Open-Wire: A Channel Transport Layer for Human-in-the-Loop Agents

A universal bidirectional protocol between AI programs on any hardware and human-connected communication channels

Kannan Reghu
Tiny Bubble Company / Rails Agent
kannan@rails-agent.com · openwire.rails-agent.com/research

Working paper · July 31, 2026 · Protocol: open-wire/1

Abstract

Autonomous software agents increasingly run on arbitrary hardware—developer laptops, cloud runtimes, edge devices, and future robotic controllers—yet still depend on humans for approval, clarification, escalation, and notification. Today every agent framework reinvents how to reach those humans through Slack, Microsoft Teams, email, SMS, WhatsApp, voice, and similar surfaces. This paper argues that the missing layer is not another agent-to-agent or agent-to-tool protocol, but a *channel transport*: a thin, bi-directional envelope that lets any AI program send and receive text (and, later, voice) on the communication channels humans already use, without owning each channel's OAuth, webhooks, and delivery semantics. Drawing on the separation of applications and transport in the early internet, we propose Open-Wire (`open-wire/1`) as a universal connection layer. We situate the claim against MCP, A2A, emerging Agent-to-Human (A2H) intent protocols, HITL confirmation drafts such as CHEQ, Matrix, and commercial unified messaging APIs. We present architectural principles, protocol semantics, an implemented reference gateway used by Rails Agent for Slack today, and outline how the same transport generalizes to a robot-heavy future where humans remain addressable principals in the loop.

1. Introduction

The internet scaled because of a deliberate separation of concerns. Applications innovated on top of a thin, shared transport layer—TCP/IP, then HTTP—without each website inventing its own cables. AI agents today lack this separation. A developer building an agent must simultaneously solve (1) reasoning, tool use, and memory within the agent framework, and (2) connecting that agent to the messaging surfaces where humans already work. Every framework hits the same wall: rebuilding OAuth flows, webhook verification, message threading, rate limiting, and outbound delivery for each channel.

The thesis of Open-Wire is deliberately narrow and deliberately physical:

> *An AI computer program, running anywhere on hardware, should be able to send a text or voice message to any human-connected communication channel—and receive a reply—as part of a human-in-the-loop flow, without re-implementing that channel's integration stack.*

That is not the claim that Slack equals WhatsApp. It is the claim that grants, envelopes, threads, and delivery belong in a shared layer, while channel-native fidelity belongs in adapters—exactly as TCP did not replace HTTP, and HTTP did not replace HTML. Frameworks on the left stay focused on reasoning. Channels on the right stay where humans already work. Open-Wire in the middle carries the conversation both ways: install once, speak once, reach many.

2. Problem Statement

2.1 Human-in-the-loop needs

HITL agent systems need humans for at least: **Inform** (notify without waiting), **Collect** (ask for missing data), **Authorize** (gate a side effect), and **Escalate** (hand off when autonomy fails). Empirically those interactions already happen on consumer and workplace channels—not inside the agent's private UI.

2.2 The integration tax

Deploying agents into production requires connecting them to existing communication infrastructure. Each channel is a distinct surface:

Concern	Slack	Teams	WhatsApp	Email
Authentication	OAuth 2.0	Azure AD	Business API	SMTP/IMAP
Event format	Events API	Bot Framework	Webhooks	IMAP IDLE
Threading	thread_ts	conversation.id	Reply context	In-Reply-To
Rich content	Block Kit	Adaptive Cards	Limited	HTML/MIME
Files	Upload API	Graph API	Media API	MIME

A team supporting five channels must understand five authentication flows, five event formats, five threading models, and five retry strategies.

2.3 The rebuild problem and framework pollution

Because no universal layer exists, each framework reinvents channel connectivity in Python, Ruby, JavaScript, and Go—for each channel. When channel concerns leak into agent frameworks, frameworks designed for reasoning must understand Block Kit schemas, Adaptive Cards, and WhatsApp templates. This pollution constrains evolution and ties agent logic to presentation details. Rebuilding also duplicates security surface (token handling), fragments audit trails, and locks products to one vendor's bot SDK.

3. Related Work

The 2024–2026 agent protocol landscape is best read as a *stack*, not a competition.

3.1 MCP (agent ↔ tools)

Anthropic's Model Context Protocol standardizes how agents connect to data sources and tools. MCP answers "what can this agent operate on?" It does not answer "how does this agent reach a human on Slack?" Open-Wire is complementary—the last mile of agent-to-human channel communication.

3.2 A2A (agent ↔ agent)

Google's Agent-to-Agent protocol (now under the Linux Foundation) standardizes discovery, tasks, and messaging between opaque agents. Humans remain external unless another layer makes them addressable.

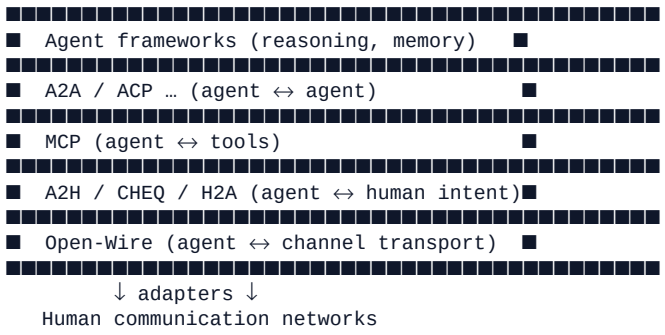
3.3 A2H / CHEQ / H2A (agent ↔ human intent)

Twilio's open A2H specification defines INFORM, COLLECT, AUTHORIZE, ESCALATE, and RESULT intents with channel-agnostic delivery and cryptographic evidence for approvals. IETF draft CHEQ focuses on signed confirmations so hallucinations cannot silently authorize actions. Community H2A efforts standardize frontend↔agent presence for in-app UIs. These specify *what* the agent needs from a human. Open-Wire specifies *how bytes reach the human's channel* and return.

3.4 Unified messaging APIs and Matrix

Commercial unified messaging APIs (Twilio, MessageBird, Sendbird) abstract multiple channels behind one API, but are proprietary and often lack bidirectional richness. Matrix is an open decentralized communication system with bridges; Open-Wire is intentionally thinner—a translation layer at the agent-to-channel boundary, not a full identity and client ecosystem.

4. Positioning: a layered stack



Without the bottom layer, A2H intents still require someone to speak Slack and Twilio. Without intent protocols, Open-Wire is “only chat.” Production HITL needs both.

5. Architectural Principles

- **Reasoning vs. transport.** Frameworks handle planning, tools, and memory. Open-Wire handles delivery, authentication, and channel semantics.
- **Content vs. presentation.** Agents produce structured content; channels render natively (Block Kit, Adaptive Cards, HTML).
- **Identity vs. addressing.** Agents have persistent identities; Open-Wire maps them to Slack user IDs, emails, phone numbers.
- **Conversation vs. threading.** Agents maintain conversation state; Open-Wire maps turns to channel-specific thread models.
- **Hardware-agnostic origin.** The sender is any process that can HTTPS—cloud agent, laptop REPL, edge box, robot controller. Location is irrelevant; the installation grant matters.
- **Envelope, not homogenization.** Channel-native payloads travel in body extensions; the protocol guarantees identity and direction, not pixel parity.
- **Installations as capability grants.** OAuth binds a workspace to a webhook. Tokens stay in the gateway, not in every agent repo.
- **Modality extension.** Voice is a body modality (audio / STT-TTS), not a separate protocol family.

6. Protocol Design

6.1 The open-wire/1 envelope (implemented)

Version `open-wire/1` is the JSON envelope used by the reference gateway. Inbound (channel → agent):

```
{
  "protocol": "open-wire/1",
  "type": "message.inbound",
  "id": "msg_…",
```

```
  "channel": "slack",
  "installation_id": "inst_…",
  "thread": { "id": "C123:1710000000.000100" },
  "from": { "id": "U123", "kind": "user" },
  "to": { "id": "C123", "kind": "channel" },
  "body": { "text": "approve the refund", "blocks": null },
  "created_at": "2026-07-30T00:00:00.000Z"
}
```

Outbound (agent → channel):

```
{
  "protocol": "open-wire/1",
  "type": "message.send",
  "installation_id": "inst_…",
  "to": { "id": "C123", "kind": "channel" },
  "thread": { "id": "C123:1710000000.000100" },
  "body": { "text": "Refund queued. Reply YES to confirm.", "blocks": [] }
}
```

Agent webhooks carry `X-Open-Wire-Protocol` and `X-Open-Wire-Installation`. Live API: installations, OAuth start/callback, Slack Events hook, and `POST /api/v1/messages` (see https://openwire.rails-agent.com/docs/api).

6.2 Content types and capability negotiation

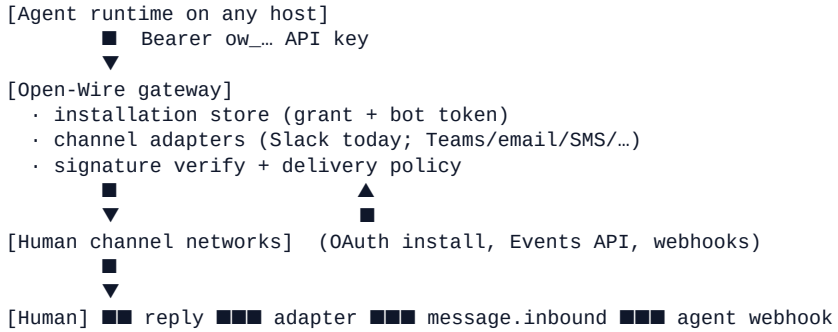
Open-Wire defines minimal content types—text, rich, action, file, typing, reaction, presence—that adapters map to channel capabilities. Not all channels support all features. The gateway advertises capabilities so agents can degrade (e.g., send a link instead of a rich card) without knowing WhatsApp specifics.

```
{
  "channel": "whatsapp",
  "capabilities": {
    "supports_rich_content": false,
    "supports_threading": false,
    "supports_files": true,
    "max_message_length": 4096,
    "supported_content_types": ["text", "file"]
  }
}
```

6.3 Bidirectional flow

Inbound: Channel emits native event → adapter normalizes to Open-Wire → gateway routes to agent. **Outbound:** Agent produces envelope → gateway selects channel → adapter renders native payload → delivery, retries, rate limits. Channel failures surface as transport-agnostic errors (`channel_unavailable`, `rate_limited`, `message_rejected`, `authentication_expired`, `recipient_unreachable`).

7. Reference Architecture



Layers: Application (agent framework) knows only open-wire/1; Gateway validates, routes, and maps identity; Adapters own channel auth, webhooks, and rendering; Channels remain native platforms. A new channel requires only a new adapter—existing agents gain reach without code changes. Gateway deployment may be embedded, standalone, or managed SaaS. Agents never hold channel credentials.

8. HITL Patterns on Open-Wire

- **INFORM** → `message.send` without awaiting inbound.
- **COLLECT / AUTHORIZE** → send with affordances (buttons or reply keywords); correlate `thread.id` on inbound.
- **ESCALATE** → send to a human queue channel or DM; optional secondary installation.

Cryptographic evidence (CHEQ/A2H) should wrap the decision object; Open-Wire carries the conversation that produces it.

9. Comparative Analysis

Approach	LOC / channel	Framework change	New channel cost
Native SDK	500–2000	Yes	N frameworks × 1 channel
Unified API	100–300	Yes (vendor API)	N frameworks × 1 API
Open-Wire	50–100 (client)	No	1 gateway adapter

Under Open-Wire the agent need not know channel auth, addressing formats, content schemas, rate limits, or threading models. Framework changes for a new channel are not required.

10. Voice and the Robot Future

If the future is robots, the thesis widens rather than weakens. Robots and opaque agents still need principals: owners, operators, customers. A2A will connect robots to robots; MCP will connect them to tools; humans will still approve irreversible actions and receive situational awareness on channels they already check. Voice fits as modality: STT on inbound audio, TTS on outbound, or PSTN/SIP adapters behind the same installation and thread identity. The protocol holds when it remains a message-bus contract to endpoints—human channels today and additional endpoint classes tomorrow.

11. Limitations and Open Problems

- **Fidelity loss.** Universal protocols risk LCD behavior; mitigated by `metadata.channel_raw`, capability negotiation, and escape hatches.
- **Adapter burden.** Centralizing integrations concentrates maintenance (acceptable if shared).
- **Human identity.** Open-Wire addresses channel IDs; cross-channel human cards remain A2H work.
- **Abuse.** Universal outbound needs rate limits, consent, and workspace policy—especially SMS/WhatsApp.
- **Adoption.** Protocols need network effects; open reference implementation bootstraps them.
- **Formalization.** JSON Schema, conformance tests, and an RFC-style document are required for multi-implementer adoption.

12. Implementation Status

As of this writing, the Open-Wire reference gateway is public at openwire.rails-agent.com, with Slack OAuth install, Events API inbound, outbound `message.send`, and Rails Agent Cloud integration for Channels Connect and workflow notifications. Source: github.com/Tiny-Bubble-Company/open-wire.

13. Conclusion

Agent frameworks should own reasoning. Intent protocols should own what the human is being asked. Channel transport should own how a message leaves a process on any hardware and arrives on a human network—and how the reply returns. Open-Wire is a working bet on that bottom layer: `open-wire/1` envelopes, installations as grants, adapters for fidelity. In a robot-heavy future, humans do not exit the system; they become scarcer, higher-leverage endpoints. Those endpoints still need a wire. The transport layer should not live inside your agent framework. It should be universal.

References

1. Anthropic. Model Context Protocol. https://modelcontextprotocol.io
2. Linux Foundation / Google. Agent2Agent (A2A) Protocol. https://a2a-protocol.org
3. Singh, R. & Ferguson, R. (2026). Introducing A2H. Twilio.
4. A2H: Agent-to-Human Protocol for AI Agent. arXiv:2602.15831.
5. Rosenberg, J. et al. CHEQ. IETF draft-rosenberg-aiproto-cheq-00.
6. H2A — Human-to-Agent Protocol. https://github.com/PDangelmaier/h2a-protocol
7. Matrix.org Foundation. Matrix Specification.
8. Fielding, R. (2000). Architectural Styles and the Design of Network-based Software Architectures.
9. Postel, J. (1981). Transmission Control Protocol. RFC 793.
10. Open-Wire protocol reference. https://openwire.rails-agent.com/docs/protocol

---

Author contact
Kannan Reghu · Tiny Bubble Company / Rails Agent
kannan@rails-agent.com
Paper: https://openwire.rails-agent.com/research

This is a working paper intended to invite critique, implementations, and conformance work—not a claim of IETF standardization. Feedback welcome at the address above.