Independent Submission
Internet-Draft
Intended status: ExperimentalCentre for Development of Advanced Computing (CDAC)
Expires: 22 July 2026

A. Mallick
I. Chebolu

18 January 2026

The Micro Agent Communication Protocol (袖ACP)
draft-mallick-muacp-02

Abstract

This document specifies the Micro Agent Communication Protocol (袖ACP), a resource-efficient messaging protocol for autonomous agents operating on resource-constrained Edge and IoT devices (including Class 1 and Class 2 devices per [RFC7228]). Existing agent communication protocols assume unbounded computational and energy resources; 袖ACP provides bounded resource consumption guarantees with deterministic memory bounds (8-byte header, up to 1024-byte TLV region, up to 65535-byte payload) and bounded processing time per message, while maintaining expressiveness sufficient for finite-state coordination patterns. The protocol defines four core message types, a fixed 64-bit header, TLV-based extensibility, and mandatory OSCORE security binding for operation in adversarial environments.

Status of This Memo

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

Copyright Notice

Table of Contents

1. Introduction

The Micro Agent Communication Protocol (ツオACP) is a compact, resource-efficient communication protocol designed for distributed autonomous agents operating on resource-constrained Edge and IoT devices (including Class 1 and Class 2 devices per [RFC7228]). It aims to bridge the gap between resource-light IoT protocols and semantically rich agent communication languages, by offering minimal overhead yet expressive interaction semantics.

Modern IoT, edge, and embedded environments involve devices with limited RAM, CPU, energy, and unreliable networks. Distributed applications require coordination, state sharing, event subscriptions, and request/response semantics. Traditional agent-communication languages impose heavy overhead unacceptable on microcontroller-class platforms, while standard IoT protocols provide minimal semantics.

ツオACP addresses this by defining a wire-efficient, fixed-header, TLV-extensible protocol with four core verbs (PING, TELL, ASK, OBSERVE) sufficient for request/response, publish/subscribe, and liveness checking. The protocol enables lean, deterministic implementations suitable for microcontroller-class devices while supporting structured multi-agent interactions. Formal foundations including resource bounds and safety verification are established in [MUACP].

This specification mandates CoAP with OSCORE [RFC8613] as the mandatory-to-implement transport binding, ensuring end-to-end confidentiality, integrity, and replay protection for constrained devices.

1.1. Goals

ツオACP aims to: provide minimal, low-overhead communication for constrained agents with structured semantics; ensure deterministic and bounded resource usage; support essential multi-agent patterns (request/response, publish/subscribe, liveness) using four orthogonal primitives; define a secure, interoperable transport binding; enable extensibility via TLV options without breaking compatibility.

1.2. Scope

This specification defines the wire format, core semantics, normative behavior, mandatory transport binding, security constraints, and IANA registries. It does not specify application-level semantics (content encoding, agent ontology, high-level negotiation), which are left to deployment-specific or higher-layer protocols.

1.3. Document Structure

Sections 2-3 define conventions, terminology, and message encoding. Sections 4-5 define protocol semantics and the mandatory CoAP/OSCORE transport binding. Sections 6-7 cover error handling, version negotiation, and IANA registries. Sections 8-9 define state machines and security considerations. Sections 10-11 cover interoperability profiles, wire examples, and conformance tests.

2. Conventions and 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 they appear in ALL CAPS. These words may also appear in lowercase or mixed case as plain English words, absent their normative meanings.

2.1. Terminology

Key terms: *Agent*寰蚤utonomous software entity participating in ツオACP communication; *Verb*寰俳ne of four primitives (PING, TELL, ASK, OBSERVE) encoded in 2 bits; *TLV*寰乃ype-Length-Value encoding (8-bit Type, 8-bit Length, variable Value); *Correlation ID*寰6-bit identifier grouping messages into a conversation; *Sequence ID*—16-bit monotonically increasing identifier for duplicate detection; *Conversation*—sequence of related messages identified by Correlation ID; *OSCORE*—Object Security for Constrained RESTful Environments [RFC8613]; *CoAP*—Constrained Application Protocol [RFC7252]; *Constrained Device*—device with limited resources per [RFC7228] (Class 1: ~10 KB RAM/100 KB flash; Class 2: ~50 KB RAM/250

KB flash).

2.2. Notation

Notation: hexadecimal values prefixed with "0x"; binary values prefixed with "0b"; network byte order (big-endian) unless otherwise specified; bit positions numbered from 0 (MSB) to n-1 (LSB); message formats shown using ASCII art diagrams.

2.3. Abbreviations

Abbreviations: CBOR (Concise Binary Object Representation [RFC8949]), CID (Correlation ID), CoAP (Constrained Application Protocol), COSE (CBOR Object Signing and Encryption [RFC8152]), EDHOC (Ephemeral Diffie-Hellman Over COSE [RFC9528]), FSM (Finite State Machine), IANA (Internet Assigned Numbers Authority), IoT (Internet of Things), MTI (Mandatory to Implement), OSCORE (Object Security for Constrained RESTful Environments), QoS (Quality of Service), SID (Sequence ID), TLV (Type-Length-Value), URI (Uniform Resource Identifier).

3. Message Model and Encoding Rules

This section defines the normative wire-level encoding of ACP messages, including the fixed header, TLV format, payload processing rules, byte ordering, and OSCORE protection boundaries. All compliant implementations MUST follow these encoding rules exactly unless otherwise specified.

3.1. Message Structure

A ACP message consists of three components encoded in the following order:

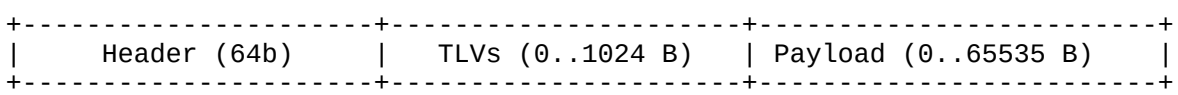


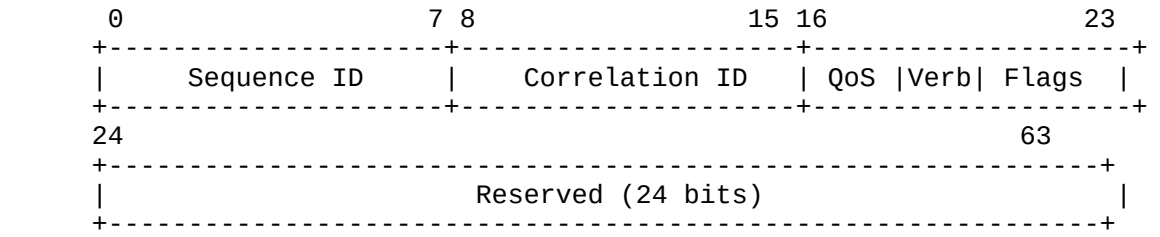
Figure 1: Figure 1: ACP Message Layout

The header format is fixed-length and MUST always appear. TLVs and payloads are optional. Messages MUST NOT exceed transport-imposed size limits; for CoAP/OSCORE, these limits are determined by underlying MTU constraints and CoAP Blockwise Transfer [RFC7959] if used.

All fields are encoded in network byte order (big-endian).

3.2. Header Format

The ACP header consists of 64 bits arranged as follows:



Byte 0-1: Sequence ID (16 bits, big-endian)
Byte 2-3: Correlation ID (16 bits, big-endian)
Byte 4: Bits 0-1: QoS (2 bits)
Bits 2-3: Verb (2 bits)
Bits 4-7: Flags (4 bits)
Byte 5-7: Reserved (24 bits, all zeros)

Figure 2: Figure 2: ACP Header Bit Layout

Sequence ID (16 bits, bytes 0-1): Monotonically increasing identifier used for duplicate detection and replay-window tracking. Sequence ID is per-sender (per OSCORE security context) and monotonically increases within each sender’s message stream. MUST wrap modulo 2^16. Sequence ID SHOULD be initialized to a random value (not 0) to prevent predictability and traffic analysis. Initialization to 0 is acceptable only when: (1) establishing a new OSCORE security context, (2) no prior communication history exists with the peer, and (3) the initialization is synchronized with the establishment of the new OSCORE context. In all other cases, Sequence IDs MUST be initialized to a random value. Sequence IDs MUST be unpredictable if security-sensitive traffic requires preventing traffic analysis.

Correlation ID (16 bits, bytes 2-3): Identifies all messages belonging to the same conversation. Correlation ID MUST be unique among active conversations from the same sender (same OSCORE security context). Different senders may independently use the same Correlation ID values, as conversations are scoped per OSCORE security context. SHOULD be randomly generated in security-sensitive deployments.

QoS (2 bits, byte 4 bits 0-1): Encodes transmission semantics (fire-and-forget, at-least-once, at-most-once). Values are defined in the IANA Considerations section.

Verb (2 bits, byte 4 bits 2-3): Identifies one of the four ACP operations: PING(0), TELL(1), ASK(2), OBSERVE(3).

Flags (4 bits, byte 4 bits 4-7): Control bits reserved for protocol-level features such as fragmentation, retransmission hints, or message cancellation. Future specifications MAY define additional meanings.

Reserved (24 bits, bytes 5-7): MUST be set to zero on transmission and ignored by receivers. Reserved bits MAY be repurposed by future ACP versions but MUST NOT change meaning in this version.

3.3. TLV Encoding

TLVs (TypeLengthValue structures) convey optional metadata and extensibility information. They appear immediately after the header and MUST appear in Type-increasing order to allow binary search and deterministic parsing.

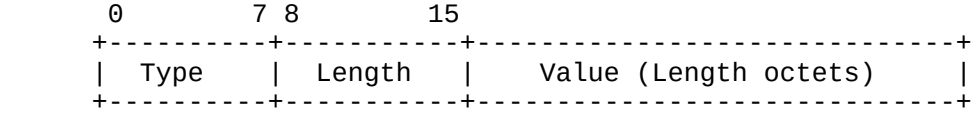


Figure 3: Figure 3: TLV Encoding

Type (8 bits): TLV identifier. The meaning of each Type is defined in the IANA registry. Types 031 are reserved and governed by Standards Action. Types 32127 require IETF Review. Types 128255 are vendor-specific.

Length (8 bits): Specifies the number of octets in the Value field (0-255). The Length field MUST NOT exceed 255. Receivers MUST validate that the declared Length does not exceed the remaining message buffer before reading the Value field. Each individual TLV's Value field MUST NOT exceed 255 octets. The total TLV region (sum of all TLV lengths plus their Type and Length fields) MUST NOT exceed 1024 bytes. Implementations MUST validate both constraints: individual TLV length 255 and total TLV region 1024 bytes.

Value: Encoded according to Type. For Types other than 0x00 (Raw Octets), the Value is subject to OSCORE protection (Section 5).

Critical TLVs: A future TLV Type range MAY designate critical TLVs. Receiving an unknown critical TLV MUST cause message rejection.

3.3.1. TLV Processing Rules

Receivers MUST apply the following rules when processing TLVs:

* TLVs MUST be parsed strictly in order.
* If Length exceeds remaining buffer size, the message MUST be discarded.
* Unknown TLV Types MUST be ignored unless they are designated critical.
* TLV order MUST be strictly increasing by Type; violating this is a format error.
* TLV Type 0x00 (Raw Octets) MUST NOT appear in encrypted messages; its use is restricted to unencrypted PING messages.

3.4. Payload Encoding

The ACP payload is an optional octet string of 065535 bytes used for application data, action parameters, event notifications, or encoded content (CBOR, JSON). Payloads MUST be OSCORE-protected unless the message Verb is PING. Payload sizes MUST be validated before allocation. If encoded using CBOR (Type=0x03), receivers MUST treat it as a single CBOR data item. If JSON (Type=0x02), it MUST be UTF-8 encoded.

3.5. Byte Ordering

All multi-octet integer fields in ACP (Sequence ID, Correlation ID, header composites) MUST be encoded in network byte order (big-endian). TLV and payload content MAY use other encoding rules (e.g., CBOR or UTF-8) as determined by their Types.

3.6. Fragmentation (Optional Feature)

ACP does not mandate fragmentation. TLV Type 0x10 is reserved for future fragmentation specification but MUST NOT be used until fully specified. Deployments using CoAP Blockwise Transfer [RFC7959] SHOULD avoid ACP-level fragmentation.

3.7. OSCORE Protection Boundaries

When ACP is transported over CoAP with OSCORE, the OSCORE-protected CoAP payload MUST contain the complete ACP message (Header | TLVs | Payload). OSCORE MUST protect: all TLVs except those in unencrypted PING messages, the entire payload, and header fields other than those needed for outer CoAP routing. Implementations MUST NOT leak semantics (e.g., Verb, QoS) through the CoAP outer header beyond what OSCORE permits.

3.8. Canonical Encoding Rules

Canonical encoding rules: fields MUST NOT be padded; TLVs MUST be sorted by ascending Type; no two TLVs MAY share the same Type unless explicitly defined; payload MUST begin immediately after the last TLV; implementations MUST normalize line endings, whitespace, or internal representations before hashing or signing application content.

4. Protocol Semantics

This section defines the normative semantics of the four ACP verbs: PING, TELL, ASK, and OBSERVE. Each verb represents a fundamental communication primitive intended to support higher-level agent behaviors, including liveness detection, request/response interactions, state dissemination, and event-driven notification.

Agents MUST implement all four verbs. Agents MUST apply OSCORE protection to all messages except PING, unless an application explicitly operates in an unauthenticated environment.

For each verb, this section defines sender behavior, receiver behavior, state-machine interactions, mandatory error cases, and expected processing-time bounds.

4.1. PING

PING provides low-cost reachability and liveness detection. PING messages SHOULD be sent unencrypted by default to remain lightweight. Implementations MAY support OSCORE-protected PING for authenticated deployments, but unencrypted PING MUST be accepted by all receivers. Sender: MAY emit PING at any time; MUST increment Sequence ID; SHOULD use unique Correlation ID; MUST rate-limit (1 per 10 seconds per peer). Receiver: SHOULD reply with TELL; MUST NOT require OSCORE for unencrypted PING; MUST accept both unencrypted and OSCORE-protected PING; SHOULD rate-limit PING processing. Unencrypted PING may leak topology/presence information; implementations SHOULD use rate limiting and randomization. For authenticated liveness, use ASK/TELL

with OSCORE or OSCORE-protected PING.

4.2. TELL

TELL conveys information, updates, or asynchronous notifications, and responds to ASK messages. TELL messages MUST be OSCORE-protected unless deployment explicitly allows unauthenticated mode. Sender: MUST include payload or meaningful TLV set; MUST increment Sequence ID; when responding to ASK, MUST use same Correlation ID. Receiver: MUST validate OSCORE; MUST associate via Correlation ID; MUST incorporate content per application policy. Errors: TELL without OSCORE MUST be rejected (unless non-secure mode); malformed TLVs MUST cause discard.

4.3. ASK

ASK initiates a request for information or action and typically elicits a TELL response. ASK messages MUST be OSCORE-protected. Sender: MUST allocate new conversation entry indexed by Correlation ID; MUST increment Sequence ID; MUST start request timer (default timeout of 30 seconds is RECOMMENDED for constrained devices, with exponential backoff for retransmissions when QoS=1); MUST enforce conversation limits. Receiver: MUST validate OSCORE; MUST associate ASK with Correlation ID; MUST generate TELL response with result or error TLV. Errors: malformed TLVs → TELL(error); security validation failure → silent discard; correlation-table limits exceeded → resource exhaustion error.

4.4. OBSERVE

OBSERVE establishes a subscription for future notifications, scoped to a single peer. OBSERVE messages MUST be OSCORE-protected. Sender: MUST allocate/update subscription state indexed by Correlation ID; MUST validate subscription limits; MUST increment Sequence ID; MAY include subscription parameter TLVs (topic, conditions); MUST send periodic TELL notifications while active. Receiver: MUST validate OSCORE; MUST establish/refresh subscription state; MUST enforce expiration and resource ceilings; when conditions met, MUST send event notifications as TELL. Cancellation: TELL or OBSERVE with Cancel-Subscription TLV (Type=0xFF) → receiver MUST delete state, stop notifications. Errors: subscription limits exceeded → TELL(error); OSCORE validation failure → drop.

4.5. Summary of Normative Requirements

Summary: PING— liveness probe, SHOULD NOT require OSCORE (unencrypted by default), implementations MAY support OSCORE-protected PING; TELL— update/response/notification, MUST use OSCORE except in explicitly insecure deployments; ASK— request, MUST use OSCORE, MUST generate TELL response; OBSERVE— subscription, MUST use OSCORE, MUST create or update subscription state. Agents MUST NOT overload verbs with incompatible semantics.

5. Mandatory Transport Binding: OSCORE/CoAP

This section defines the mandatory-to-implement (MTI) transport binding for ACP: the combination of the Constrained Application Protocol (CoAP) as the transport substrate and OSCORE as the end-to-end object security mechanism. All compliant ACP implementations MUST support this binding.

Deployments MAY support additional bindings (e.g., DTLS/UDP as specified in [RFC9147] or QUIC) but such bindings are outside the scope of this specification and MUST NOT weaken or replace the OSCORE/CoAP MTI profile.

5.1. Mapping ACP Messages to CoAP

Each ACP message (Header | TLVs | Payload) is encoded as a byte string and placed entirely within the CoAP message payload. Only OSCORE-protected CoAP messages may carry ACP messages (except PING, which MAY be unprotected). ACP messages MUST use: Method=POST, URI-Path="muacp" (fixed), Content-Format=application/muacp+binary, Payload=Full ACP message. Each ACP message corresponds to exactly one CoAP POST.

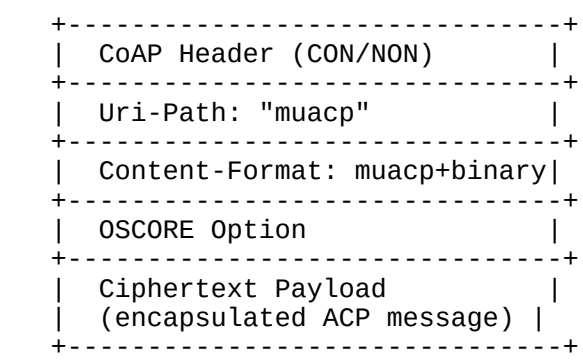


Figure 4: Figure 4: CoAP Envelope Carrying a ACP Message

5.2. OSCORE Protection Requirements

All ACP messages except unencrypted PING MUST be protected using OSCORE [RFC8613], which uses COSE [RFC8152] for cryptographic operations. OSCORE MUST protect: the entire ACP header (except outer CoAP routing metadata), all TLVs except raw TLVs permitted for PING, and the entire ACP payload. OSCORE replay protection MUST be enabled with replay windows configured to match expected message rate and resource constraints. OSCORE MUST use a unique security context per agent-pair.

5.3. Establishing OSCORE Security Contexts

Security contexts for OSCORE MAY be derived by: EDHOC (RECOMMENDED), Pre-Shared Keys (PSK), or out-of-band provisioning. When EDHOC is used, the resulting OSCORE context MUST be bound to the EDHOC handshake transcript to prevent identity misbinding attacks. If a device exhausts its available context storage, it MUST reject new context establishment requests with ERR_RESOURCE_EXHAUSTED. If all OSCORE contexts are active and a new context establishment request arrives, implementations MUST reject it with ERR_RESOURCE_EXHAUSTED. Implementations SHOULD implement context eviction policies (e.g., least-recently-used) for inactive contexts but MUST NOT terminate active conversations. For Class 1 devices, implementations SHOULD limit concurrent OSCORE contexts (e.g., 8-16 contexts).

5.4. CoAP Message Types and Reliability

ACP builds upon CoAP reliability semantics to achieve its QoS model. Implementations MUST map ACP QoS codes to CoAP message types as follows:

ACP QoS	Meaning	CoAP Message Type
0	fire-and-forget	NON (Non-confirmable)
1	at-least-once delivery	CON (Confirmable)
2	at-most-once delivery	NON (Non-confirmable, no retransmission)

Table 1

QoS Semantics: QoS=0 (fire-and-forget) provides best-effort delivery. QoS=1 (at-least-once) ensures delivery through CoAP retransmissions. QoS=2 (at-most-once) provides a single delivery

attempt without retransmission, suitable for idempotent operations. Both QoS=0 and QoS=2 use CoAP NON messages. Implementations MUST NOT retransmit QoS=2 messages at the ACP layer.

CoAP-level acknowledgments MUST NOT be interpreted as ACP-level responses. Application responses are always encoded as TELL messages.

5.5. Mapping ASKTELL to CoAP Request/Response

ASK messages MUST be sent as CoAP POST requests; TELL responses as CoAP responses. OSCORE MUST protect both directions. The Correlation ID uniquely links ASK with TELL response. CoAP Message IDs MUST NOT be used for application correlation. Receivers MUST respond with TELL even when requests fail, using an Error TLV.

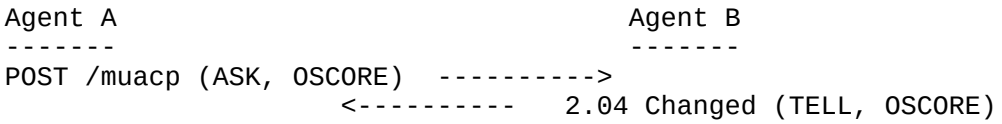


Figure 5: Figure 5: ASK/TELL Over OSCORE-CoAP

5.6. Mapping OBSERVE Subscriptions

OBSERVE establishes a long-lived subscription. ACP defines its own subscription model, independent of CoAP's Observe extension [RFC7641]. OBSERVE MUST be mapped as: CoAP POST containing ACP OBSERVE message; notifications delivered as CoAP POSTs containing TELL messages. Implementations MUST NOT use CoAP Observe for ACP subscriptions.

5.7. Congestion Control Requirements

All ACP-over-CoAP deployments MUST implement congestion control to prevent network collapse and unfair bandwidth usage. Implementations MUST follow CoAP congestion control mechanisms as specified in [RFC7252] Section 4.7.

Agents MUST adhere to: exponential backoff on CoAP CON retransmissions (initial timeout 2s, max 247s per [RFC7252]); PING rate limiting (1 per 10 seconds per peer); OBSERVE throttling when bandwidth pressure is detected; deterministic resource usage; message rate limits per conversation. When Blockwise Transfer [RFC7959] is used, agents MUST ensure block sizes do not exceed memory limits.

5.8. Transport-Layer Error Handling

Transport errors (CoAP timeouts, OSCORE decryption failures, missing acknowledgments) MUST be translated into ACP-level behavior. OSCORE decryption failures → drop message. Unacknowledged CoAP CON → apply ACP QoS semantics for retransmission. Repeated timeouts → conversation enters failure state. Malformed CoAP envelopes → discard.

5.9. Summary of MTI Requirements

All compliant ACP implementations MUST: support CoAP POST to fixed path "muacp"; support Content-Format application/muacp+binary; protect all messages except unencrypted PING with OSCORE; enforce OSCORE replay protection; derive OSCORE contexts using EDHOC or equivalent; map QoS codes to CoAP message types; generate TELL responses for all ASK messages; deliver OBSERVE notifications as TELL messages. This binding ensures interoperability and establishes a minimum security baseline.

6. Error Handling, Version Negotiation, and Extensibility

This section defines normative error-handling rules, version-negotiation mechanism, downgrade protection requirements, and the extensibility framework provided by the TLV architecture.

6.1. Error Code TLVs

All protocol-level errors MUST be communicated using a TELL message that includes an Error-Code TLV. Error codes are encoded as unsigned integers and MUST follow the registry defined in the IANA Considerations section.

```
Type:   0x22 (Error-Code, see IANA registry)
Length: 1 or 2 octets
Value:  Integer error code
```

Figure 6: Figure 6: Error-Code TLV

The sender MUST set the Correlation ID of the error response to match the ID of the failing message. Receivers MUST interpret the error code as part of the ACP conversation state.

6.2. Standardized Error Conditions

The following error codes are defined for ACP:

Code	Name	Description
0x00	SUCCESS	No error; operation completed successfully. This code is OPTIONAL. If omitted, successful completion is indicated by the absence of an Error-Code TLV. Receivers MUST treat the absence of an Error-Code TLV as equivalent to SUCCESS (0x00).
0x01	ERR_MALFORMED	Malformed header, TLV, or payload.
0x02	ERR_UNSUPPORTED_VERB	Verb not recognized or not supported by receiver.
0x03	ERR_UNSUPPORTED_TLV	Critical TLV not understood.
0x04	ERR_FORBIDDEN	Operation not permitted due to policy or authorization.
0x05	ERR_RESOURCE_EXHAUSTED	Memory, CPU, or subscription/ conversation limits exceeded.
0x06	ERR_VERSION_MISMATCH	Message uses unsupported protocol version.
0x07	ERR_TIMEOUT	Sender or receiver timed out while waiting for a response.
0x08	ERR_INTERNAL	Internal failure not covered by other error categories.

Table 2

Implementations MAY define additional vendor-specific error codes in the vendor range but MUST NOT redefine standardized codes.

6.3. Handling Malformed Messages

Receivers MUST apply strict validation: if TLV Length exceeds remaining bytes, discard; if TLVs appear out of Type order, discard; if required TLV (future versions) is absent, reject; if header fields contain invalid combinations (e.g., reserved bits set), reject; if OSCORE decryption fails, discard without error signaling. Where feasible, receivers SHOULD send TELL(error) unless doing so would amplify a denial-of-service attack.

6.4. Conversation-Lifetime Error Handling

Conversations MAY fail due to timeouts, resource limits, or message corruption. When such failures occur:

* The agent MUST free associated resources (conversation-table entries).

* The agent SHOULD send an ERR_TIMEOUT or ERR_RESOURCE_EXHAUSTED TELL message.

* For resource exhaustion, an agent MUST NOT attempt recovery that risks violating its resource budget.

If a Correlation ID collision is detected (a new message arrives with a Correlation ID matching an active conversation from the same sender, as identified by the OSCORE security context), the receiver MUST apply the following deterministic strategy in order:

1. If the conversation table is full (all entries occupied), reject the new message with ERR_RESOURCE_EXHAUSTED and maintain the existing conversation.

2. If the new message’s Sequence ID (from the same sender) is greater than the existing conversation’s last observed Sequence ID from that sender, terminate the existing conversation (free its resources), accept the new message, and create a new conversation entry. This handles legitimate Correlation ID reuse after conversation completion or timeout. Note: Sequence IDs are per-sender and monotonically increase within each sender’s message stream. To handle Sequence ID wrap-around (modulo 2^16), implementations MUST use sequence number comparison as defined in [RFC1982] Section 3.1: given two Sequence IDs S1 and S2, S1 is considered greater than S2 if (S1 > S2 and S1 - S2 < 2^15) or (S1 < S2 and S2 - S1 > 2^15). This ensures correct ordering even when Sequence IDs wrap from 0xFFFF to 0x0000.

3. If the new message's Sequence ID is less than or equal to the existing conversation's last observed Sequence ID from the same sender, reject the new message as a potential replay or out-of-order delivery. The receiver MUST NOT modify the existing conversation state and SHOULD silently discard the new message (or MAY send ERR_MALFORMED if the message appears valid but out-of-order).

This deterministic strategy ensures interoperability while preventing resource exhaustion and replay attacks. Correlation ID collisions are rare when Correlation IDs are randomly generated with sufficient entropy. Collisions from different senders (different OSCORE contexts) are handled separately, as each OSCORE context maintains its own conversation state. The Sequence ID comparison is secure because Sequence IDs are authenticated and integrity-protected by OSCORE.

Example: If an active conversation exists with Correlation ID=0x1234 and last observed Sequence ID=0x0010 from sender A (identified by OSCORE context A), and a new message arrives with Correlation ID=0x1234 and Sequence ID=0x0015 from the same sender A, the receiver terminates the old conversation and accepts the new one. If the new message has Sequence ID=0x0005 from sender A, it is rejected as a potential replay. If a message arrives with Correlation ID=0x1234 from sender B (different OSCORE context), it is treated as a separate conversation, as conversations are scoped per OSCORE security context.

6.5. Version Negotiation

ACP includes a Version-TLV (Type=0x01) that MAY be included in any message to indicate supported protocol versions. If no Version-TLV is present, receivers MUST assume version 0x00 (this specification).

Version negotiation follows these rules:

* If a message includes a Version-TLV that indicates only unsupported versions (i.e., all versions listed in the Version-TLV are higher than the receiver's maximum supported version), the receiver MUST return ERR_VERSION_MISMATCH in a TELL error response.

* If the Version-TLV contains at least one supported version, the receiver MUST use the highest mutually supported version for subsequent messages in the conversation.

* When both parties send Version-TLVs (e.g., in ASK and TELL), each party MUST independently select the highest mutually supported version from the union of both Version-TLV lists. If no common version exists, the receiver MUST return ERR_VERSION_MISMATCH.

* The selected version applies to all messages in the conversation identified by the Correlation ID. Once a version is selected, it MUST NOT be changed for that conversation.

* Version negotiation MUST occur under OSCORE protection. PING messages (which are typically unencrypted) SHOULD NOT carry Version-TLV to maintain lightweight operation. If version negotiation is required before OSCORE context establishment, implementations SHOULD use ASK/TELL with OSCORE or establish the OSCORE context first, then negotiate versions in subsequent messages.

```
Type:   0x01 (Version)
Length: N (number of supported versions, 1-255)
Value:  Sequence of N unsigned 8-bit integers, each representing
        a supported protocol version number (e.g., [0x00] for version 0)
```

Figure 7: Figure 7: Version TLV

Encoding: The Value field of the Version TLV is a sequence of N unsigned 8-bit integers (where N is the Length field value). Each integer represents a protocol version number. For example, a Version TLV indicating support for versions 0x00 and 0x01 would have Length=2 and Value=[0x00, 0x01]. Implementations MUST encode version numbers as single octets (0-255). Receivers MUST parse the Value field as a sequence of Length octets, each interpreted as an unsigned 8-bit version number.

6.6. Downgrade and Version-Rollback Protection

Implementations MUST ensure attackers cannot force a peer to use a lower protocol version when a higher mutually supported version is available. The highest mutually supported version MUST be chosen. Version negotiation MUST occur inside OSCORE-protected messages (except PING). Agents MUST NOT downgrade versions unless a failure condition explicitly requires fallback.

6.7. Extensibility Framework

ACP evolves through TLV-based extensibility. Constraints: receivers MUST ignore unknown non-critical TLVs; implementations MUST NOT reuse TLV Types for different semantics; future versions MAY introduce critical TLVs (unsupported critical TLVs trigger ERR_UNSUPPORTED_TLV); all TLVs MUST be sorted by increasing Type value; Types 128255 are vendor-specific and require no global registration. Complex extensions SHOULD define new structured TLVs rather than overloading primitive types.

6.8. Summary of Normative Requirements

Malformed messages MUST be rejected and SHOULD trigger TELL(error) unless unsafe. Errors MUST use standardized codes. Version negotiation MUST prefer the highest mutually supported version. Unknown non-critical TLVs MUST be ignored; unknown critical TLVs MUST trigger errors. OSCORE failures MUST cause silent discard. Resource exhaustion MUST lead to conservative cleanup behavior.

7. IANA Considerations

This section requests the creation of new registries and assignments required for ACP to function as an interoperable Internet protocol. All registries use the policies defined in [RFC8126]. Unless otherwise stated, values are allocated using the "IETF Review" policy.

7.1. ACP TLV Types Registry

IANA is requested to create a new registry entitled "ACP TLV Types" (8-bit values 0255). Each entry MUST contain: Value, Name, Description, Value format, Reference. The range is divided as follows:

* *031:* Standards Action (as defined in [RFC8126] Section 4.1)

* *32127:* IETF Review

* *128255:* Vendor-specific (First Come First Served)

Initial values:

Value	Name	Description	Format	Reference
0x00	RAW_OCTETS	Unstructured data; MUST NOT appear in encrypted messages except PING.	Opaque	This document
0x01	VERSION	Advertised supported protocol versions.	Array of integers	This document
0x02	CONTENT_TYPE	Specifies payload encoding.	Integer	This document
0x03	CBOR_PAYLOAD	Payload encoded as CBOR.	CBOR data item	This document
0x10	RESERVED	Reserved for future fragmentation specification. Implementations MUST NOT use Type=0x10 until fragmentation is fully specified in a future extension document.	N/A	This document
0x20	TOPIC	Subscription topic for OBSERVE.	UTF-8 string	This document
0x21	CONDITION	Trigger condition for OBSERVE.	UTF-8 or CBOR	This document
0x22	ERROR_CODE	Error code returned in TELL(error).	Integer	This document

0xFF	CANCEL_SUBSCRIPTION	Explicit termination of OBSERVE subscription.	Empty	This document

Table 3

Future extensions MUST NOT assign new semantics to existing TLV values.

7.2. ACP QoS Codes Registry

IANA is requested to create a registry entitled "ACP QoS Codes". QoS is encoded as a 2-bit field in the header (values 03).

Value	Name	Description	Reference
0	FIRE_AND_FORGET	No reliability; mapped to CoAP NON.	This document
1	AT_LEAST_ONCE	Retransmissions required; mapped to CoAP CON.	This document
2	AT_MOST_ONCE	No retransmission; mapped to CoAP NON.	This document
3	RESERVED	Reserved for future use.	This document

Table 4

7.3. ACP Verb Codes Registry

IANA is requested to create a registry entitled "ACP Verb Codes". Verb values occupy 2 bits but are listed numerically (03).

Value	Name	Description	Reference
0	PING	Liveness probe.	This document
1	TELL	State update, notification, or response.	This document
2	ASK	Request for information or action.	This document
3	OBSERVE	Subscription to events or state changes.	This document

Table 5

7.4. ACP Error Codes Registry

IANA is requested to create a registry entitled "ACP Error Codes" consisting of integers 0255.

The initial contents are listed in the Error Handling section. The assignment policy for values 0127 is IETF Review. Values 128255 are vendor-specific and use the "First Come First Served" policy.

7.5. CoAP Content-Format Registration

IANA is requested to register the following CoAP Content-Format:

Name	Media Type	Encoding	ID	Reference
application/ muacp+binary	application/ muacp+binary	Binary	TBD (to be assigned by IANA)	This document

Table 6

This Content-Format is mandatory for all ACP-over-CoAP messages.

Note: The Content-Format ID value marked as "TBD" will be assigned by IANA during the IESG review process, prior to publication of this document as an RFC. The assignment will follow the "IETF Review" policy as specified in [RFC8126].

7.6. Media Type Registration

IANA is requested to register the following media type in the "application" registry per [RFC6838]:

Type name: application
Subtype name: muacp+binary
Required parameters: none
Optional parameters: none
Encoding considerations: binary
Security considerations: See Security Considerations section.
Interoperability considerations: Defined by TLV and header structure.
Published specification: This document.
Intended usage: COMMON
Author/Change controller: IETF

7.7. Well-Known CoAP Resource

IANA is requested to register the following CoAP Well-Known URI per [RFC8615] using the "IETF Review" policy:

URI	Description	Reference
/.well-known/muacp	Discovery resource indicating ACP support.	This document

Table 7

A CoAP GET to /.well-known/muacp SHOULD return a CBOR structure (Content-Format: application/cbor) describing supported TLVs, maximum sizes, and supported versions as specified in Section 10.5. A successful response (2.05 Content) MUST contain a CBOR map. A 4.04 Not Found response indicates that ACP is not supported by the device. Implementations MUST handle both success and error responses gracefully.

7.8. Summary of IANA Actions

IANA is requested to: create the ACP TLV Types registry and populate initial values; create the ACP QoS Codes registry; create the ACP Verb Codes registry; create the ACP Error Codes registry; register the CoAP Content-Format application/muacp+binary; register the media type application/muacp+binary; register the well-known CoAP resource /.well-known/muacp. These actions enable interoperable deployment and ensure long-term extensibility under IETF governance.

8. State Machines and Processing Logic

This section defines normative finite-state machines (FSMs) governing ACP conversations. Implementations MUST implement these FSMs for deterministic, interoperable behavior. Agents operate according to: receive message, validate OSCORE (if required), validate header/TLVs/ payload, identify conversation via Correlation ID, execute verb-specific FSM transition, emit resulting messages. Agents MUST support at least 64 concurrent conversations and MUST reject new conversations if resource ceilings are exceeded (ERR_RESOURCE_EXHAUSTED).

8.1. ASK/TELL Conversation State Machine

ASK initiates a conversation; TELL completes it. States: IDLE → (send ASK) → WAIT_RESP → (recv TELL or timeout) → COMPLETED → cleanup. If timer expires and QoS=1, retransmit ASK. If QoS=0/2, transition to COMPLETED with ERR_TIMEOUT. Receiver MUST emit TELL(error) for errors.

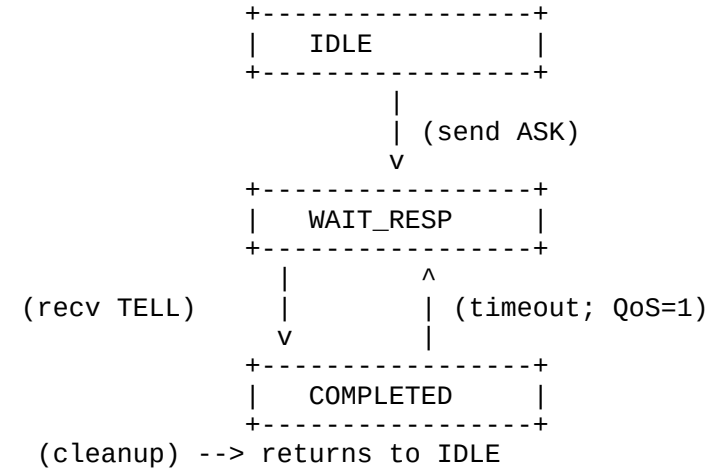


Figure 8: Figure 8: ASK/TELL State Machine

8.2. PING/TELL State Machine

PING serves as a minimal liveness check. PING is stateless and does NOT create persistent conversation table entries. States: IDLE → (send PING) → WAIT_PONG → (recv TELL or timeout) → COMPLETED. PING SHOULD NOT require OSCORE (unencrypted by default), implementations MAY support OSCORE-protected PING, MUST NOT modify application state, and MUST NOT cause retransmissions on timeout.

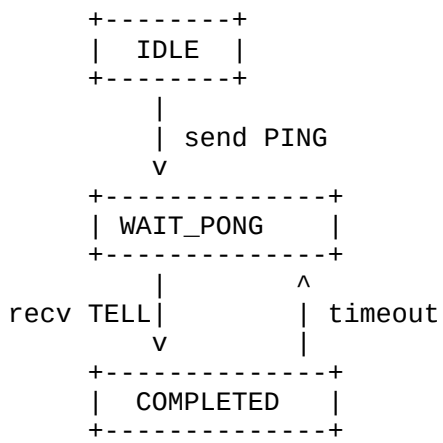


Figure 9: Figure 9: PING/TELL Liveness FSM

8.3. OBSERVE Subscription State Machine

OBSERVE establishes a long-lived subscription. States: NO_SUB → (recv OBSERVE) → SUBSCRIBED → (event trigger) → NOTIFY (TELL) → (recv CANCEL_SUB) → TERMINATED → cleanup. Subscriptions MUST expire after configured lifetime and MUST enforce resource ceilings (max subscriptions per peer).

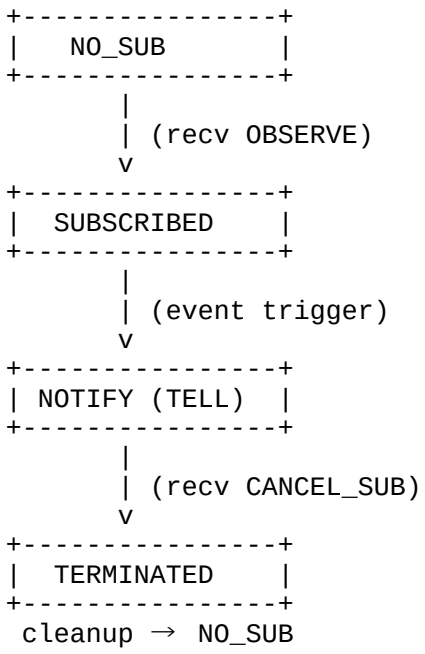


Figure 10: Figure 10: OBSERVE Subscription FSM

8.4. Error-State Transitions

Errors MUST transition FSMs to predictable termination states: ERR_MALFORMED → discard, no state; ERR_UNSUPPORTED_TLV → terminate conversation, send error TELL; ERR_TIMEOUT → COMPLETED with error, free resources; ERR_RESOURCE_EXHAUSTED → reject, no new state; OSCORE failure → silent discard, no state update.

8.5. Processing Time and Resource Bounds

All FSM transitions MUST complete in bounded time and memory. Required limits: conversation table (minimum 64 entries), subscription table (minimum 16 entries), deterministic message buffer sizes (header+1024-byte TLVs+payload), timers without per-message dynamic allocation. Platforms MAY use preallocated memory pools or static tables.

9. Security Considerations

This section defines the security properties, assumptions, and mandatory mitigations for ACP. The protocol relies on OSCORE and the underlying transport for security. All implementations MUST follow the requirements in this section to avoid exposure to denial-of-service, spoofing, downgrade, replay, or privacy attacks.

9.1. Threat Model

The ACP threat model assumes attackers may: passively eavesdrop; modify, inject, reorder, or replay messages; exhaust memory/CPU/storage/energy/subscription tables; desynchronize state; conduct traffic analysis; attempt version downgrades; exploit weak random number generation or incorrect OSCORE configuration. The protocol provides security *only* when implemented with OSCORE. Attackers are assumed to have full control of the transport layer but not of OSCORE-protected channels.

9.2. Authentication, Integrity, and Confidentiality

All ACP messages except unencrypted PING MUST be authenticated and integrity-protected using OSCORE. OSCORE provides peer authentication (when derived from EDHOC or provisioned credentials), integrity protection over header/TLVs/payload, replay protection, and request/response binding. Implementations MUST use a unique OSCORE security context per communicating peer. TELL, ASK, and OBSERVE messages MUST be encrypted via OSCORE. Authorization MUST be enforced before performing operations triggered by ASK or OBSERVE.

9.3. Replay Prevention and Freshness

ACP relies on OSCORE replay protection. Implementations MUST enable and correctly maintain OSCORE replay windows. Receivers SHOULD maintain a per-peer sliding window of recent Sequence IDs. Subscription-triggered notifications MUST validate freshness. Agents MUST reject delayed or reordered messages if OSCORE replay windows indicate a stale nonce.

9.4. Denial-of-Service and Resource Exhaustion

Implementations MUST enforce: maximum active conversations (minimum 64), maximum OBSERVE subscriptions (minimum 16), rate limits on PING and ASK, TLV region size limits (max 1024 bytes), payload size limits (max 65535 bytes), and static/preallocated memory pools. When limits are exceeded, agents MUST return ERR_RESOURCE_EXHAUSTED or silently drop messages. CoAP-level DoS mitigation (exponential backoff, NON vs CON) MUST also be applied.

9.5. Subscription Security

OBSERVE and CANCEL_SUB MUST be OSCORE-protected. Subscriptions MUST be bound to an authenticated OSCORE context. Correlation IDs MUST be unpredictable. Subscription deletion MUST require a valid CANCEL_SUB from the same authenticated peer or timeout/resource exhaustion. Agents MUST reject subscription attempts exceeding resource ceilings.

9.6. Downgrade Protection

The highest mutually supported version MUST be used. Version negotiation MUST occur under OSCORE (except PING). Agents MUST reject messages advertising only unsupported versions and MUST NOT fall back silently to lower versions.

9.7. Key Management

Implementations MUST provide: secure key provisioning (EDHOC, PSK, or manufacturing-time injection), rotation of OSCORE master secrets, secure deletion of expired keys, protection against key reuse across peers, and protection against side-channel extraction. Compromise of OSCORE keys compromises all ACP security properties.

Key Rotation: OSCORE master secrets SHOULD be rotated periodically (e.g., time-based: 30-90 days, usage-based: after 2^32 messages, or event-based: upon compromise suspicion). Rotation procedures MUST preserve active conversations where possible.

9.8. Side-Channel Attacks

Constrained devices may be vulnerable to side-channel attacks (timing, power, electromagnetic). Implementations SHOULD: use constant-time cryptographic operations, minimize observable timing differences, protect against power analysis (HSMs or software countermeasures), avoid leaking information through error timing or resource allocation, use secure random number generators for Correlation IDs and Sequence IDs. While complete side-channel resistance may be impractical on severely constrained devices, implementations SHOULD document their threat model and mitigations.

9.9. Safe Failure Modes

Malformed messages MUST be discarded without modifying state. OSCORE failures MUST be silent and MUST NOT produce error messages usable for oracle attacks. Timeouts MUST clean up state deterministically. Subscription state MUST never persist without authenticated refresh.

10. Interoperability and Deployment Profiles

This section defines the minimum feature set required for interoperability between ACP implementations, along with deployment profiles tailored to different classes of devices and networks.

10.1. Minimum Interoperability Profile (MIP)

All ACP implementations MUST support: the 64-bit header format; all four verbs (PING, TELL, ASK, OBSERVE); TLV processing with ordering and size limits; OSCORE/CoAP transport binding; Content-Format application/muacp+binary; conversation state for at least 64 active Correlation IDs; subscription state for at least 16 active OBSERVE subscriptions; error-handling and state-machine behavior as defined in this specification.

10.2. Constrained Node Profile (CNP)

CNP targets microcontroller-class devices (ARM Cortex-M, ESP32). Implementations MUST use static/preallocated buffers, enforce strict resource bounds, minimize logging, and restrict payload sizes. Implementations SHOULD prefer PSK/EDHOC-based OSCORE contexts and disable vendor-specific TLVs.

10.3. Infrastructure Node Profile (INP)

INP targets edge gateways and cloud-side collectors. Implementations MUST support full subscription features, extended TLV sets, high-throughput replay windows, EDHOC key exchange, and rate-shaping for constrained peers. INP nodes MAY provide protocol translation and hardware-accelerated crypto.

10.4. Cross-Profile Interoperability

INP nodes MUST respect CNP resource ceilings, MUST NOT exceed CNP size limits, and SHOULD apply traffic shaping. CNP nodes MUST ignore unsupported TLVs. Fallback to MIP MUST always be possible. All profile interactions MUST preserve security properties.

10.5. Feature Negotiation

Feature discovery uses GET /.well-known/muacp, returning a CBOR map describing the device's ACP capabilities. The response MUST use Content-Format application/cbor and MUST conform to the following CDDL schema:

```
muacp-capabilities = {
  ? "max-tlv-size" => uint,              ; Maximum TLV region size in bytes
  ? "max-payload-size" => uint,          ; Maximum payload size in bytes
  ? "supported-tlv-types" => [*uint],    ; List of supported TLV Type values
  ? "supported-versions" => [*uint],     ; List of supported protocol versions
  ? "congestion-modes" => [*text],        ; Supported congestion control modes
  ? "conversation-limit" => uint,        ; Maximum concurrent conversations
  ? "subscription-limit" => uint         ; Maximum concurrent subscriptions
}
```

Figure 11: Feature Negotiation Response Format (CDDL)

All fields are optional. If a field is omitted, implementations MUST assume the minimum required value for that capability. Nodes SHOULD cache results until expiration or reboot. If the resource is unavailable (4.04 Not Found), implementations MUST assume default minimum capabilities: max-tlv-size=1024 (minimum required), max-payload-size=65535 (minimum required), supported-versions=[0x00]. These defaults represent minimum required capabilities; implementations MAY advertise higher limits in their feature negotiation responses.

11. Wire Examples

This section provides essential normative examples of ACP messages. Additional test vectors are available in the reference implementation repository [MUACP-IMPL]. Byte order is network byte order (big-endian).

11.1. Minimal PING (unencrypted)

A minimal PING contains only the ACP header. The complete 64-bit header is:

```
00 01   # Sequence ID = 0x0001
00 01   # Correlation ID = 0x0001
00      # QoS = 0 (fire-and-forget), Verb = 0 (PING), Flags = 0
00 00 00   # Reserved (24 bits, all zeros)
```

Total: 8 bytes

Figure 12: Example 1: PING Message (Hex)

No TLVs, no payload. This message may be sent unencrypted over CoAP NON.

11.2. ASK/TELL over OSCORE

ASK messages are sent as CoAP POST requests with OSCORE protection. The unencrypted ACP ASK structure: Header (Sequence ID=0x0002, Correlation ID=0x0003, QoS=1, Verb=2), optional TLVs, optional payload. After OSCORE encryption, the complete ACP message becomes the CoAP payload. TELL responses use the same Correlation ID and are also OSCORE-protected.

Complete Example: The following shows a complete ASK/TELL exchange:

```
Step 1: Unencrypted ACP ASK Message (before OSCORE encryption):
  Header (8 bytes):
    00 02              # Sequence ID = 0x0002
    00 03              # Correlation ID = 0x0003
    40                 # QoS = 1 (at-least-once), Verb = 2 (ASK), Flags = 0
    00 00 00           # Reserved (24 bits, all zeros)

  TLVs (optional, none in this example):
    [No TLVs]

  Payload (optional, CBOR-encoded request):
    A1                  # CBOR map with 1 key-value pair
    66 61 63 74 69 6F 6E  # "action" (UTF-8 string)
    64 72 65 61 64        # "read" (UTF-8 string)

  Total ACP message: 8 bytes (header) + 0 bytes (TLVs) + 11 bytes (payload) = 19 bytes

Step 2: CoAP POST with OSCORE (encrypted):
  CoAP Header: 44 02 7A 10  # CON, POST, MID=0x7A10
  CoAP Options:
    0B 6D 75 61 63 70  # Uri-Path: "muacp"
    11 2A              # Content-Format: application/muacp+binary (0x2A, TBD)
    09 XX              # OSCORE Option (flag byte and partial IV)

  OSCORE-Protected Payload (encrypted ACP message):
    [Encrypted ciphertext - actual value depends on OSCORE context]
    Note: The complete 19-byte ACP message from Step 1 is encrypted here

Step 3: TELL Response (unencrypted structure before OSCORE):
  Header (8 bytes):
    00 03              # Sequence ID = 0x0003
    00 03              # Correlation ID = 0x0003 (matches ASK)
    10                 # QoS = 0 (fire-and-forget), Verb = 1 (TELL), Flags = 0
    00 00 00           # Reserved (24 bits, all zeros)

  TLVs:
    22 01 00           # Error-Code TLV: Type=0x22, Length=1, Value=0x00 (SUCCESS)

  Payload (optional, CBOR-encoded result):
    A1                  # CBOR map with 1 key-value pair
    65 76 61 6C 75 65    # "value" (UTF-8 string)
    F9 41 AC             # CBOR half-precision float: 21.5

Step 4: CoAP Response with OSCORE (encrypted):
  CoAP Header: 64 44 7A 10  # ACK, 2.04 Changed, MID=0x7A10
  OSCORE-Protected Payload (encrypted ACP TELL):
    [Encrypted ciphertext - actual value depends on OSCORE context]
```

Figure 13: Example 2: Complete ASK/TELL Exchange

Complete hexdumps of encrypted payloads with actual OSCORE ciphertext are provided in the reference implementation repository, as they depend on specific OSCORE security contexts, nonces, and key material.

12. Conformance Tests

This section defines the normative conformance tests required to validate ACP implementations. A device or software stack MUST pass all tests in this section to be considered ACP-compliant. Detailed test vectors and a complete test suite are available in the reference implementation repository [MUACP-IMPL].

Implementations MUST pass tests in the following categories:

* *Header and TLV Encoding:* Correct parsing of all header fields, TLV ordering, size limits (1024 bytes TLV region, 65535 bytes payload), and handling of unknown TLVs.

* *Parser Robustness:* Safe handling of malformed headers, oversized payloads, invalid TLV lengths, and resource exhaustion conditions.

* *State-Machine Behavior:* ASK/TELL conversation lifecycle, OBSERVE subscription management, PING statelessness, and deterministic error transitions.

* *OSCORE Security:* Authentication and decryption of protected messages, replay window enforcement, context isolation, and downgrade protection.

* *Resource Constraints:* Conversation table limits (minimum 64 entries), subscription limits (minimum 16), and bounded processing time.

* *Interoperability:* Successful message exchange between independent implementations under the Minimum Interoperability Profile.

All test requirements are derived from the normative sections of this specification. Passing these tests validates full compliance with ACP.

13. References

13.1. Normative References

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

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014, <https://www.rfc-editor.org/rfc/rfc7252>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, July 2017, <https://www.rfc-editor.org/rfc/rfc8152>.

[RFC8613] Selander, G., Mattsson, J., and T. Fossati, "OSCORE: Object Security for Constrained RESTful Environments", RFC 8613, April 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, September 2015, <https://www.rfc-editor.org/rfc/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, "Blockwise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, August 2016, <https://www.rfc-editor.org/rfc/rfc7959>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9528] Selander, G., Mattsson, J., and M. Furuhed, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9528, March 2024, <https://www.rfc-editor.org/rfc/rfc9528>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014, <https://www.rfc-editor.org/rfc/rfc7228>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996, <https://www.rfc-editor.org/rfc/rfc1982>.

13.2. Informative References

[FIPA-ACL] (FIPA), F. F. I. P. A., "ACL Message Structure Specification", 1997, <https://www.fipa.org/specs/fipa00061/>.

[MUACP] Mallick, A. and I. Chebolu, "ACP: A Formal Calculus for Expressive, Resource-Constrained Agent Communication", arXiv 2601.00219, archivePrefix arXiv, primaryClass cs.MA, 2026, <https://arxiv.org/abs/2601.00219>.

[RFC9147] Rescorla, E., "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[MUACP-IMPL] Mallick, A., "ACP Reference Implementation", GitHub Repository arnab-m1/miuACP, 2025, <https://github.com/arnab-m1/miuACP>.

[AGENT-SURVEY] Finin, T., Labrou, Y., and J. Mayfield, "A Survey of Agent Communication Languages: Formalisms and Applications", Communications of ACM 40(5), DOI 10.1145/265563.265564, 1997, <https://doi.org/10.1145/265563.265564>.

[IOT-SURVEY] Palattella, M., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L., Boggia, G., and M. Dohler, "IoT Protocols for Resource-Constrained Devices: A Comparative Survey", IEEE Communications Surveys & Tutorials 18(3), DOI 10.1109/COMST.2016.2549528, 2016, <https://doi.org/10.1109/COMST.2016.2549528>.

[RPL] Winter, T. and P. Thubert, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, March 2012, <https://www.rfc-editor.org/rfc/rfc6550>.

[DDS] Group, O. M., "Data Distribution Service (DDS) for Real-Time Systems, Version 1.4", OMG Document formal/15-04-10, April 2015.

[DT] Boschert, S. and R. Rosen, "Digital Twin— The Simulation Aspect", Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and their Designers Chapter 3, Publisher Springer, DOI 10.1007/978-3-319-32156-1_3, 2016, <https://doi.org/10.1007/978-3-319-32156-1_3>.

Acknowledgments

The design of ACP benefited from feedback across multiple research and engineering communities working on IoT systems, multi-agent communication, and distributed protocol design. The authors acknowledge the valuable insights provided by early reviewers, prototype implementers, and colleagues who explored ACP in constrained-device testbeds.

Special thanks are extended to members of the open-source contributors who reviewed early drafts of the ACP calculus and provided implementation reports via the project repository. Their feedback led to refinements in the state machines, TLV model, and transport bindings.

The authors also thank participants from constrained-network and OSCORE working groups whose discussions influenced the treatment of fragmentation, replay protection, and authentication in this specification.

This specification incorporates lessons from deployments in microcontroller-based sensing systems, autonomous control nodes, and large-scale telemetry environments. The authors acknowledge these deployments for motivating the resource model and deterministic behavior guarantees underlying ACP.

This work is an independent contribution and does not represent the views of any organization or government entity.

Authors’ Addresses

Arnab Mallick
Centre for Development of Advanced Computing (CDAC)
Hyderabad
India
Email: arnabm@cdac.in

Indraveni Chebolu
Centre for Development of Advanced Computing (CDAC)
Hyderabad
India
Email: indravenik@cdac.in