rtgwg
Internet-Draft
Intended status: Informational
Expires: 7 May 2026

R. Zhang
J. Mao
B. Liu
N. Geng
X. Shang
Q. Gao
Z. Li
Huawei

3 November 2025

Use Cases and Requirements of Communication Protocol for Troubleshooting Agents on Network Devices
draft-zhang-rtgwg-ai-agents-troubleshooting-00

Abstract

This document focuses on the use cases and requirements of communication protocols for troubleshooting agents on network devices.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://example.com/LATEST. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-zhang-rtgwg-ai-agents-troubleshooting/.

Discussion of this document takes place on the rtgwg Working Group mailing list (mailto:WG@example.com), which is archived at https://example.com/WG.

Source for this draft and an issue tracker can be found at https://github.com/USER/REPO.

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/.

Copyright Notice

Table of Contents

1. Introduction

This document focus on communication protocols and associated requirements for network troubleshooting interactions among agents on network devices. As modern networks evolve toward greater complexity and dynamism, traditional centralized management systems face significant challenges in real-time fault detection and resolution. Intelligent agents embedded within network devices represent a paradigm shift toward distributed, autonomous network operations. This draft addresses the need for standardized communication methodologies that enable these agents to collaboratively identify, diagnose, and recover network issues across diverse environments.

The contents of this document are as follows:

First, this document introduces three use cases to illustrate communication workflows between network device agents during troubleshooting. Second, this document analyzes existing transport protocols for these interactions, highlighting the strengths and limitations for agent scenarios. Finally, this document establishes fundamental requirements for implementing effective agent-to-agent troubleshooting systems. By analyzing those interoperable communication modes, this draft aims to facilitate the development of self-healing networks capable of maintaining service levels despite increasing operational complexity.

The use cases and requirements outlined herein are designed to be applicable across various network domains, including data center networks, campus networks, and IoT edge networks.

2. Conventions and Definitions

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.

Troubleshooting Agent: An agent that runs on network devices to identify, diagnose, and recover from network failures.

3. Use Cases

3.1. Use Case 1: Data Center Network

In a large-scale data center network, multiple troubleshooting agents on network devices, such as switches and routers, need to collaboratively identify and diagnose a transient latency issue affecting application performance. The troubleshooting workflow begins when an application performance monitoring agent detects elevated response times and assign a diagnosis task to troubleshooting agents on network devices. The application performance monitoring agent also can reports this issue to the agent on the network controller, which then notifies troubleshooting agents on network devices to root the cause of this performance issue. The ways by which troubleshooting agents on network devices receive tasks is beyond the scope of this draft.

For this use case, a high-performance, low-cost communication protocol is required. In existing works, gRPC provides significant advantages for this scenario. This part takes gRPC as an example.

When a troubleshooting agent on the network device receives a task to identify, diagnose, and recover from network failures, the communication flow may same as this figure.

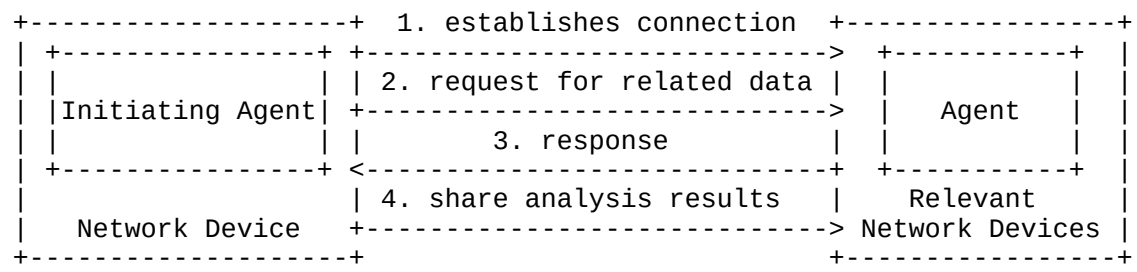

Figure: Data Center Networks

The communication flow includes these steps as follows. This document provides message examples for each step.

Step 1, the initiating agent establishes connections with relevant network device agents. This step may include some security-related steps and description about failure.

```
{
        "Sender": "Agent-I",
        "Failure":
        {
                "Location": "Host 1",
                "Type": "Packet loss rate greater than threshold",
                "Description": "...",
        },
        "Solution":
        {
        },
        "Analysis":
        {
        },
        ...
}
```

Step 2, through bidirectional connection, the initiator requests real-time telemetry data including interface statistics, queue depths, and latency measurements.

The request message of initiator could be as following.

```
   {
           "Sender": "Agent-I",
           "Failure":
           {
                   "Location": "Host 1",
                   "Type": "Packet loss rate greater than threshold",
                   "Description": "...",
           },
           "Solution":
           {
                   "RelatedNetDevice1":
                   {
                           "Type": "Request",
                           "Resource": "Data",
                           "Description": "Traffic patterns of the device’s ingress and e
gress over a certain period of time.",
                           "TransMehtods": ["gRPC", "QUIC"]
                   },
                   "RelatedHost2":
                   {
                           "Type": "Request",
                           "Resource": "Method",
                           "Description": "The device needs to send colored packets to co
llect path data."
                   },
                   ....
           },
           "Analysis":
           {
           },
           ...
   }
```


   Step 3, network agents respond with telemetry data in real-time. The
   related network device would send this message as response.

```
   {
           "Sender": "Agent-D",
           "Response":
           {
                   "Resource": "Data",
                   "Data":
                   {
                           "Description": "Traffic patterns of this device’s ingress and
egress over a certain period of time.",
                           "TransMethods": "gRPC",
                           ...
                   }
                   ....
           },
           ...
   }
```

Step 4, optionally, the initiating agent share analysis results through the same channels, enabling collaborative root cause identification.

Step 5, once the root casue of the failure is identified, agents negotiate and implement traffic engineering adjustments.

3.2. Use Case 2: Campus Network

A network segmentation issue in an enterprise campus requires verification of consistent policy application across multiple security domains. Agents residing in firewalls, switches, and wireless controllers must collaboratively audit their configurations against intended policies to identify discrepancies causing the segmentation failure.

For this use case, a configuration-oriented troubleshooting scenario, HTTP-based RESTCONF offers several benefits. This part takes RESTCONF as an example. The RESTful architecture provides familiar, standardized operations (GET, PATCH, DELETE) for configuration manipulation. YANG data modeling ensures semantic consistency across multi-vendor environments, crucial for accurate policy verification. HTTP/2’s header compression and request multiplexing improve efficiency when interacting with numerous agents simultaneously. The protocol’s stateless nature simplifies error recovery, while standardized status codes and error responses enable predictable failure handling. Rich authentication mechanisms integrate seamlessly with existing enterprise security infrastructures. However, RESTCONF lacks streaming capabilities for real-time telemetry exchange.

In this use case, the agent who is informed to complete this task is named coordinator agent.

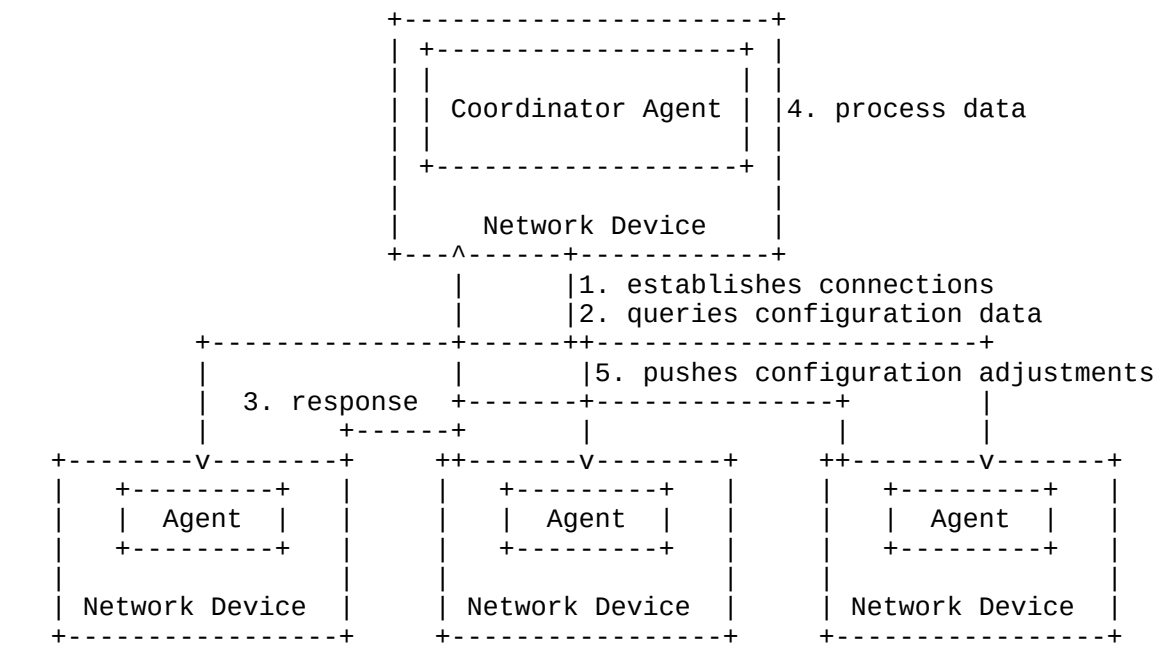

Figure: Campus Networks

The communication flow includes these steps:

1. A coordinator agent establishes connections with relevant network agents.

2. The coordinator queries configuration data from multiple device agents using standardized YANG data models.

3. Each agent responds with structured configuration data representing its current operational state.

4. The coordinator analyzes the collective configuration data, identifies inconsistencies in access control lists and routing policies, and generates remediation instructions.

5. Using RESTCONF PATCH operations or other network management operations, the coordinator pushes configuration adjustments to specific agents.

6. Agents respond with structured error messages if operations fail, enabling precise fault localization.

3.3. Use Case 3: IoT Edge Network

In an IoT edge network, multiple constrained devices experience intermittent connectivity issues. Lightweight agents on these devices must efficiently share fault information and coordinate recovery actions while conserving bandwidth and battery resources.

In IoT network, MQTT protocol is used widely. This part takes MQTT as an example. MQTT's publish-subscribe model offers distinct advantages for distributed troubleshooting scenarios. The decoupled communication pattern allows agents to exchange information without direct connections, reducing coordination overhead. Configurable QoS levels enable reliability matching for different message. For example, types— QoS 0 for non-critical telemetry, QoS 1 for important fault notifications, and QoS 2 for critical configuration changes. The minimal protocol overhead conserves bandwidth and battery life on constrained devices. Last Will and Testament features ensure other agents are notified when a device becomes unreachable, enabling rapid detection of network partitions. The topic-based routing simplifies message filtering and delivery to interested parties only.

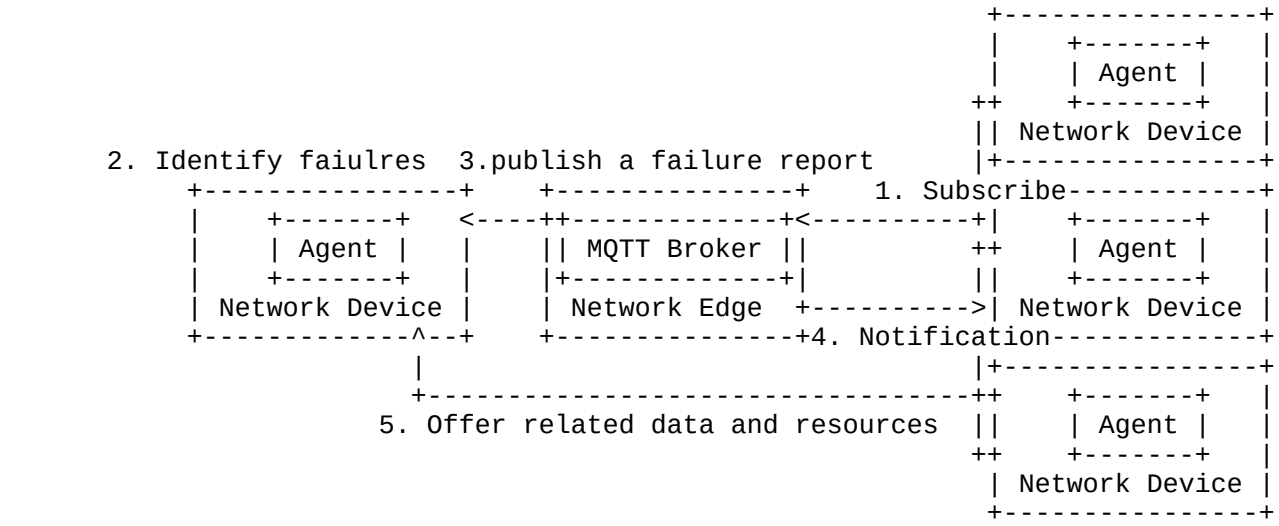

Figure: IoT Edge Networks

The communication flow includes these steps:

1. Agents subscribe to relevant fault notification topics on an MQTT broker deployed at the network edge.

2. The agent on the network device which happened a network failure identifies the network failure.

3. Agent publishes a structured failure report to appropriate topics.

4. Subscribed agents receive the notification and contribute additional context from their perspectives.

5. The subscribed agents may offer some data and resources to diagnose or recover FROM this failure.

6. The agent on the network device that caused this failure recovers the failure or reports it.

4. Requirements

According to those use cases, this draft concludes requirements of communication protocol for network troubleshooting interactions among agents on network devices.

4.1. Data Transport Requirement

4.1.1. Data Format

The interaction between Agents should use human-readable language, e.g., natural language. However, in terms of communication performance, messages delivered by agents should be encapsulated in structured format. The message sent by agent would be as follows.

```
{
        "Sender": "Agent",
        "Failure":
        {
                "Location": "..",
                "Type": "...",
                "Description": "...",
        },
        "Solution":
        {
        },
        "Analysis":
        {
        },
        ...
}
```

4.1.2. Streaming Capabilities

Troubleshooting agents MUST support bidirectional streaming for real-time telemetry exchange and collaborative analysis. Streaming implementations SHOULD include flow control mechanisms to prevent resource exhaustion and MUST maintain message ordering within streams. Agents SHOULD implement priority handling for critical troubleshooting messages within streams to ensure timely delivery of urgent notifications.

4.1.3. Transaction Integrity

For configuration modifications during troubleshooting, agents MUST implement transactional semantics to maintain network consistency. Multi-agent transactions SHOULD support two-phase commit protocols or equivalent distributed consensus mechanisms. All configuration changes MUST be idempotent to allow safe retransmission in case of delivery uncertainties.

4.2. Protocol Implementation Requirements

4.2.1. Mandatory Transport Security

All inter-agent communications MUST employ transport-layer security (TLS 1.2 or higher) with mutual authentication. Certificate-based authentication is PREFERRED over pre-shared keys for scalable deployment. Agents MUST implement certificate revocation checking and SHOULD support forward secrecy cipher suites.

4.2.2. Standardized Error Handling

Agents MUST implement consistent error reporting mechanisms across all communication protocols. Error responses MUST include machine-readable error codes, human-readable descriptions, and suggested remediation actions. Protocol-specific error mappings SHOULD be defined to translate underlying transport errors to application-level troubleshooting semantics.

4.2.3. Message Prioritization and Preemption

Troubleshooting systems MUST implement message prioritization to ensure critical fault notifications receive appropriate network resources. Agents SHOULD support preemption of lower-priority communications when high-priority troubleshooting sessions require immediate attention. Quality of Service differentiation SHOULD be implemented at both transport and application layers.

4.3. Operational Requirements

4.3.1. Interoperability and Versioning

Agents MUST implement protocol version negotiation to maintain backward compatibility during upgrades. Data schema evolution SHOULD follow compatibility rules that prevent communication breakdowns. Agents SHOULD support graceful degradation of functionality when communicating with older implementations.

4.3.2. Resource Management

Agent implementations MUST include configurable resource limits to prevent exhaustion during mass troubleshooting events. Memory, bandwidth, and processing quotas SHOULD be enforced per communication session. Agents MUST implement circuit breaker patterns to isolate misbehaving peers and maintain overall system stability.

4.3.3. Observability and Audit

All troubleshooting communications MUST be logged with sufficient detail to reconstruct decision processes. Log entries SHOULD include message timestamps, participant identities, and semantic content summaries. Audit trails MUST be protected against tampering and available for post-incident analysis.

5. Security Considerations

TBD

6. IANA Considerations

This document has no IANA actions.

7. Conclusion

TBD

8. Normative References

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

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

Acknowledgments

TODO acknowledge.

Authors' Addresses

Ruyi Zhang
Huawei
Email: zhangruyi8@huawei.com

Jianwei Mao
Huawei
Email: maojianwei@huawei.com

Bing Liu
Huawei
Email: leo.liubing@huawei.com

Nan Geng
Huawei
Email: gengnan@huawei.com

Xiaotong Shang
Huawei
Email: shangxiaotong@huawei.com

Qiangzhou Gao
Huawei
Email: gaoqiangzhou@huawei.com

Zhenbin Li
Huawei
Email: robinli314@163.com