

intarea
Internet-Draft
Intended status: Standards Track
Expires: 23 September 2026

X. Li
L. Sun
Y. Qiu
Nankai University
22 March 2026

Agent Transfer Protocol (ATP)
draft-li-atp-00

Abstract

The Agent Transfer Protocol (ATP) is a communication protocol designed for autonomous agents to exchange messages, requests, and events in a secure, structured manner. ATP supports the emerging Internet of Agents (IoA) paradigm, where autonomous agents operate across four deployment scenarios: household (small scale), service (medium scale), enterprise (large scale), cloud provider (huge scale), and etc.

ATP employs a two-tier architecture where agents connect to the global Internet through ATP servers, enabling server-mediated communication for proper routing, security enforcement, and resource management. The protocol provides DNS-based service discovery using SVCB records, mandatory authentication via Agent Transfer Sender (ATS) policies and Agent Transfer Keys (ATK), and support for multiple interaction patterns including asynchronous messaging, synchronous request/response, and event-driven streaming.

This specification defines the discovery mechanism, identity model, authentication framework, transport layer, and message semantics for ATP.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 23 September 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	4
1.1. Motivation	5
1.1.1. The Vision of Internet of Agents	6
1.1.2. Four Deployment Scenarios	6
1.1.3. Why ATP is Needed	7
1.1.4. Internet of Agents Architecture	9
1.2. Design Goals	11
1.3. Terminology	12
1.4. Protocol Stack Overview	13
2. Discovery Mechanism	14
2.1. DNS SVCB Record Format	14
2.1.1. Record Name	14
2.1.2. Record Format	14
2.1.3. Example	15
2.1.4. Parameters	15
2.2. Discovery Process	15
2.3. Capability Discovery	16
2.3.1. DNS-Based Capability Advertisement	16
2.3.2. HTTP-Based Capability Query	16
3. Identity Model	17
3.1. Agent Identifier Format	17
3.1.1. Examples	17
3.1.2. Local-part Semantics	17
3.1.3. Domain Requirements	18
3.2. Delegation	18
3.2.1. CNAME Delegation	18
3.2.2. SVCB Alias Mode	18
4. Security Framework	18
4.1. Transport Layer Security	18
4.1.1. TLS Requirements	18

4.1.2. Mutual TLS (Optional)	19
4.2. Agent Transfer Sender Policy (ATS)	19
4.2.1. ATS Record Format	19
4.2.2. Example ATS Records	19
4.2.3. Policy Directives	19
4.2.4. ATS Query Process	20
4.2.5. ATS Processing Algorithm	21
4.2.6. ATS Error Codes	21
4.2.7. ATS Best Practices	21
4.3. Agent Transfer Key (ATK)	22
4.3.1. ATK Record Format	22
4.3.2. Example ATK Records	22
4.3.3. Key Parameters	22
4.3.4. ATK Query Process	23
4.3.5. ATK Key Management	24
4.3.6. ATK Best Practices	24
4.4. Message Signature	24
4.4.1. Signature Envelope	24
4.4.2. Signature Algorithm	25
4.4.3. Canonicalization	25
4.4.4. Signature Verification	25
4.4.5. Signature Error Handling	26
5. Transport Protocol	26
5.1. HTTPS Transport	26
5.1.1. Default Port	26
5.1.2. Endpoints	26
5.1.3. Connection Management	27
5.2. QUIC Transport (Optional)	27
5.2.1. ALPN Identifier	27
5.2.2. Benefits	27
6. Message Format	28
6.1. Common Fields	28
6.1.1. Field Definitions	28
6.2. Message Types	29
6.2.1. Message (Asynchronous)	29
6.2.2. Request/Response	29
6.2.3. Event/Subscription	30
6.3. Payload Encoding	31
6.3.1. JSON Encoding	32
6.3.2. CBOR Encoding	32
6.4. Message Size Limits	32
7. Protocol Semantics	32
7.1. Message (Asynchronous)	32
7.1.1. Delivery Model	32
7.1.2. Reliability	33
7.2. Request/Response (Synchronous)	33
7.2.1. Interaction Model	33
7.2.2. State Management	34

7.3. Event/Subscription (Streaming)	34
7.3.1. Pub/Sub Model	34
8. Security Considerations	35
8.1. Threat Model	35
8.2. Authentication	36
8.2.1. TLS Security	36
8.2.2. ATS (Agent Transfer Sender Policy)	36
8.2.3. ATK (Agent Transfer Key)	37
8.2.4. Message Signature	37
8.3. Privacy	38
8.3.1. Metadata Exposure	38
8.3.2. Payload Confidentiality	38
8.4. Denial of Service	39
8.4.1. Resource Exhaustion Attacks	39
8.4.2. Amplification Attacks	39
8.5. DNS Security	39
8.5.1. DNS Spoofing and Cache Poisoning	39
8.5.2. DNS Query Privacy	40
8.6. ATP Server Security	40
8.6.1. Server Compromise	40
8.6.2. Multi-tenant Isolation	40
8.7. Security Best Practices	41
8.8. Known Limitations	41
9. IANA Considerations	41
9.1. Application-Layer Protocol Negotiation (ALPN) Protocol Identifier	42
9.2. Well-Known URI	42
9.3. Media Types	42
9.4. Service Name and Transport Protocol Port Number Registry	42
10. References	42
10.1. Normative References	42
10.2. Informative References	43
Appendix A. Example Message Flows	44
A.1. Sending an ATP Message	44
A.2. Request/Response Flow	45
A.3. Event Subscription Flow	47
Acknowledgments	48
Authors' Addresses	48

1. Introduction

The evolution of the Internet has always centered around the theme of "connection":

- * *Person-to-Person*: Email and instant messaging connect people with people

- * *Person-to-Service*: Web pages and mobile apps connect people with services

- * *Thing-to-Thing*: IoT devices connect the physical world

The next trend is emerging: agents will become part of Internet infrastructure.

With the advancement of artificial intelligence technology, autonomous agents are moving from concept to large-scale application:

- * Households deploy intelligent assistants to act on behalf of users for daily tasks

- * Enterprises deploy customer service bots and operations agents to automate business processes

- * Cloud providers offer agent-based capabilities including AI services and IoT coordination

When agents are ubiquitous, they need to communicate with each other - this requires a communication protocol designed specifically for agents.

The Agent Transfer Protocol (ATP) is proposed as a communication protocol design reference for the era of Internet of Agents.

1.1. Motivation

The emergence of autonomous agents represents a fundamental shift in how digital services are delivered and consumed. We are entering the era of the *Internet of Agents (IoA)*, where autonomous agents will become ubiquitous in both personal and commercial environments.

The Agent Transfer Protocol (ATP) is a communication protocol designed for autonomous agents to exchange messages, requests, and events across the Internet. ATP enables secure, structured agent-to-agent communication through a server-mediated architecture that provides routing, security enforcement, and resource management.

This document describes the motivation, architecture, and technical specifications of ATP. We first present the vision of the Internet of Agents (IoA) and the four deployment scenarios that ATP supports. We then describe the design goals, terminology, and core components of the protocol, including discovery, identity, authentication, transport, and message semantics.

1.1.1. The Vision of Internet of Agents

In the near future, autonomous agents will be deployed across four primary deployment scenarios, each serving distinct needs and scales:

***Household (Small Scale)*:** Every household will deploy personal agent systems—either as physical robots or dedicated home servers—that manage daily tasks, coordinate schedules, and interact with external services on behalf of family members. Each household member will have their own agent account, enabling personalized interactions while maintaining shared context.

***Service (Medium Scale)*:** Commercial organizations will deploy agent services to automate customer interactions, process transactions, and provide intelligent assistance. These service agents handle high-volume interactions with customers worldwide.

***Enterprise (Large Scale)*:** Large corporations and organizations will deploy internal agent systems to streamline business operations, from HR and IT support to development and operations automation.

***Cloud Provider (Huge Scale)*:** Major technology providers will operate multi-tenant agent platforms serving millions of global users, providing AI services, data management, IoT coordination, and machine learning capabilities across continents.

1.1.2. Four Deployment Scenarios

The agent ecosystem spans four distinct deployment scenarios, each with different scale requirements:

- * ***Household Agents (Small)*:** A household domain (e.g., family.example.com) hosts a few personal agent identities:
 - parent1@family.example.com - Agent for first parent
 - parent2@family.example.com - Agent for second parent
 - home@family.example.com - Home automation coordination agent
- * ***Service Agents (Medium)*:** Service providers operate multiple specialized agents:
 - service-bot@company.com - General customer service agent
 - shop-bot@retailer.com - Shopping assistant agent
 - support@tech-company.com - Technical support agent

- * *Enterprise Agents (Large)*: Organizations deploy agents for various business functions:
 - hr@enterprise.com - Human resources assistant
 - dev@enterprise.com - Development workflow agent
 - ops@enterprise.com - IT operations agent
- * *Cloud Provider Agents (Huge)*: Global platforms serve diverse user bases:
 - user-us@cloud-provider.com - North American user services
 - user-eu@cloud-provider.com - European user services
 - user-cn@cloud-provider.com - Asian user services
 - ai-service@cloud-provider.com - AI/ML service endpoints
 - iot@cloud-provider.com - IoT device coordination

1.1.3. Why ATP is Needed

Existing protocols cannot adequately meet the requirements of the Internet of Agents era:

Requirement	Description	Limitations of Existing Protocols
Multi-tenant Identity	Each domain hosts multiple agent identities (from 2-5 in households to millions on cloud platforms)	Email supports multi-user but lacks identity management for automated scenarios
Structured Data Exchange	JSON/CBOR payloads for automated processing	HTTP can transport JSON but lacks native agent semantics
Strong Security Guarantees	Mandatory authentication and integrity verification for automated operations	TLS only protects transport; application-layer authentication depends on implementation
Multiple Interaction Patterns	Asynchronous messaging, synchronous request/response, event-driven streaming	Typically requires combining multiple protocols (e.g., HTTP + WebSocket)
Scalable Discovery	Cross-internet agent location and authentication	Relies on centralized directory services or additional infrastructure
Context Awareness	Stateful multi-turn dialogues and complex workflow coordination	No native support; must be implemented at application layer
Variable Scale Support	Efficient handling from household to cloud platform scenarios	Typically optimized for specific scales, difficult to accommodate all

Table 1: Requirements for Internet of Agents and Limitations of Existing Protocols

Therefore, agents need their own communication protocol.

This vision requires a communication infrastructure that supports:

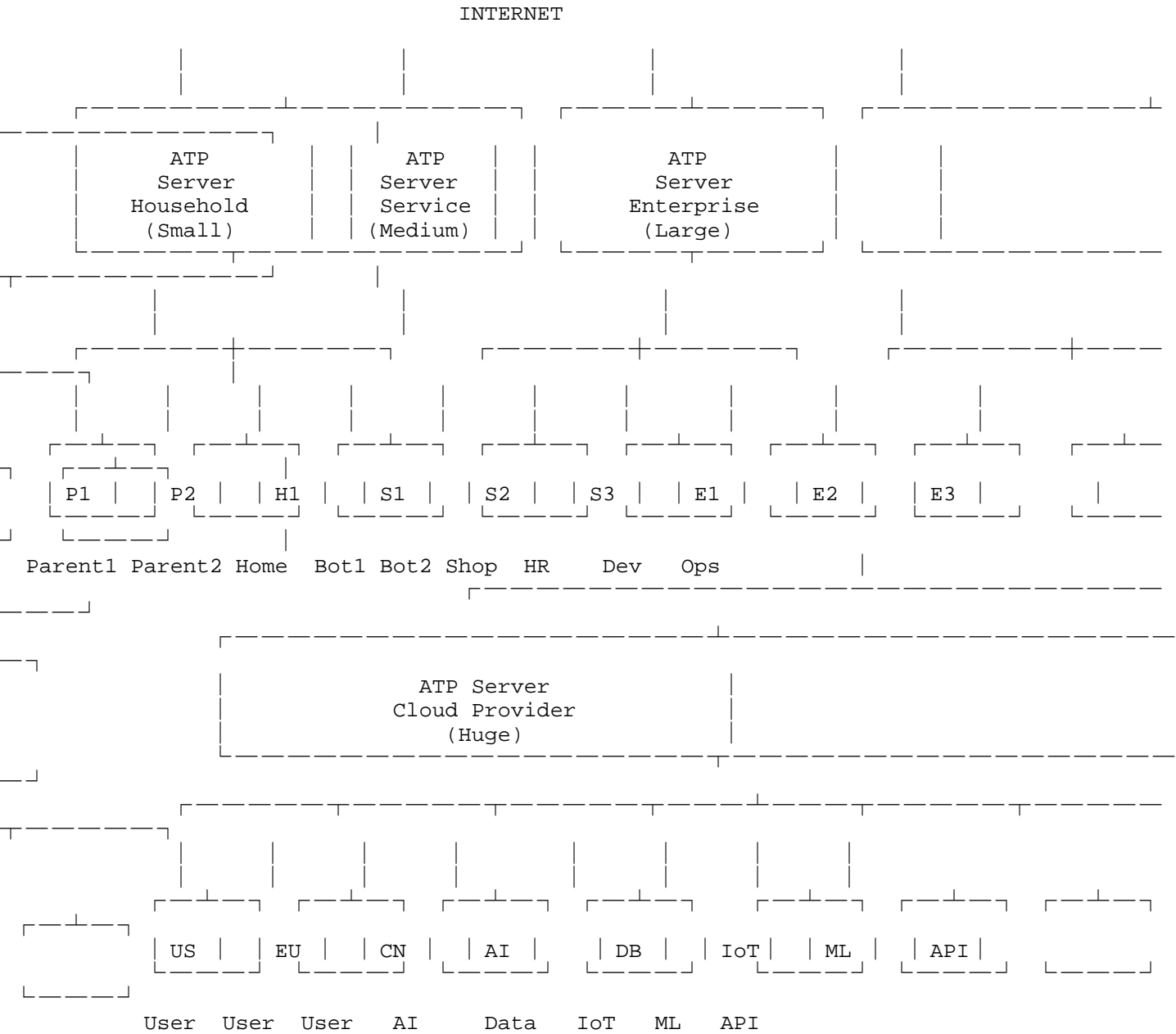
1. ***Multi-tenant Identity***: Each domain hosts multiple agent identities, from a few agents in households to thousands in cloud platforms, requiring scalable identity management and routing.
2. ***Structured Data Exchange***: Agents communicate using structured payloads (JSON, CBOR) rather than unstructured text, enabling automated processing and decision-making.
3. ***Strong Security Guarantees***: Agent communication involves automated actions on behalf of users, demanding mandatory authentication and integrity verification to prevent unauthorized operations.
4. ***Multiple Interaction Patterns***: Beyond simple messaging, agents need:
 - * Synchronous request/response for RPC-style interactions
 - * Event-driven streaming for real-time updates
 - * Asynchronous messaging for notifications and broadcasts
5. ***Scalable Discovery***: Agents must locate and authenticate each other across the global internet using existing, proven infrastructure.
6. ***Context Awareness***: Agents maintain state across interactions, support multi-turn dialogues, and coordinate complex workflows involving multiple parties.
7. ***Variable Scale Support***: The protocol must efficiently handle scenarios ranging from small household deployments (2-5 agents) to massive cloud platforms (millions of users), with appropriate resource allocation and routing strategies for each scale.

ATP addresses these requirements by providing a modern, secure, and extensible protocol built upon existing internet infrastructure while introducing agent-specific semantics.

1.1.4. Internet of Agents Architecture

The Internet of Agents (IoA) follows a two-tier architecture where agents connect to the global internet through ATP servers. This design ensures proper resource allocation, security enforcement, and efficient routing.

Internet of Agents (IoA)



Household (Small): Personal/Family agents (Parent1/2, Home Assistant)
Service (Medium): Commercial service agents (Service Bot1/2, Shopping)
Enterprise (Large): Corporate agents (HR Bot, Dev Bot, Ops Bot)
Cloud Provider (Huge): Multi-tenant cloud serving global users (US, EU, CN, AI, Data, IoT, ML, API)

Figure 1: Internet of Agents (IoA) Architecture: Four deployment scenarios showing Household (Small), Service (Medium), Enterprise (Large), and Cloud Provider (Huge) ATP server categories

In this architecture:

* *ATP Servers* act as gateways for agents, handling:

- Resource allocation and scheduling for local agents
 - Outbound connection management and routing
 - Inbound traffic filtering and security enforcement
 - Policy enforcement (ATS/ATK validation)
 - Message queuing and delivery guarantees
- * **Agents** operate within their ATP Server's domain:
- All outbound communication goes through their local ATP Server
 - All inbound communication arrives via their ATP Server
 - Agents benefit from server-side security and filtering
 - Multiple agents share server resources efficiently
- * **Scale Considerations**:
- **Household (Small)**: 2-10 agents, minimal resource requirements
 - **Service (Medium)**: 10-100 agents, moderate traffic handling
 - **Enterprise (Large)**: 100-1000+ agents, high availability requirements
 - **Cloud Provider (Huge)**: Millions of users, global distribution, multi-region deployment

This architecture reflects the reality that in the Internet of Agents era, every household and organization will deploy an ATP server (or manager) to coordinate local agent activities, manage network connections, and provide security boundaries.

1.2. Design Goals

1. **Security-first**: Authentication and integrity verification are mandatory, not optional.
2. **Structured semantics**: Native support for multiple interaction patterns (message, request/response, event/subscription).
3. **DNS-based discovery**: Leverage existing DNS infrastructure for agent discovery and service location.

4. **Transport agnostic**: Support multiple transport protocols (HTTPS, QUIC) for flexibility and performance.
5. **Backward compatible**: Coexist with existing internet infrastructure and standards.
6. **Extensible**: Support capability discovery and protocol negotiation for future evolution.
7. **Multi-tenant support**: Efficiently handle multiple agents per domain, similar to multi-user email systems.
8. **Server-mediated communication**: All agent communication MUST flow through ATP servers for proper routing, security enforcement, and resource management. This design reflects the reality that agents operate within managed domains (households, organizations, cloud services) that require centralized coordination.

1.3. 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.

This document uses the following terms:

- * **Agent**: An autonomous software entity capable of communication, decision-making, and task execution.
- * **Agent ID**: A unique identifier in the format local@domain that identifies a specific agent.
- * **ATP Server**: A server implementing the ATP protocol and handling agent communication. The ATP Server acts as a gateway for agents within its domain, providing resource allocation, connection management, security enforcement, and message routing. All agent communication MUST flow through their respective ATP servers.
- * **ATP Client**: An agent or application initiating ATP communication.
- * **ATP Transfer**: The process of transferring messages between ATP servers across the Internet.

- * ***ATS***: Agent Transfer Sender policy - a DNS record defining authorized senders for a domain.
- * ***ATK***: Agent Transfer Key - a DNS record publishing cryptographic keys for message signature verification.
- * ***SVCB***: Service Binding - DNS record type defined in [RFC9460] for service discovery.
- * ***ALPN***: Application-Layer Protocol Negotiation - TLS extension for protocol negotiation.

1.4. Protocol Stack Overview

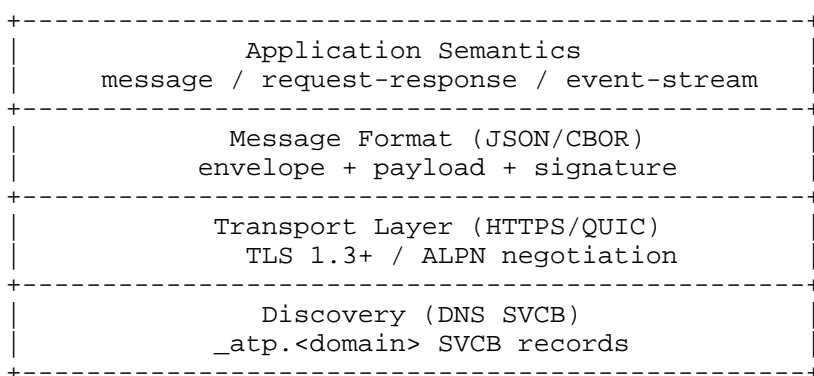


Figure 2: ATP Protocol Stack: Four-layer architecture with Discovery (DNS SVCB), Transport (HTTPS/QUIC), Message Format (JSON/CBOR), and Application Semantics

ATP is built upon a layered protocol stack that leverages existing Internet infrastructure while introducing agent-specific semantics. The stack consists of four layers, from bottom to top:

Discovery Layer: Uses DNS SVCB records [RFC9460] to enable agents to locate and discover ATP services for any given domain. The discovery query targets _atp.<domain> to retrieve service endpoint information including hostname, port, and supported protocols.

Transport Layer: Supports multiple transport protocols for flexibility and performance:

- * ***HTTPS***: Primary transport protocol, enabling deployment behind existing CDNs, load balancers, and firewalls. Uses TLS 1.3+ for encryption and ALPN for protocol negotiation.

- * ***QUIC***: Optional transport protocol [RFC9000] for low-latency scenarios, providing 0-RTT handshakes and connection migration capabilities.

Message Format Layer: Defines the structure of ATP messages using structured data encodings:

- * ***JSON***: Recommended encoding [RFC8259] with Content-Type application/atp+json
- * ***CBOR***: Optional encoding [RFC8949] for bandwidth-constrained environments with Content-Type application/atp+cbor

Messages consist of an envelope (containing from, to, timestamp, nonce, type) and a payload, with a cryptographic signature for integrity and authenticity.

Application Semantics Layer: Provides three interaction patterns for different use cases:

- * ***Message***: Asynchronous, fire-and-forget communication
- * ***Request/Response***: Synchronous RPC-style interactions
- * ***Event/Subscription***: Streaming and publish-subscribe patterns

2. Discovery Mechanism

2.1. DNS SVCB Record Format

ATP uses DNS SVCB (Service Binding) records [RFC9460] for agent discovery. The SVCB record provides the hostname, port, protocol version, and optional connection hints for the ATP service. The DNS query follows the standard DNS protocol [RFC1034][RFC1035].

2.1.1. Record Name

The standard SVCB query name for ATP discovery is:

`_atp.<domain>`

Where <domain> is the domain portion of the recipient Agent ID.

2.1.2. Record Format

```
_atp.<domain>. IN SVCB <priority> <target> (  
  port=<port-number>  
  alpn=<protocol-identifier>  
  [ipv4hint=<ipv4-address>]  
  [ipv6hint=<ipv6-address>]  
  [atp-capabilities=<capability-list>]  
)
```

2.1.3. Example

```
_atp.example.com. IN SVCB 1 agent.example.com. (  
  port=7443  
  alpn="atp/1"  
  ipv4hint=192.0.2.1,192.0.2.2  
  ipv6hint=2001:db8::1,2001:db8::2  
  atp-capabilities="message,request,event"  
)
```

2.1.4. Parameters

- * *port*: TCP/UDP port number for the ATP service. If not specified, the default port is 7443.
- * *alpn*: Application-layer protocol negotiation identifier. Supported values:
 - atp/1 - ATP version 1
 - atp-json - ATP with JSON payload encoding
 - atp-cbor - ATP with CBOR payload encoding
 - atp+proto - ATP with Protocol Buffers encoding
- * *ipv4hint* (optional): Comma-separated list of IPv4 addresses for faster connection establishment.
- * *ipv6hint* (optional): Comma-separated list of IPv6 addresses for faster connection establishment.
- * *atp-capabilities* (optional): Comma-separated list of supported protocol capabilities.

2.2. Discovery Process

The ATP client performs the following steps to discover the recipient's ATP service:

1. ***Extract Domain***: Parse the recipient Agent ID to extract the domain portion.
2. ***DNS SVCB Query***: Query the SVCB record for `_atp.<domain>`.
3. ***Fallback Query***: If the query fails, attempt `_agent.<domain>` for backward compatibility.
4. ***Address Resolution***: Resolve the target hostname to IP addresses using A/AAAA records.
5. ***Connection Establishment***: Establish a secure connection to the resolved endpoint using the specified port.

2.3. Capability Discovery

Agents can advertise their capabilities to enable protocol negotiation and feature discovery.

2.3.1. DNS-Based Capability Advertisement

Capabilities can be included in the SVCB record using the `atp-capabilities` parameter:

```
_atp.example.com. IN SVCB 1 agent.example.com. (  
    port=7443  
    alpn="atp/1"  
    atp-capabilities="message,request,event,payment,search"  
)
```

2.3.2. HTTP-Based Capability Query

Agents can query capabilities via an HTTP endpoint:

```
GET /.well-known/atp/v1/capabilities  
Host: agent.example.com  
Accept: application/json
```

Response:

```
{
  "version": "1.0",
  "capabilities": [
    "message",
    "request",
    "event",
    "payment",
    "search"
  ],
  "protocols": ["atp/1", "atp-json"],
  "max_payload_size": 1048576,
  "rate_limits": {
    "messages_per_second": 100,
    "requests_per_minute": 1000
  }
}
```

3. Identity Model

3.1. Agent Identifier Format

Agent identifiers follow the standard email address format but with extended semantics for agent communication.

agent-id = local-part "@" domain

The local-part identifies a specific agent within the domain, and the domain identifies the ATP server responsible for routing messages to that agent.

3.1.1. Examples

- * al@example.com - Individual agent
- * chatbot@service.org - Service agent
- * billing.taskbot@example.com - Task-specific agent with hierarchical naming
- * weather-agent-v2@provider.net - Versioned agent identifier

3.1.2. Local-part Semantics

The local-part can use alphanumeric characters, period (.), hyphen (-), underscore (_), and plus (+).

Domains MUST support case-insensitive matching of local-parts, similar to email systems.

3.1.3. Domain Requirements

The domain portion MUST be a valid Internationalized Domain Name (IDN) as defined in [RFC5890]. ATP servers MUST support both ASCII and UTF-8 encoded domain names.

3.2. Delegation

Domains can delegate agent handling to external ATP servers using DNS CNAME records or SVCB alias mode.

3.2.1. CNAME Delegation

```
_atp.user.example.com. IN CNAME _atp.provider.net.
```

This allows users to maintain their identity (@user.example.com) while using third-party ATP services.

3.2.2. SVCB Alias Mode

```
_atp.delegated.example.com. IN SVCB 0 _atp.target.org.
```

SVCB alias mode provides a more modern delegation mechanism with additional metadata capabilities.

4. Security Framework

4.1. Transport Layer Security

All ATP connections MUST use TLS 1.3 [RFC8446] or higher. TLS 1.2 MAY be used for backward compatibility, but its use is NOT RECOMMENDED.

4.1.1. TLS Requirements

- * ***Mandatory Encryption***: All ATP traffic MUST be encrypted using TLS.
- * ***Certificate Validation***: Clients MUST validate server certificates against trusted Certificate Authorities (CAs).
- * ***Cipher Suites***: Servers SHOULD support modern cipher suites (e.g., TLS_AES_128_GCM_SHA256).
- * ***Forward Secrecy***: Ephemeral key exchange (ECDHE) MUST be used.

4.1.2. Mutual TLS (Optional)

For high-security environments, ATP servers MAY require mutual TLS (mTLS) authentication.

4.2. Agent Transfer Sender Policy (ATS)

The Agent Transfer Sender policy (ATS) defines which entities are authorized to send ATP messages on behalf of a domain. ATS is designed specifically for agent communication with strict enforcement.

4.2.1. ATS Record Format

ATS policies are published as DNS TXT records at the following location:

```
ats._atp.<domain>. IN TXT "v=atp1 <policy-directives>"
```

4.2.2. Example ATS Records

Simple IP-based policy:

```
ats._atp.example.com. IN TXT "v=atp1 allow=ip:192.0.2.0/24"
```

Multi-source policy:

```
ats._atp.example.com. IN TXT "v=atp1 allow=ip:192.0.2.0/24 allow=domain:agent-provider.com include=ats._atp.trusted-partner.net"
```

Permissive policy:

```
ats._atp.example.com. IN TXT "v=atp1 allow=all"
```

Restrictive policy:

```
ats._atp.example.com. IN TXT "v=atp1 deny=all allow=ip:192.0.2.10"
```

4.2.3. Policy Directives

ATS policy directives are evaluated in order, with later directives overriding earlier ones when conflicts occur.

* **v=atp1**: Version identifier (REQUIRED). Indicates ATP version 1 policy format.

* **allow=ip:<cidr>**: Authorize messages from specific IP address ranges.

- * `*allow=domain:<domain>*`: Authorize messages from agents at the specified domain.
- * `*allow=all*`: Authorize messages from any source (NOT RECOMMENDED).
- * `*deny=ip:<cidr>*`: Explicitly deny messages from specific IP ranges.
- * `*deny=domain:<domain>*`: Explicitly deny messages from agents at the specified domain.
- * `*deny=all*`: Deny all sources (must be followed by specific allow directives).
- * `*include:<record>*`: Include policy from another ATS record.
- * `*redirect=<domain>*`: Redirect policy evaluation to another domain.
- * `*exp=<domain>*`: Specify a domain for explanation text (human-readable).

4.2.4. ATS Query Process

When an ATP server receives a message claiming to be from `sender@domain.com`, it performs the following ATS verification:

1. `*Query ATS Record*`: Query the DNS TXT record for `ats._atp.domain.com`.
2. `*Extract IP*`: Determine the source IP address of the incoming connection.
3. `*Policy Evaluation*`: Evaluate the ATS policy against the source IP and sender domain.
4. `*Authorization Decision*`:
 - * If policy evaluation results in PASS, accept the message.
 - * If policy evaluation results in FAIL, reject the message with error code 550 5.7.26 ATS validation failed.
 - * If no ATS record exists, treat as NEUTRAL (accept but flag for monitoring).

4.2.5. ATS Processing Algorithm

The ATS verification algorithm processes policy directives as follows:

```
result = NEUTRAL
```

```
FOR EACH directive in policy:
```

```
  IF directive is 'allow=ip:<cidr>' AND source_ip matches <cidr>:
```

```
    result = PASS
```

```
  ELSE IF directive is 'deny=ip:<cidr>' AND source_ip matches <cidr>:
```

```
    result = FAIL
```

```
  ELSE IF directive is 'allow=domain:<domain>' AND sender_domain matches <domain>:
```

```
    result = PASS
```

```
  ELSE IF directive is 'deny=domain:<domain>' AND sender_domain matches <domain>:
```

```
    result = FAIL
```

```
  ELSE IF directive is 'allow=all':
```

```
    result = PASS
```

```
  ELSE IF directive is 'deny=all':
```

```
    result = FAIL
```

```
  ELSE IF directive is 'include:<record>':
```

```
    include_result = query_ats(<record>)
```

```
    IF include_result is PASS or FAIL:
```

```
      result = include_result
```

```
RETURN result
```

4.2.6. ATS Error Codes

ATP servers MUST use the following error codes for ATS failures:

- * *550 5.7.26*: ATS validation failed - sender not authorized
- * *451 4.7.26*: Temporary ATS validation failure - try again later
- * *550 5.7.27*: ATS record syntax error

4.2.7. ATS Best Practices

- * *Start Restrictive*: Begin with a restrictive policy and gradually add authorized sources.
- * *Use Includes*: Use include: directives to inherit policies from trusted providers.
- * *Monitor Logs*: Regularly review ATS validation logs to identify unauthorized attempts.

- * ***Gradual Deployment***: Deploy ATS gradually, starting with monitoring mode before enforcement.

4.3. Agent Transfer Key (ATK)

The Agent Transfer Key (ATK) record publishes cryptographic public keys used for message signature verification. ATK is mandatory in ATP and uses modern cryptographic algorithms.

4.3.1. ATK Record Format

ATK records are published as DNS TXT records at the following location:

```
<selector>.atk._atp.<domain>. IN TXT "v=atp1 <key-parameters>"
```

Where <selector> is an identifier for the specific key (e.g., default, 2026q1, rotated-key).

4.3.2. Example ATK Records

***Ed25519 key (RECOMMENDED)*:**

```
default.atk._atp.example.com. IN TXT "v=atp1 k=ed25519 p=MCowBQYDK2VwAyEAtLJ5VqH7K+R5VZ8cD9XwY3J2mN8K+R5VZ8cD9XwY3J2m"
```

***RSA key (2048-bit)*:**

```
legacy.atk._atp.example.com. IN TXT "v=atp1 k=rsa p=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA..."
```

***ECDSA key (P-256)*:**

```
p256.atk._atp.example.com. IN TXT "v=atp1 k=ecdsa n=prime256v1 p=MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE..."
```

4.3.3. Key Parameters

- * ***v=atp1***: Version identifier (REQUIRED). Indicates ATP version 1 key format.
- * ***k=<algorithm>***: Key algorithm (REQUIRED). Supported values:
 - ed25519 - Ed25519 (RECOMMENDED for new deployments)
 - rsa - RSA (2048-bit minimum)
 - ecdsa - ECDSA (P-256, P-384, or P-521)
- * ***p=<base64-key>***: Base64-encoded public key (REQUIRED).

- * ***n=<curve-name>***: Elliptic curve name (REQUIRED for ECDSA). Supported values: prime256v1, secp384r1, secp521r1.
- * ***h=<hash-alg>***: Hash algorithm (OPTIONAL). Defaults to sha256. Supported values: sha256, sha384, sha512.
- * ***s=<service>***: Service type (OPTIONAL). Indicates which ATP services use this key.
- * ***t=<flags>***: Flags (OPTIONAL). Comma-separated list of flags:
 - y - Testing mode (signature validation is informational only)
 - s - Key is revoked (do not use for new signatures)
- * ***x=<expiry>***: Key expiry timestamp (OPTIONAL). Unix timestamp after which the key should not be used.

4.3.4. ATK Query Process

When an ATP server receives a signed message, it performs the following ATK verification:

1. ***Extract Signature Information***: Parse the message signature to extract key selector, domain, signature algorithm, and signature value.
2. ***Query ATK Record***: Query the DNS TXT record for <selector>.atk._atp.<domain>.
3. ***Validate Key Parameters***: Verify that the key algorithm and parameters match the signature.
4. ***Canonicalization***: Canonicalize the message payload according to the specified algorithm.
5. ***Signature Verification***: Verify the signature using the public key from the ATK record.
6. ***Validation Decision***:
 - * If signature verification succeeds, accept the message.
 - * If signature verification fails, reject the message with error code 550 5.7.28 ATK signature validation failed.
 - * If ATK record is not found, reject the message with error code 550 5.7.29 ATK key not found.

4.3.5. ATK Key Management

4.3.5.1. Key Rotation

Domains SHOULD rotate ATK keys periodically (e.g., every 90 days).

4.3.5.2. Key Revocation

If a key is compromised, it SHOULD be revoked immediately by updating the ATK record to include the s flag.

4.3.6. ATK Best Practices

- * ***Use Ed25519***: Prefer Ed25519 keys for new deployments due to their security and performance characteristics.
- * ***Key Size***: For RSA keys, use at least 2048 bits. For ECDSA, use P-256 or stronger curves.
- * ***Multiple Keys***: Maintain multiple keys (current, next, previous) for smooth rotation.
- * ***DNSSEC***: Sign ATK records with DNSSEC to prevent DNS spoofing attacks.
- * ***Monitoring***: Monitor signature validation failures to detect potential attacks or misconfigurations.

4.4. Message Signature

All ATP messages MUST be cryptographically signed to ensure integrity and authenticity.

4.4.1. Signature Envelope

The signature is included in the message envelope as a signature field:

```
{
  "from": "sender@example.com",
  "to": "recipient@example.com",
  "timestamp": 1710000000,
  "nonce": "msg-12345-abcde",
  "type": "message",
  "payload": {},
  "signature": {
    "key_id": "default.atk._atp.example.com",
    "algorithm": "ed25519",
    "signature": "MEUCIQDR...",
    "headers": ["from", "to", "timestamp", "nonce", "type"],
    "timestamp": 1710000000
  }
}
```

4.4.2. Signature Algorithm

The signature algorithm depends on the key type: - **Ed25519**: Sign the canonicalized message bytes directly. - **RSA**: Use RSASSA-PSS with SHA-256 (or stronger). - **ECDSA**: Use ECDSA with the specified curve and hash algorithm.

4.4.3. Canonicalization

Messages MUST be canonicalized before signing to ensure consistent signature verification. ATP uses JSON Canonicalization Scheme (JCS) [RFC8785] for JSON payloads.

Canonicalization Process:

1. Remove the signature field from the message object.
2. Apply JCS canonicalization to the remaining JSON object.
3. Convert the canonicalized JSON to UTF-8 bytes.
4. Sign the UTF-8 bytes using the specified algorithm.

4.4.4. Signature Verification

The recipient verifies the signature as follows:

1. Extract the signature field from the message.
2. Reconstruct the message object without the signature field.
3. Apply the same canonicalization process used by the sender.

4. Verify the signature using the public key from the ATK record.
5. Check that signed headers match the actual message headers.

4.4.5. Signature Error Handling

If signature verification fails, the recipient **MUST** reject the message and **MAY** log the failure for monitoring.

5. Transport Protocol

5.1. HTTPS Transport

The primary transport for ATP is HTTPS, enabling easy deployment behind existing infrastructure such as CDNs, load balancers, and firewalls.

5.1.1. Default Port

The default port for ATP over HTTPS is ***7443***. However, ATP servers **MAY** use any port, which is specified in the SVCB record.

5.1.2. Endpoints

ATP servers **MUST** implement the following standard endpoints:

5.1.2.1. Send Message Endpoint

POST /.well-known/atp/v1/message
Content-Type: application/atp+json

Request Body: ATP message envelope

Response:

- * *202 Accepted***: Message accepted for delivery
- * *400 Bad Request***: Invalid message format
- * *401 Unauthorized***: Authentication failed
- * *429 Too Many Requests***: Rate limit exceeded
- * *500 Internal Server Error***: Server error

5.1.2.2. Capability Discovery Endpoint

GET /.well-known/atp/v1/capabilities

Response: JSON object describing server capabilities

5.1.2.3. Health Check Endpoint

GET /.well-known/atp/v1/health

Response:

```
{
  "status": "ok",
  "version": "1.0.0",
  "uptime": 86400,
  "load": 0.45
}
```

5.1.3. Connection Management

- * *Keep-Alive*: Clients SHOULD use HTTP keep-alive for multiple requests.
- * *Timeout*: Servers SHOULD implement idle timeout (RECOMMENDED: 60 seconds).
- * *Rate Limiting*: Servers MAY implement rate limiting with appropriate HTTP headers.

5.2. QUIC Transport (Optional)

For low-latency scenarios, ATP supports QUIC transport [RFC9000].

5.2.1. ALPN Identifier

The ALPN identifier for ATP over QUIC is atp/1.

5.2.2. Benefits

- * *0-RTT Handshake*: Establish connections with zero round-trip time.
- * *Multiplexing*: Multiple ATP messages over a single connection.
- * *Connection Migration*: Maintain connections across network changes.

- * **Better Performance**: Reduced latency over lossy networks.

6. Message Format

6.1. Common Fields

All ATP messages share a common envelope structure with the following fields:

```
{
  "from": "string",           // Sender Agent ID (REQUIRED)
  "to": "string",             // Recipient Agent ID (REQUIRED)
  "cc": ["string"],           // Carbon copy recipients (OPTIONAL)
  "timestamp": "integer",     // Unix timestamp in seconds (REQUIRED)
  "nonce": "string",          // Unique message identifier (REQUIRED)
  "type": "string",           // Message type (REQUIRED)
  "payload": "object",        // Message-specific content (REQUIRED)
  "signature": "object",      // Signature envelope (REQUIRED)
  "routing": "object"         // Routing information (OPTIONAL)
}
```

6.1.1. Field Definitions

- * **from**: The Agent ID of the message sender. MUST be a valid agent identifier.
- * **to**: The Agent ID of the primary recipient. MUST be a valid agent identifier.
- * **cc**: Array of Agent IDs for carbon-copy recipients.
- * **timestamp**: Unix timestamp (seconds since epoch) when the message was created.
- * **nonce**: Cryptographically random unique identifier for the message.
- * **type**: Message type indicator. Values: message, request, response, event.
- * **payload**: Message-specific content. Structure depends on message type.
- * **signature**: Cryptographic signature envelope.
- * **routing**: Routing information for multi-hop scenarios.

6.2. Message Types

ATP supports three primary message types, each with distinct semantics and use cases.

6.2.1. Message (Asynchronous)

The message type is used for asynchronous, fire-and-forget communication.

```
{
  "from": "a1@example.com",
  "to": "a2@example.com",
  "timestamp": 1710000000,
  "nonce": "msg-12345-abcde",
  "type": "message",
  "payload": {
    "subject": "Hello from Agent A1",
    "body": "This is an asynchronous message",
    "attachments": [
      {
        "name": "data.json",
        "content_type": "application/json",
        "data": "base64-encoded-data"
      }
    ],
    "priority": "normal"
  },
  "signature": {}
}
```

6.2.2. Request/Response

The request/response pattern supports synchronous RPC-style interactions.

6.2.2.1. Request Format

```
{
  "from": "client@example.com",
  "to": "service@example.org",
  "timestamp": 1710000000,
  "nonce": "req-67890-fghij",
  "type": "request",
  "payload": {
    "action": "get_weather",
    "params": {
      "location": "New York",
      "units": "metric"
    },
    "timeout": 30,
    "correlation_id": "corr-12345"
  },
  "signature": {}
}
```

6.2.2.2. Response Format

```
{
  "from": "service@example.org",
  "to": "client@example.com",
  "timestamp": 1710000001,
  "nonce": "resp-67890-klmno",
  "type": "response",
  "in_reply_to": "req-67890-fghij",
  "payload": {
    "status": "success",
    "data": {
      "temperature": 22,
      "conditions": "sunny",
      "humidity": 65
    },
    "correlation_id": "corr-12345"
  },
  "signature": {}
}
```

6.2.3. Event/Subscription

The event type supports streaming and event-driven communication.

6.2.3.1. Subscription Request

```
{
  "from": "subscriber@example.com",
  "to": "publisher@example.org",
  "timestamp": 1710000000,
  "nonce": "sub-11111-pqrst",
  "type": "request",
  "payload": {
    "action": "subscribe",
    "event_types": ["price_update", "news", "alert"],
    "filter": {
      "symbol": "AAPL",
      "priority": ">=high"
    },
    "delivery_mode": "push",
    "subscription_id": "sub-12345"
  },
  "signature": {}
}
```

6.2.3.2. Event Message

```
{
  "from": "publisher@example.org",
  "to": "subscriber@example.com",
  "timestamp": 1710000010,
  "nonce": "evt-22222-uvwxy",
  "type": "event",
  "payload": {
    "event_type": "price_update",
    "subscription_id": "sub-12345",
    "data": {
      "symbol": "AAPL",
      "price": 150.25,
      "timestamp": 1710000010,
      "volume": 1000000
    },
    "sequence_number": 42
  },
  "signature": {}
}
```

6.3. Payload Encoding

ATP supports multiple payload encoding formats.

6.3.1. JSON Encoding

JSON [RFC8259] is the RECOMMENDED encoding format.

Content-Type: application/atp+json

6.3.2. CBOR Encoding

CBOR [RFC8949] is an OPTIONAL encoding format for bandwidth-constrained environments.

Content-Type: application/atp+cbor

6.4. Message Size Limits

ATP servers SHOULD enforce message size limits:

* *Default Maximum*: 1 MB (1,048,576 bytes)

* *Minimum Supported*: 64 KB (65,536 bytes)

7. Protocol Semantics

7.1. Message (Asynchronous)

Asynchronous messages follow a store-and-forward model.

7.1.1. Delivery Model

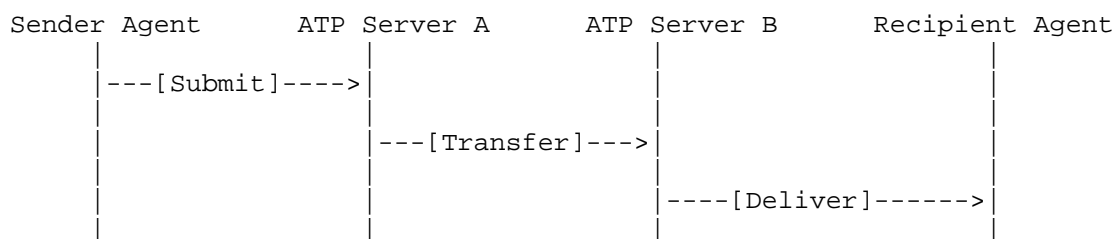


Figure 3: Asynchronous Message Delivery Model: Submit → Transfer → Deliver flow through ATP servers

In this model: 1. The Sender Agent submits a message to its local ATP Server A 2. ATP Server A performs policy enforcement (ATS/ATK validation) and transfers the message to ATP Server B 3. ATP Server B performs security checks and delivers the message to the Recipient Agent

Each hop (Submit/Transfer/Deliver) involves independent ATS/ATK validation for security enforcement.

7.1.2. Reliability

- * ***Store-and-Forward***: Messages are stored on intermediate servers until delivery succeeds.
- * ***Retry Mechanism***: Failed deliveries are retried with exponential backoff.
- * ***Bounce Handling***: Undeliverable messages generate bounce notifications to the sender.

7.2. Request/Response (Synchronous)

Synchronous request/response follows an RPC pattern.

7.2.1. Interaction Model

All agent communication **MUST** go through their respective ATP servers. This design ensures proper routing, security filtering, and policy enforcement.

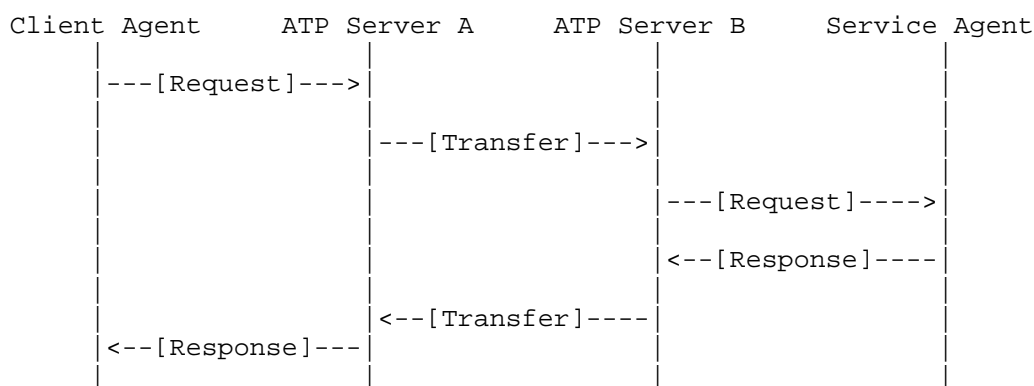


Figure 4: Request/Response Interaction Model: Synchronous RPC-style flow with bidirectional server-mediated communication

In this model: 1. The Client Agent submits a request to its local ATP Server A via HTTP POST 2. ATP Server A performs policy enforcement (ATS/ATK validation) and transfers the request to ATP Server B 3. ATP Server B performs security checks and delivers the request to the Service Agent 4. The Service Agent processes the request and returns a response 5. The response flows back through the same path (Service Agent → ATP Server B → ATP Server A → Client Agent)

Each hop (Submit/Transfer/Deliver) involves independent ATS/ATK validation for security enforcement.

7.2.2. State Management

- * ***Stateless***: Each request is independent.
- * ***Correlation ID***: Clients MAY include a `correlation_id` to track workflows.
- * ***Idempotency***: Requests SHOULD be idempotent.

7.3. Event/Subscription (Streaming)

Event/subscription follows a publish-subscribe pattern.

7.3.1. Pub/Sub Model

All event publications and subscriptions MUST go through ATP servers for proper routing and access control.

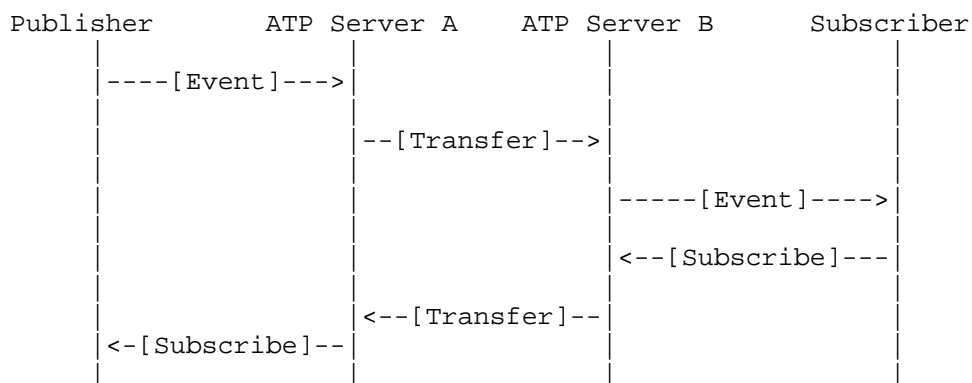


Figure 5: Event/Subscription Pub/Sub Model: Bidirectional flow for subscription establishment and event notification

The event/subscription flow consists of two phases:

***Subscription Phase*:** The Subscriber initiates a subscription request to its local ATP Server, which transfers the request to the Publisher's ATP Server, which then delivers it to the Publisher. This establishes the subscription relationship across the server chain.

***Event Notification Phase*:** When an event occurs, the Publisher sends the event to its local ATP Server (Server B), which transfers it across the Internet to the Subscriber's ATP Server (Server A), which then delivers the notification to the Subscriber.

This bidirectional flow ensures: - Proper access control at each ATP server boundary - ATS/ATK policy enforcement for both subscription and event messages - Subscription state management at the Publisher's server

8. Security Considerations

ATP is designed with security as a first-class requirement. This section analyzes the threat model and security considerations for each component of the protocol.

8.1. Threat Model

The threat model for ATP considers the following adversaries and attack vectors:

***Network Adversaries*:** Passive or active attackers on the network path between agents and ATP servers, or between ATP servers. Capabilities may include:

- * Eavesdropping on unencrypted traffic
- * Man-in-the-middle (MitM) attacks to intercept or modify communications
- * Replay attacks using captured messages
- * Traffic analysis to infer communication patterns

***Malicious Agents*:** Compromised or malicious agents that attempt to:

- * Send unauthorized messages on behalf of other agents (spoofing)
- * Flood servers with excessive requests (DoS)
- * Exploit protocol vulnerabilities to gain unauthorized access

***Rogue ATP Servers*:** Compromised or malicious ATP servers that may:

- * Fail to enforce ATS/ATK policies
- * Leak sensitive message content
- * Drop or delay messages selectively

***DNS Attackers*:** Adversaries that attempt to compromise DNS infrastructure:

- * DNS cache poisoning to redirect ATP traffic
- * DNS spoofing to provide fraudulent SVCB records
- * DNS amplification attacks

8.2. Authentication

8.2.1. TLS Security

***Threat*:** Network adversaries attempting eavesdropping or MitM attacks.

***Mitigation*:**

- * All ATP connections MUST use TLS 1.3 [RFC8446] or higher
- * Certificate validation against trusted Certificate Authorities is mandatory
- * Cipher suites MUST provide forward secrecy (ECDHE key exchange)
- * Cipher suites SHOULD use authenticated encryption (AES-GCM, ChaCha20-Poly1305)

***Residual Risk*:** Certificate authority compromise or mis-issuance. Deployments with high security requirements SHOULD implement certificate pinning or use DANE [RFC6698].

8.2.2. ATS (Agent Transfer Sender Policy)

***Threat*:** Malicious agents attempting to send messages on behalf of domains they do not control (spoofing).

***Mitigation*:**

- * ATS records define authorized sending sources per domain

- * Policy directives include IP ranges, domains, and explicit deny rules
 - * Each ATP server performs ATS validation on every incoming message
 - * Failed ATS validation results in immediate rejection
- *Residual Risk*: ATS record tampering via DNS attacks. Deployments SHOULD sign ATS records with DNSSEC [RFC4033].

8.2.3. ATK (Agent Transfer Key)

- *Threat*: Message tampering or forgery by network adversaries or malicious agents.
- *Mitigation*:
- * ATK records publish cryptographic public keys for signature verification
 - * All ATP messages MUST be cryptographically signed
 - * Supported algorithms: Ed25519 (RECOMMENDED), RSA (2048+ bit), ECDSA (P-256+)
 - * Signature covers all critical message fields (from, to, timestamp, nonce, type, payload)
- *Residual Risk*: Key compromise. Domains MUST implement key rotation and maintain revocation procedures via the s flag in ATK records.

8.2.4. Message Signature

- *Threat*: Message modification in transit, replay attacks.
- *Mitigation*:
- * Cryptographic signatures cover canonicalized message content
 - * Nonce prevents replay attacks
 - * Timestamp enables expiration checking
 - * Signature includes headers list to prevent header manipulation
- *Residual Risk*: Long-term key compromise enables retroactive forgery. Future work may introduce forward-secure signature schemes.

8.3. Privacy

8.3.1. Metadata Exposure

***Threat*:** Traffic analysis revealing communication patterns, relationships, or sensitive business information.

***Exposure*:**

- * from and to fields are visible to ATP servers and network observers
- * timestamp reveals timing of communications
- * type indicates interaction pattern (message, request, response, event)

***Mitigation*:**

- * Payload content SHOULD be encrypted end-to-end when confidentiality is required
- * Agents MAY use pseudonymous identifiers for sensitive communications
- * ATP servers SHOULD minimize metadata logging

***Residual Risk*:** Metadata analysis remains possible. Applications with strong privacy requirements SHOULD implement additional obfuscation at the application layer.

8.3.2. Payload Confidentiality

***Threat*:** Unauthorized access to message content by ATP servers or network adversaries.

***Mitigation*:**

- * TLS encryption protects payload in transit between hops
- * Agents MAY encrypt payload content end-to-end using recipient's public key
- * CBOR encoding supports embedded encrypted content

***Residual Risk*:** ATP servers must inspect messages for policy enforcement. Deployments handling sensitive data SHOULD implement server-side encryption with customer-managed keys.

8.4. Denial of Service

8.4.1. Resource Exhaustion Attacks

***Threat*:** Adversaries flooding ATP servers with excessive messages or connections.

***Attack Vectors*:**

- * Message flooding to exhaust server storage or bandwidth
- * Connection exhaustion to deplete server connection pools
- * Computational DoS via expensive cryptographic operations

***Mitigation*:**

- * Rate limiting based on sender reputation and domain
- * Message size limits (default 1 MB maximum)
- * Connection limits per sender IP and domain
- * Exponential backoff for retry attempts
- * Quota enforcement per agent and per domain

8.4.2. Amplification Attacks

***Threat*:** Attackers using ATP to amplify traffic toward victims.

***Mitigation*:**

- * ATS validation prevents unauthorized relaying
- * Response messages only sent to request initiators
- * Subscription confirmation required before event delivery
- * No broadcast or multicast mechanisms

8.5. DNS Security

8.5.1. DNS Spoofing and Cache Poisoning

***Threat*:** Attackers providing fraudulent DNS responses to redirect ATP traffic.

***Mitigation*:**

- * ATP servers SHOULD validate DNS responses using DNSSEC [RFC4033]
- * SVCB records SHOULD be cached with appropriate TTL
- * Multiple independent DNS resolvers SHOULD be consulted

***Residual Risk*:** DNSSEC adoption is not universal. Applications requiring high assurance SHOULD implement additional verification.

8.5.2. DNS Query Privacy

***Threat*:** DNS queries revealing which domains agents communicate with.

***Mitigation*:**

- * DNS-over-HTTPS (DoH) or DNS-over-TLS (DoT) for query confidentiality
- * DNS query caching to reduce query frequency
- * Batch queries when discovering multiple domains

8.6. ATP Server Security

8.6.1. Server Compromise

***Threat*:** Compromised ATP server leaking or modifying messages.

***Mitigation*:**

- * End-to-end message signatures detect modification
- * Payload encryption protects content from server inspection
- * Audit logging for security monitoring
- * Regular security updates and hardening

8.6.2. Multi-tenant Isolation

***Threat*:** One tenant's agents accessing or interfering with another tenant's agents.

***Mitigation*:**

- * Strict domain-based access control
- * Per-tenant quota enforcement
- * Isolated processing contexts
- * ATS/ATK validation per message

8.7. Security Best Practices

Deployments SHOULD implement the following security practices:

1. *DNSSEC*: Sign all DNS zones containing ATS and ATK records
2. *Key Rotation*: Rotate ATK keys every 90 days minimum
3. *Monitoring*: Log and alert on ATS/ATK validation failures
4. *Certificate Management*: Monitor certificate expiration and implement automated renewal
5. *Incident Response*: Maintain procedures for key revocation and emergency policy updates
6. *Defense in Depth*: Implement multiple layers of security controls

8.8. Known Limitations

1. *Metadata Visibility*: ATP does not hide communication metadata from ATP servers
2. *DNS Trust*: DNS security depends on DNSSEC adoption
3. *Server Trust*: ATP servers can observe unencrypted payload content
4. *Key Management*: Compromised keys enable message forgery until revocation

Future revisions of ATP may address these limitations through techniques such as onion routing, encrypted DNS, or decentralized trust models.

9. IANA Considerations

9.1. Application-Layer Protocol Negotiation (ALPN) Protocol Identifier

IANA is requested to register the following ALPN protocol identifier:

- * *Protocol*: atp/1
- * *Identification Sequence*: 0x61 0x74 0x70 0x2f 0x31 (atp/1)
- * *Specification*: This document

9.2. Well-Known URI

IANA is requested to register the following well-known URI:

- * *URI Suffix*: atp
- * *Change Controller*: IETF
- * *Specification*: This document

9.3. Media Types

IANA is requested to register the following media types:

- * *application/atp+json*
- * *application/atp+cbor*

9.4. Service Name and Transport Protocol Port Number Registry

IANA is requested to register the following service:

- * *Service Name*: atp
- * *Port Number*: 7443
- * *Transport Protocol*: TCP, UDP
- * *Description*: Agent Transfer Protocol

10. References

10.1. Normative References

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

- [RFC5890] Klensin, J. and RFC Series, "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", DOI 10.17487/RFC5890, RFC 5890, August 2010, <<https://doi.org/10.17487/RFC5890>>.
- [RFC8174] Leiba, B. and RFC Series, "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", DOI 10.17487/RFC8174, BCP 14, RFC 8174, May 2017, <<https://doi.org/10.17487/RFC8174>>.
- [RFC8259] Bray, T., Ed. and RFC Series, "The JavaScript Object Notation (JSON) Data Interchange Format", DOI 10.17487/RFC8259, STD 90, RFC 8259, December 2017, <<https://doi.org/10.17487/RFC8259>>.
- [RFC8446] Rescorla, E. and RFC Series, "The Transport Layer Security (TLS) Protocol Version 1.3", DOI 10.17487/RFC8446, RFC 8446, August 2018, <<https://doi.org/10.17487/RFC8446>>.
- [RFC8785] Rundgren, A., Jordan, B., Erdtman, S., and RFC Series, "JSON Canonicalization Scheme (JCS)", DOI 10.17487/RFC8785, RFC 8785, June 2020, <<https://doi.org/10.17487/RFC8785>>.
- [RFC8949] Bormann, C., Hoffman, P., and RFC Series, "Concise Binary Object Representation (CBOR)", DOI 10.17487/RFC8949, STD 94, RFC 8949, December 2020, <<https://doi.org/10.17487/RFC8949>>.
- [RFC9000] Iyengar, J., Ed., Thomson, M., Ed., and RFC Series, "QUIC: A UDP-Based Multiplexed and Secure Transport", DOI 10.17487/RFC9000, RFC 9000, May 2021, <<https://doi.org/10.17487/RFC9000>>.
- [RFC9110] Fielding, R., Ed., Nottingham, M., Ed., Reschke, J., Ed., and RFC Series, "HTTP Semantics", DOI 10.17487/RFC9110, STD 97, RFC 9110, June 2022, <<https://doi.org/10.17487/RFC9110>>.
- [RFC9460] Schwartz, B., Bishop, M., Nygren, E., and RFC Series, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", DOI 10.17487/RFC9460, RFC 9460, November 2023, <<https://doi.org/10.17487/RFC9460>>.

10.2. Informative References

- [RFC1034] Mockapetris, P. and RFC Series, "Domain names - concepts and facilities", DOI 10.17487/RFC1034, STD 13, RFC 1034, November 1987, <<https://doi.org/10.17487/RFC1034>>.
- [RFC1035] Mockapetris, P. and RFC Series, "Domain names - implementation and specification", DOI 10.17487/RFC1035, STD 13, RFC 1035, November 1987, <<https://doi.org/10.17487/RFC1035>>.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., Rose, S., and RFC Series, "DNS Security Introduction and Requirements", DOI 10.17487/RFC4033, RFC 4033, March 2005, <<https://doi.org/10.17487/RFC4033>>.
- [RFC6698] Hoffman, P., Schlyter, J., and RFC Series, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", DOI 10.17487/RFC6698, RFC 6698, August 2012, <<https://doi.org/10.17487/RFC6698>>.

Appendix A. Example Message Flows

A.1. Sending an ATP Message

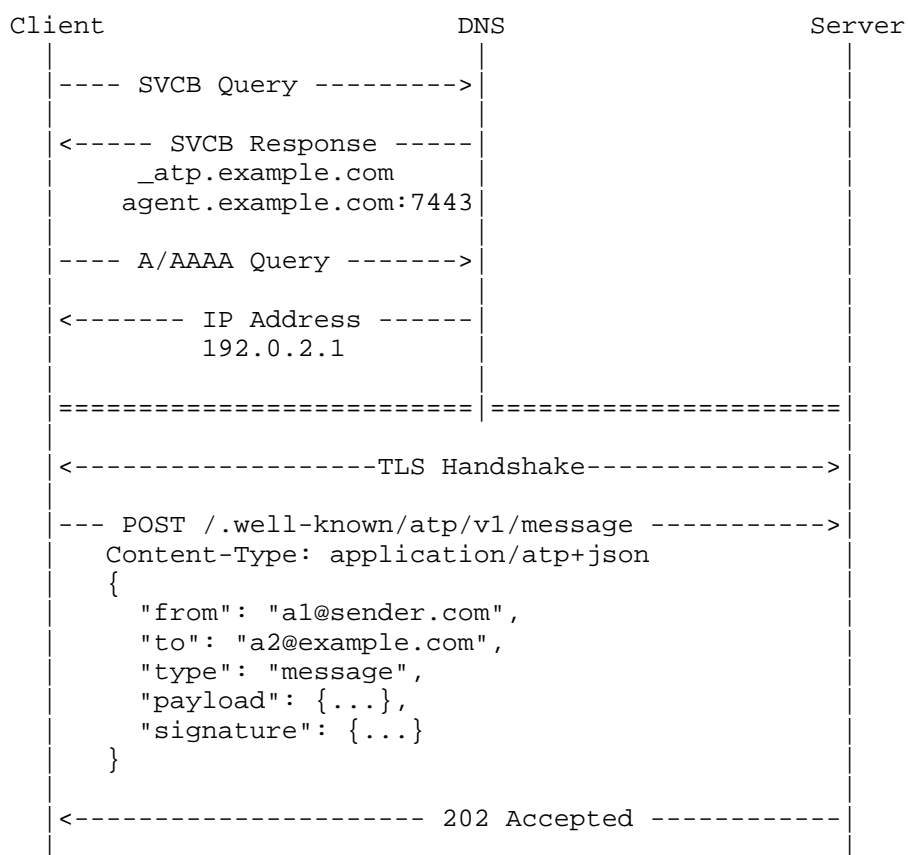
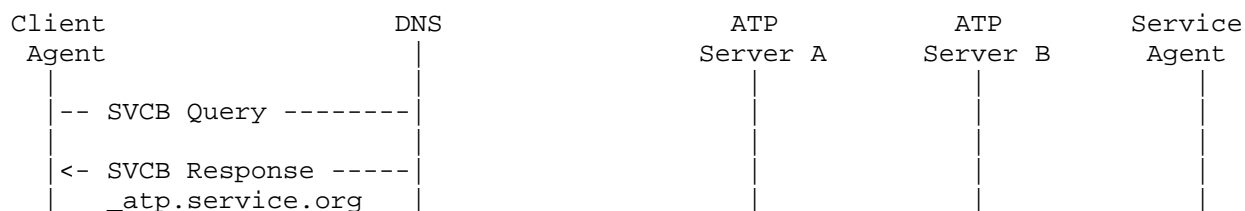


Figure 6: Sending an ATP Message: DNS discovery, TLS handshake, and message submission flow

A.2. Request/Response Flow

The request/response flow demonstrates synchronous RPC-style interaction between agents. The Client Agent connects to its local ATP Server (Server A), which enforces policy and transfers the request to the destination ATP Server (Server B), which delivers it to the Service Agent. The response follows the reverse path.



<pre> service.org:7443 -- A/AAAA Query -----> <- IP Address ----- 198.51.100.1 ===== <-----TLS Handshake-----> -- POST /.well-known/atp/v1/message -----> Content-Type: application/atp+json { "from": "client@example.com", "to": "service@service.org", "timestamp": 1710000000, "nonce": "req-67890-fghij", "type": "request", "payload": { "action": "get_weather", "params": {"location": "New York"} }, "signature": {...} } </pre>	<pre> -[Transfer]-> </pre>	<pre> -[Request]--> </pre>
<pre> <- 202 Accept/Response ----- { "from": "service@service.org", "to": "client@example.com", "timestamp": 1710000002, "nonce": "resp-67890-klmno", "type": "response", "in_reply_to": "req-67890-fghij", "payload": { "status": "success", "data": {"temperature": 22} }, "signature": {...} } </pre>	<pre> <-[Transfer]- </pre>	<pre> <-[Response]- </pre>

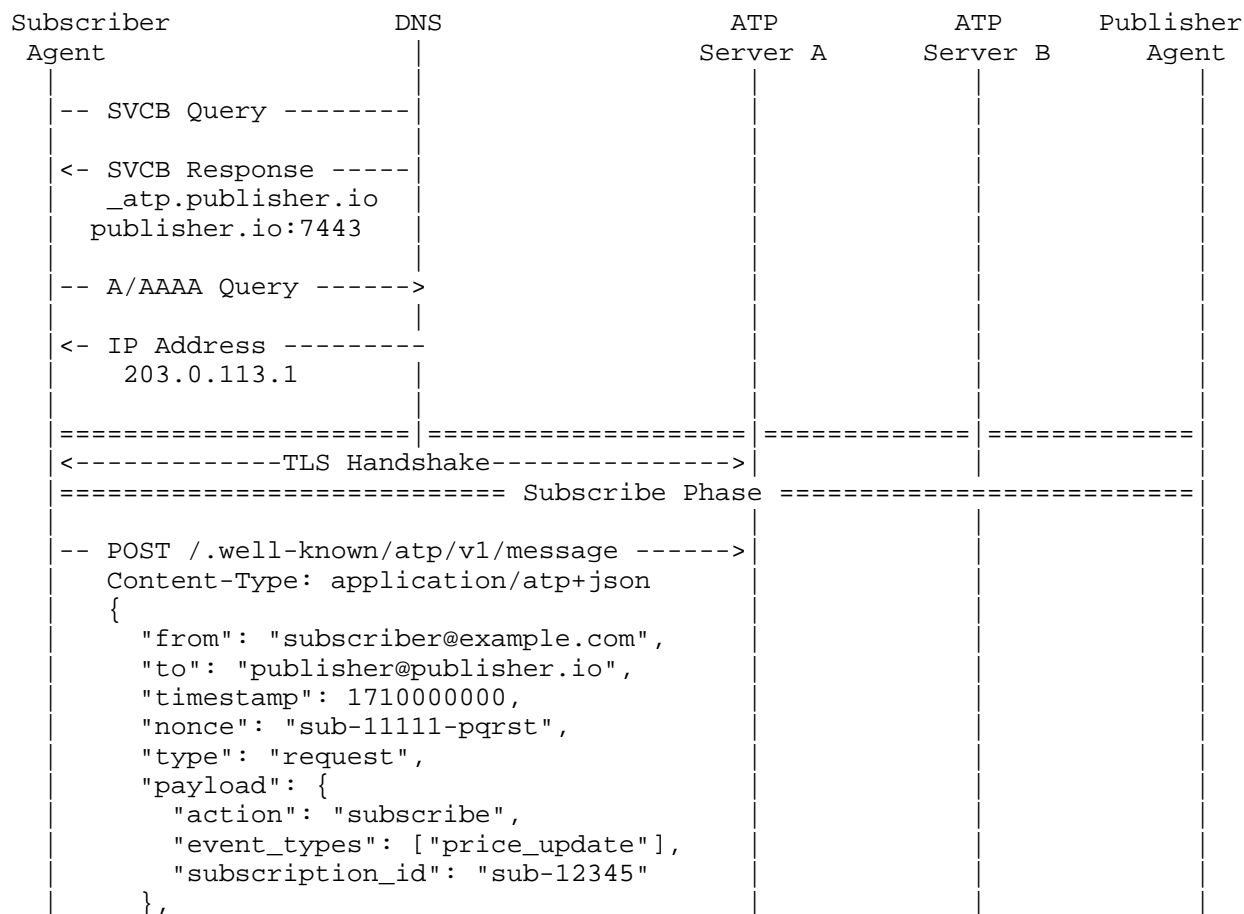
Figure 7: Request/Response Flow: DNS discovery, TLS handshake, request submission, server transfer, and response return

A.3. Event Subscription Flow

The event subscription flow demonstrates the publish-subscribe pattern with streaming notifications. The flow has two phases:

***Phase 1 - Subscribe*:** The Subscriber sends a subscription request to its local ATP Server (Server A), which transfers it to the Publisher's ATP Server (Server B), which delivers it to the Publisher.

***Phase 2 - Event Notification*:** When an event occurs, the Publisher sends the event to Server B, which transfers it to Server A, which delivers the notification to the Subscriber.



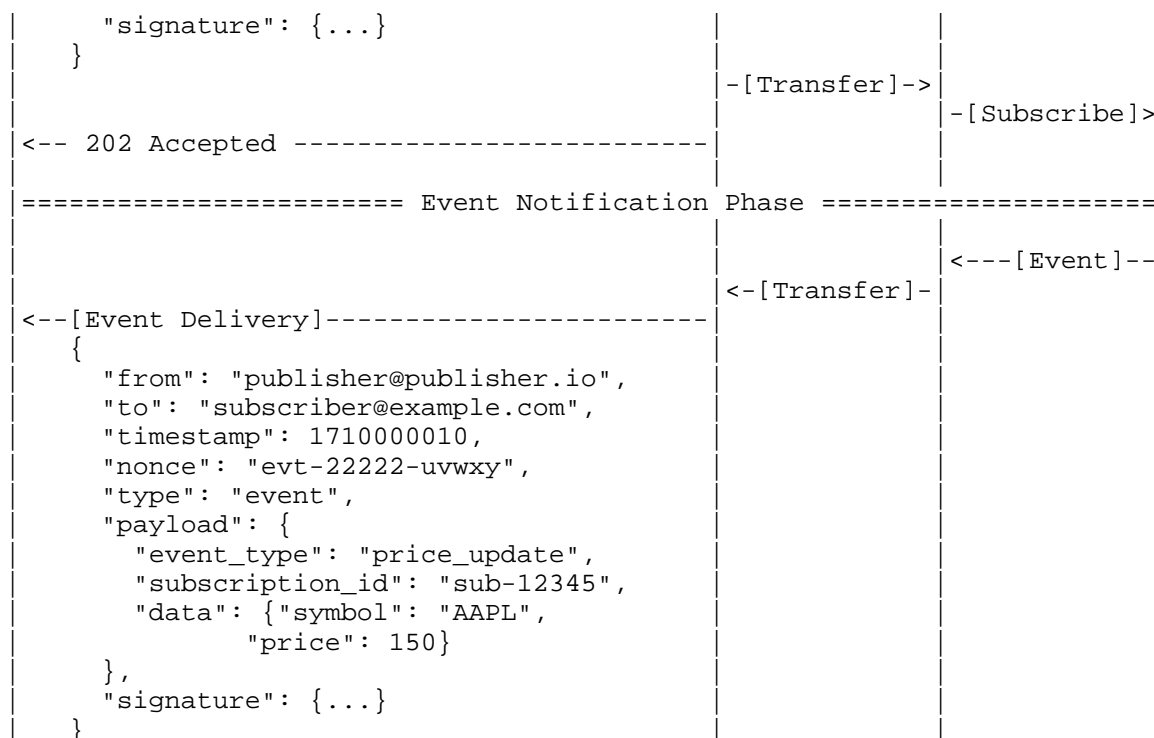


Figure 8: Event Subscription Flow: Two-phase publish-subscribe
with Subscribe Phase and Event Notification Phase

Acknowledgments

This protocol draws inspiration from multiple existing protocols and standards:

- * DNS SVCB [RFC9460] - for service discovery
- * HTTP/2 [RFC9110] - for multiplexing
- * QUIC [RFC9000] - for low-latency transport
- * TLS [RFC8446] - for secure transport

Authors' Addresses

Xiang Li
Nankai University
Tongyan Road
Tianjin
300350
China
Email: lixiang@nankai.edu.cn

Lu Sun
Nankai University
Tongyan Road
Tianjin
300350
China
Email: sunlu25@mail.nankai.edu.cn

Yuqi Qiu
Nankai University
Tongyan Road
Tianjin
300350
China
Email: qiuyuqi@mail.nankai.edu.cn