to primary endpoints - List only the primary M-URL in the sitemap to avoid duplication; agents SHOULD follow redirects but rely on sitemap entries for canonical endpoint discovery - Ensure robots.txt permits the chosen machine paths as appropriate for your policy

Caching and CDNs - Configure intermediaries to honor validators and serve 304 responses; avoid private on M-URLs and sitemaps to enable shared caching - Enable compression (gzip or br) for sitemaps and JSON payloads

Discovery and Verification - Prefer discovery via HTML <link rel="alternate" type="application/json">, HTTP Link headers, or sitemap entries; agents MUST NOT assume a fixed path - Agents SHOULD verify the M-URL’s canonical link header before processing and skip ingestion if missing or mismatched

Network and WAFs - Allow the HEAD method (some WAFs block by default) to enable efficient validation - If using an edge worker/ proxy, ensure required headers (Link, ETag, Cache-Control, Vary) are preserved or injected consistently

11. Security Considerations

11.1. HTTPS and TLS

M-URLs and sitemaps MUST be served over HTTPS. Agents MUST validate TLS certificates using platform trust stores and MUST reject invalid certificates.

11.2. Rate Limiting

Servers MAY expose RateLimit fields ([RFC9448]); agents SHOULD respect them.

11.3. Content Integrity

The template-invariant fingerprint (SHA-256) provides: - *Tamper detection*: Agents can verify content integrity - *Cache validation*: Ensures served content matches expected hash

However, SHA-256 alone does NOT provide: - *Authentication*: Does not prove publisher identity - *Non-repudiation*: Publisher can deny serving content

For authenticated content delivery, publishers MAY implement digital signatures (outside scope of this specification).

11.4. Privacy

Sitemap exposure MAY reveal: - *Content inventory*: All published URLs - *Modification patterns*: Publishing/update frequency

Publishers SHOULD: - Apply access controls if sitemaps contain sensitive URLs - Use robots.txt to restrict crawler access if needed

The content field may contain sensitive or personally identifiable information (PII). Publishers MUST apply access controls to M-URLs equivalent to those applied to the corresponding C-URLs, and to any sitemaps listing protected resources (see "Protected Sitemaps" in MUST Requirements).

For general privacy considerations related to Internet protocols, see [RFC6973].

11.5. Denial of Service

11.5.1. Sitemap Abuse

Large sitemaps MAY be used for DoS attacks. Agents SHOULD: - Implement request rate limiting - Set maximum sitemap size limits (e.g., 100 MB) - Use streaming JSON parsers for large sitemaps

11.5.2. HEAD vs GET Bandwidth

HEAD requests provide DoS mitigation by enabling validation without body transfer: - HEAD request: ~500 bytes (headers only) - GET request: 1-100 KB (full JSON body)

Agents SHOULD use HEAD for validation before GET.

11.6. Injection Surface Reduction

M-URL JSON responses contain only structured text, reducing injection attack surface compared to HTML: - No <script> tags or inline JavaScript - No CSS injection vectors - No HTML parsing ambiguities

However, agents MUST: - Validate JSON syntax before processing - Sanitize content before rendering to users - Treat URLs in JSON as untrusted input

11.7. Cache Poisoning

Shared caches that ignore validators can serve stale or mixed content. Publishers SHOULD include Cache-Control: must-revalidate; agents SHOULD verify the endpoint ETag against cached values and MUST NOT treat mismatched ETags as fresh.

11.8. Fingerprint Collision and Normalization Variance

SHA-256 fingerprints provide practical collision resistance. The greater risk is normalization variance. Implementations MUST apply the baseline normalization consistently; agents MUST NOT infer byte identity from strong ETags.

11.9. Content Provenance and Origin Authentication (Optional)

For authenticated origin verification of content representations, see HTTP Message Signatures [RFC9421].

Publishers MAY enhance content authenticity using:

Content-Digest ([RFC9530]):

Provides base64-encoded digest over the representation body.

Content-Digest: sha-256=:X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

Note: The value is base64 encoding of the binary SHA-256 digest per [RFC9530].

HTTP Message Signatures (RFC 9421): ~~~http Signature: sig1=:MEUCIQDXlI...; Signature-Input: sig1=("content-digest" "content-type"); created=1618884473; keyid="key-1" ~~~

These mechanisms are OPTIONAL extensions outside TCT core requirements.

11.10. Privacy and PII

M-URL JSON MAY contain personally identifiable information (PII) or sensitive content. Publishers SHOULD: - Apply same access controls as C-URL HTML - Respect user privacy preferences - Comply with applicable data protection regulations (GDPR, CCPA, etc.)

Agents SHOULD: - Honor robots.txt and meta robots directives as a site-wide policy across resources - Respect HTTP authentication requirements - Not expose cached content beyond publisher-specified Cache-Control directives

11.11. Access Control

Publishers MAY restrict M-URL access using standard HTTP mechanisms: - HTTP Authentication (Basic, Bearer, etc.) - IP allowlisting - Rate limiting per client - Geographic restrictions

Access controls SHOULD be consistent between C-URL and M-URL for the same resource.

Authenticated Content Caching:

Agents that access authenticated M-URLs MUST NOT store the resulting responses in caches that could be accessed by other unauthenticated parties. Authenticated content MUST be stored in a private, credential-scoped cache.

Authenticated Sitemaps (Informative): Publishers that protect M-URLs MAY apply equivalent access controls to sitemap endpoints. Agents SHOULD obtain and reuse credentials to access protected sitemaps. Sitemaps MUST NOT disclose protected resources without equivalent access controls.

Security Note: Authenticated M-URLs MUST NOT leak protected content via publicly accessible sitemaps. Publishers MUST either exclude protected resources from sitemaps or apply equivalent access controls to the sitemap itself.

12. Energy Efficiency Considerations

Summary: Network transmission savings are direct and measurable. The impact on AI inference energy depends on model architecture and deployment specifics and is presented for illustrative context.

12.1. Overview

The Collaboration Tunnel Protocol’s bandwidth and token reduction directly translates to significant energy savings across network infrastructure and AI model inference operations. This section quantifies the environmental impact of TCT deployment at scale.

12.2. Network Energy Consumption

Data transmission consumes energy at every network layer: routers, switches, content delivery networks (CDNs), data centers, and end-user devices. Current estimates for network energy intensity range from 0.03 kWh/GB (fixed networks) to 0.14 kWh/GB (mobile networks), with a conservative industry standard of 0.06 kWh/GB for mixed-mode transmission.

TCT bandwidth reduction (measured): - HTML-only retrieval: 103 KB average - TCT JSON delivery: 17.7 KB average - *Reduction: 85.3 KB per fetch (83% savings)*

Energy savings per fetch: - Bandwidth saved: 85.3 KB = 0.0000853 GB - Energy saved: 0.0000853 GB x 0.06 kWh/GB = *0.0000051 kWh* (0.0051 Wh) per fetch

Scaled impact (1 million fetches/day): - Daily energy savings: 5.1 kWh (5,100 Wh) - Annual energy savings: 1,861.5 kWh - *Carbon equivalent*: ~930 kg CO2 avoided (assuming 0.5 kg CO2/kWh grid average)

12.3. AI Inference Impact (Informative Summary)

TCT reduces tokenized payloads by ~86% through normalized JSON delivery. The impact on inference energy depends on model size, hardware, batching, and caching; therefore specific kWh figures vary significantly across deployments. See Appendix "Energy Methodology" for an illustrative, non-normative scenario and assumptions.

12.4. Sitemap-First Zero-Fetch Optimization

Beyond per-fetch savings, TCT's sitemap-first verification enables complete request elimination when content is unchanged.

Measured skip rate: 90%+ for unchanged content

For a typical deployment with 1,000 URLs checked daily: - Traditional crawler: 1,000 full fetches/day - TCT deployment: ~100 fetches/day (90% skipped via sitemap comparison) - *Additional savings: 900 fetches avoided*

Combined energy impact: - Network: Fewer full responses reduce transmission energy proportionally to bytes avoided - Computation: Fewer fetches and elimination of local rendering/extraction reduce crawler CPU/memory work - Inference: Reduced token processing (impact varies by model and deployment; see Energy Methodology appendix)

12.5. Comparison to Existing Approaches

Method	Avg Size	Tokens	Bandwidth Reduction
Full HTML page	350 KB	47,000	Baseline
HTML body only	103 KB	13,900	71%
AMP HTML	52 KB	7,000	85%
TCT JSON	*17.7 KB*	*1,960*	*95%*

Table 1

Energy impact per fetch depends on network infrastructure, model architecture, and deployment patterns. See Energy Methodology appendix for illustrative calculations.

12.6. Cumulative Environmental Impact

TCT's bandwidth reduction (typically 80-90%) and elimination of client-side rendering translate to measurable energy savings at scale. Actual impact depends on:

* Deployment scope (number of sites, request volume)

* Network infrastructure efficiency

* Model provider’s hardware and batching strategies

* Grid carbon intensity in serving regions

Measured improvements: - 95% smaller payloads (measured across production deployments) - 90%+ fetch elimination via sitemap-first logic - 94% reduction in crawler CPU/memory usage

Organizations deploying TCT should measure baseline energy consumption and monitor post-deployment savings specific to their infrastructure. See Energy Methodology appendix for example calculation approaches.

12.7. Relationship to IETF GREEN Working Group

The IETF Getting Ready for Energy-Efficient Networking (GREEN) Working Group focuses on infrastructure-level energy optimization through monitoring, measurement, and network-wide traffic optimization.

TCT complements GREEN’s infrastructure focus by addressing *application-layer efficiency*:

* GREEN targets: Device power consumption, network-wide traffic flow optimization

* TCT targets: Content delivery efficiency, redundant fetch elimination, token cost reduction

Together, these approaches create a comprehensive energy reduction strategy spanning both network infrastructure (GREEN WG) and application protocols (TCT).

12.8. Recommendations for Implementers

Publishers deploying TCT SHOULD:

1. *Monitor metrics*: Track bandwidth savings, 304 hit rates, and skip rates

2. *Report impact*: Document energy savings for sustainability reporting

3. *Optimize aggressively*: Minimize JSON payload size to maximize efficiency

4. *Promote adoption*: Encourage crawler operators to implement sitemap-first logic

Automated agents consuming TCT endpoints SHOULD:

1. *Implement zero-fetch*: Use sitemap hashes to skip unchanged content

2. *Respect 304 responses*: Honor conditional request discipline

3. *Cache aggressively*: Store ETags and avoid redundant fetches

4. *Measure savings*: Track bandwidth, token, and energy reduction

12.9. Future Work

Potential enhancements for energy optimization:

* *Delta encoding*: Transmit only content changes instead of full payloads

* *Push notifications*: WebSub or IndexNow integration to eliminate polling

* *Compression*: Brotli/gzip to further reduce transmission size

* *Edge caching*: CDN-level 304 responses to reduce origin load

13. IANA Considerations

This document has no IANA actions.

Future versions may register:

1. *Well-known URI*: /.well-known/llm-sitemap.json for sitemap discovery

2. *Profile URI*: Identifying the JSON payload/sitemap format (e.g., https://llmpages.org/profile/tct-1)

3. *Link Relation*: If rel="index" with type="application/json" proves insufficient, a dedicated relation may be registered

Note: The protocol name and URL slug convention (/llm/) is a non-normative example; implementations choose their own paths.

14. Comparison to Prior Art

14.1. ResourceSync

ResourceSync [ResourceSync] provides sitemap-based synchronization. It includes content digests (md5, sha-256) in ResourceSync Change Lists.

Key differences:

1. *Endpoint validator*: ResourceSync does NOT specify using the digest as endpoint ETag

2. *Handshake*: No bidirectional C-URL <-> M-URL discovery

3. *Zero-fetch*: No specification for skipping endpoint fetch when sitemap hash matches

4. *Precedence discipline*: No requirement for If-None-Match precedence

TCT integrates the SAME hash in both sitemap AND endpoint ETag, enabling zero-fetch optimization.

Unlike ResourceSync, which provides digests over entire representation files, TCT provides a semantic fingerprint of the textual content payload, decoupling it from a specific file or HTML representation.

14.2. AMP

Accelerated Mobile Pages [AMP] provides: - C-URL -> AMP-URL mapping via <link rel="amphtml"> - Lighter HTML (no JavaScript allowed)

Key differences:

1. *Format*: AMP uses HTML; TCT uses JSON

2. *Fingerprinting*: AMP has no content hashing

3. *Sitemap*: AMP sitemaps lack content hashes

4. *Conditional requests*: No specified validator discipline

14.3. XML Sitemaps

Traditional XML sitemaps provide: - URL discovery - <lastmod> timestamp

Key differences:

1. *Hashes*: XML sitemaps lack content fingerprints

2. *Timestamps*: lastmod insufficient for template changes

3. *Conditional requests*: No integration with HTTP caching

TCT adds content hashes enabling efficient change detection.

15. Implementation Status

[Note to RFC Editor: Remove this section before publication.]

Reference Implementations:

* WordPress Plugin: https://github.com/antunjurkovic-collab/trusted-collab-tunnel

* Python Client: https://github.com/antunjurkovic-collab/collab-tunnel-python

* Cloudflare Worker: https://github.com/antunjurkovic-collab/trusted-collab-worker

* Protocol Specification: https://github.com/antunjurkovic-collab/collab-tunnel-spec

As of October 2025, TCT has been implemented in:

Publisher implementations: - WordPress plugin (970 URLs across 3 production sites, 100% compliance) - Cloudflare Worker (edge-based implementation)

Measurements: - HTML-only: 103 KB average (13,900 tokens) - TCT JSON: 17.7 KB average (1,960 tokens) - *Bandwidth savings: 83%* - *Token savings: 86%*

Sitemap-first performance: - Skip rate: 90%+ for unchanged content - 304 hit rate: 95%+ on changed content

16. Acknowledgments

Thanks to the WordPress, Cloudflare, and IETF HTTP Working Group communities for feedback on early drafts.

Test Vector 1: Basic ASCII with Whitespace - Input String: literal HelloWorld\n\t\nTesting (two spaces, newline, tab, newline) - Normalized String: hello world testing - SHA-256 (hex): 479045cd11cebe841bab15d5ffba3dbac4fed0ca5c4eb74d1102e562a45f4f1f -

```
contentHash: sha256-
479045cd11cebe841bab15d5ffba3dbac4fed0ca5c4eb74d1102e562a45f4f1f -
ETag: "sha256-
479045cd11cebe841bab15d5ffba3dbac4fed0ca5c4eb74d1102e562a45f4f1f"
```

17. References

17.1. Normative References

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

[RFC9111] Fielding, R., Nottingham, M., and J. Reschke, "HTTP Caching", June 2022, <https://www.rfc-editor.org/rfc/rfc9111>.

[RFC8288] Nottingham, M., "Web Linking", October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

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

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

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", July 2002, <https://www.rfc-editor.org/rfc/rfc3339>.

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

[UAX15] Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", 2024, <https://www.unicode.org/reports/tr15/>.

[Unicode-CaseFolding] Unicode Consortium, "Unicode Case Folding", 2024, <https://www.unicode.org/versions/Unicode15.1.0/ch03.pdf>.

17.2. Informative References

[RFC9530] Pardue, L. and R. Polli, "Digest Fields", February 2024, <https://www.rfc-editor.org/rfc/rfc9530>.

[RFC9421] Backman, A., Richer, J., and M. Sporny, "HTTP Message Signatures", February 2024, <https://www.rfc-editor.org/rfc/rfc9421>.

[RFC9448] Polli, R. and A. Martinez, "RateLimit Fields for HTTP", October 2023, <https://www.rfc-editor.org/rfc/rfc9448>.

[RFC6596] Ohye, M. and J. Kupke, "The Canonical Link Relation", April 2012, <https://www.rfc-editor.org/rfc/rfc6596>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[ResourceSync] Open Archives Initiative, "ResourceSync Framework Specification", May 2017, <https://www.openarchives.org/rs/1.1/resourcesync>.

[AMP] "Accelerated Mobile Pages (AMP) HTML Specification", n.d., <https://amp.dev/documentation/guides-and-tutorials/learn/spec/amphtml/>.

[WHATWG-HTML] WHATWG, "HTML Living Standard - Named character references", n.d., <https://html.spec.whatwg.org/multipage/named-characters.html>.

[XMLSitemaps] sitemaps.org, "Sitemap XML format", n.d., <https://www.sitemaps.org/protocol.html>.

Appendix A. Example Implementation (WordPress)

This section is informative.

PHP implementation (simplified, using Method A - Canonical JSON Strong-Byte):

```
function handle_llm_endpoint() {
  // Get the current post
  $post_id = get_the_ID();

  // Build content string (plain text, no HTML)
  $title = get_the_title($post_id);
  $body = apply_filters(
    'the_content',
    get_post_field('post_content', $post_id)
  );
  $body_text = wp_strip_all_tags($body);

  // Combine with deterministic separator
  $content = $title . "\n\n" . $body_text;

  // Generate ETag from content string
  $normalized = normalize_text($content);
  $hash_hex = hash('sha256', $normalized);
  $hash = "sha256-{$hash_hex}";
  $etag = "\"{$hash}\"";  // Strong ETag (no W/ prefix)

  $canonical_url = get_permalink($post_id);

  // Check If-None-Match
  if (
    isset($_SERVER['HTTP_IF_NONE_MATCH'])
    && trim($_SERVER['HTTP_IF_NONE_MATCH']) === $etag
  ) {
    header('HTTP/1.1 304 Not Modified');
    header("ETag: {$etag}");
    header("Link: <{$canonical_url}>; rel=\"canonical\"");
    exit;
  }

  // Return JSON
  header('Content-Type: application/json');
  header("ETag: {$etag}");
  header("Link: <{$canonical_url}>; rel=\"canonical\"");
  header(
    'Cache-Control: max-age=0, must-revalidate, '
    . 'stale-while-revalidate=60, stale-if-error=86400'
```

```
    );
    header('Vary: Accept-Encoding');

    echo json_encode([
      'profile' => 'tct-1',
      'canonical_url' => $canonical_url,
      'title' => $title,
      'content' => $content,
      'hash' => $hash
    ]);
  }
  // Minimal normalization helper (for Method B implementations)
  function normalize_text($text) {
    $text = html_entity_decode($text, ENT_QUOTES | ENT_HTML5, 'UTF-8');
    if (class_exists('Normalizer')) {
      $text = Normalizer::normalize($text, Normalizer::NFKC);
    }
    if (function_exists('mb_convert_case')) {
      $text = mb_convert_case($text, MB_CASE_FOLD, 'UTF-8');
    } else {
      $text = strtolower($text);
    }
    $text = preg_replace(
      '/[\x00-\x08\x0B-\x0C\x0E-\x1F\x7F-\x9F]/u',
      '',
      $text
    );
    $text = preg_replace(
      '/[ \t\n\r\f]+/',
      ' ',
      $text
    );
    return trim($text);
  }
```

Appendix B. Example Sitemap

This section is informative.

```
{
  "version": 1,
  "items": [
    {
      "cUrl": "https://example.com/article-1/",
      "mUrl": "https://example.com/article-1/llm/",
      "modified": "2025-10-01T12:00:00Z",
      "contentHash": "sha256-abc123..."
    },
    {
      "cUrl": "https://example.com/article-2/",
      "mUrl": "https://example.com/article-2/llm/",
      "modified": "2025-09-15T08:30:00Z",
      "contentHash": "sha256-def456..."
    }
  ]
}
```

Normalization Test Vectors (Informative)

This appendix provides test vectors to assist implementers in validating normalization logic. The final hash MUST be an exact byte-for-byte match of the normalized string encoded as UTF-8.

Author’s Address

Antun Jurkovikj
North Macedonia
Email: antunjurkovic@gmail.com