CASA: Capability-Aware Agent Discovery and Binding for Network Operating Systems

Riya Ponraj
University of Oregon
ripo@uoregon.edu

River Bartz
University of Oregon
rba@uoregon.edu

Ramakrishnan Durairajan
University of Oregon & Link Oregon
ramd@uoregon.edu

Abstract

AI-driven network operations increasingly use agents for tasks such as BGP analysis, telemetry analysis, topology inference, and incident diagnosis. However, today's network operations are often tied to the specific agents performing these tasks; so each operation must handle agent failures or changing conditions itself. We present CASA, a discovery and replacement layer for agentic network operating systems. With CASA, a network operation specifies what task it needs performed and any operational requirements, rather than naming a particular agent. CASA finds a suitable agent, selects one that currently satisfies those requirements, and replaces it if it becomes unavailable or no longer meets them. Across registries containing 10 to 1,000 agents, median discovery latency increases from 0.75 ms to 77.37 ms, with a 95th-percentile latency of 82.77 ms at 1,000 agents. At that scale, median agent selection and task assignment take 2.33 ms and 0.03 ms, respectively.

1 Introduction

AI agents increasingly support networking tasks including BGP analysis, telemetry analysis, topology inference, and incident diagnosis Wang et al. [2025], Hamadanian et al. [2023], Wang et al. [2024], Wu et al. [2024], Zaid et al. [2026]. We use the term *network operation* to mean the overall objective, such as investigating a sudden traffic shift, and *task* to mean one piece of work needed to accomplish that objective, such as analyzing BGP routes or telemetry data. Existing agentic network-management systems commonly assign such tasks to explicitly instantiated or specialized agents Wang et al. [2025], Zaid et al. [2026]. As a result, deciding what to do when an agent fails, becomes slow, or is no longer suitable can become the responsibility of each network operation rather than a function provided by the network operating system (NOS).

This limitation is important during network incidents, where the agents available to perform a task may change while an investigation is in progress. Consider an operator investigating a sudden traffic shift. The investigation may first use a BGP-analysis agent to determine whether routing changed and then a telemetry-analysis agent to identify affected links. If the BGP-analysis agent becomes unavailable or becomes too slow, the NOS should be able to find another agent that can perform the same task and allow the investigation to continue. This replacement is not as simple as choosing another copy of the same service: agents that perform the same task may use different implementations and data sources, cover different parts of the network, produce results of different quality, or have different permissions and current performance. The NOS must therefore find an agent that can perform the requested task *and* satisfies the requirements of the current operation.

Existing agent directories and protocols already allow agents to describe what they can do and allow other systems to find them Foundation for Intelligent Physical Agents [2002], A2A Protocol Working Group [2026], AGNTCY [2026a,b]. However, these mechanisms do not by themselves decide which agent should be used for a network task based on current network requirements, nor do they automatically replace that agent when those requirements are no longer met. The key challenge

Preprint.

is therefore to separate a network operation from the identity of the agent performing each task. An operation should state what it needs done and any requirements on how the task is performed; the NOS should determine which available agent to use and change that choice when conditions change. This follows the broader principle of moving functionality shared by many applications into a common network-control platform Koponen et al. [2010].

We present CASA, a capability-aware agent discovery and replacement layer for agentic NOS. A network operation specifies the task it needs and requirements such as latency, data freshness, network scope, or authorization. CASA finds agents that can perform the task, chooses one that currently satisfies these requirements, and keeps track of that choice during execution. If the selected agent becomes unavailable or no longer satisfies the requirements, CASA selects a replacement without requiring the network operation to implement its own discovery and recovery logic.

We implement a prototype of CASA and evaluate it with up to 1,000 agents. In an experiment where the latency of a BGP-analysis agent is deliberately increased beyond the operation's requirement, CASA switches the task to another suitable agent within 80 ms without changing the network operation. At 1,000 registered agents, median discovery latency is 77.37 ms, with a p95 of 82.77 ms, while median agent selection and task assignment require 2.33 ms and 0.03 ms, respectively. These results show that CASA can dynamically choose and replace agents at runtime and identify agent discovery as the dominant scaling cost in the current prototype.

This paper makes three contributions.

- First, we introduce a NOS abstraction that decouples network operations from agent identities: operations specify the task to be performed and its operational requirements, while the NOS resolves that request to an appropriate agent.
- Second, we design CASA, which combines capability discovery with policy-aware selection and maintains the resulting task-to-agent binding as a NOS-level control loop, replacing the selected agent when it no longer satisfies the operation's requirements.
- Third, we implement a CASA prototype and evaluate its behavior under agent degradation and increasing registry size, demonstrating runtime replacement without changing the network operation and quantifying the overhead of discovery, selection, and binding.

2 Motivation and Related Work

Agent discovery is needed when a network operation specifies what needs to be done without specifying which agent should do it. Consider a network operator investigating: *Why is traffic between A and B experiencing elevated latency, and which networks are affected?* Answering this question may require several tasks, such as examining network topology, BGP routes, telemetry, and active measurements. CASA allows the operation to describe these tasks rather than name particular agents. The NOS then finds suitable agents at runtime. If an agent becomes unavailable, too slow, stale, or no longer provides sufficient coverage, the NOS can replace it without requiring the network operation to implement its own discovery and recovery logic.

The Agent Discovery Problem. We distinguish three elements of the discovery problem: (1) *task requirement*, which describes what needs to be done and any requirements that must be satisfied; (2) *agent capability*, which describes what an agent can do and how it can be invoked; and (3) *agent state*, which captures properties that can change over time, such as availability, latency, and data freshness.

The discovery problem is therefore to map a task requirement to an appropriate agent while considering both what the agent can do (i.e., agent capability) and whether it is currently suitable (i.e., agent state). For example, several agents may be able to analyze BGP data, but they may differ in latency, availability, measurement coverage, or resource requirements. Discovery identifies the agents that can perform the task, while selection determines which of them is appropriate under the current conditions. Concretely, we ask: *given a task requirement, a changing population of heterogeneous agents, and operational requirements, how can a NOS find and select an appropriate agent without requiring the network operation to know that agent's identity?*

This leads to three requirements for CASA. First, agents need a common way to describe what they can do, independent of how they are implemented. Second, the system must separate identifying agents that can perform a task from deciding which of those agents is currently suitable. Third, the

system must be able to replace a selected agent when it becomes unavailable or no longer satisfies the operation's requirements.

Related Work. Recent work has explored LLMs and multi-agent systems for network management, configuration, and incident diagnosis. Confucius Wang et al. [2025] coordinates multiple LLM agents to translate operator intent into network-management actions. CASA is complementary: such a system could use CASA beneath its workflow to resolve a required task to an eligible agent and replace that agent as operational conditions change. Other work (e.g., Hamadanian et al. [2023], Wang et al. [2024], Wu et al. [2024], Zaid et al. [2026]) applies LLMs to network configuration, incident analysis, and networking-specific prediction and decision tasks. These systems primarily address how AI agents perform or coordinate network-management tasks. CASA addresses a different problem: how a network operation can ask for a task to be performed without depending on the identity of the agent that performs it.

Agent discovery itself is not new. FIPA introduced directory facilitators for advertising and discovering agent services Foundation for Intelligent Physical Agents [2002], while A2A and AGNTCY allow agents to describe their capabilities and be discovered by other systems A2A Protocol Working Group [2026], AGNTCY [2026a,b]. CASA does not propose a new directory protocol or search mechanism; an indexed registry or existing discovery service could provide the candidate lookup. Instead, CASA places a control loop around discovery inside the NOS: it combines capability information with live operational state and policy, selects an eligible agent, maintains that choice during execution, and reconsiders it when the requirements of the network operation are no longer satisfied.

CASA also relates to conventional service discovery and dynamic service selection. DNS-SD Cheshire and Krochmal [2013] and service-mesh mechanisms Chandramouli and Butcher [2020] locate and route requests among instances of known services, while prior work on Web services studied QoS-aware selection and runtime rebinding Maximilien and Singh [2004], Zeng et al. [2004], Canfora et al. [2008]. CASA builds on these ideas rather than claiming that selection or rebinding is itself new. The distinction is that agents advertising the same task need not implement interchangeable semantics: they may use different models, tools, data sources, permissions, and reasoning procedures and may therefore produce different outputs for the same request. CASA makes these differences visible to the NOS through capability, state, trust, provenance, and authorization attributes. Determining whether two agents are semantically equivalent, or whether a replacement preserves result quality, remains an open problem.

3 CASA: Capability-Aware Agent Discovery and Dynamic Binding

CASA decouples network operations from the identity, implementation, and location of the agents executing their tasks (see Figure 1). An operation specifies required capabilities and operational constraints; CASA discovers and selects suitable agents and maintains the resulting bindings during execution, rediscovering an alternative when a binding becomes invalid.

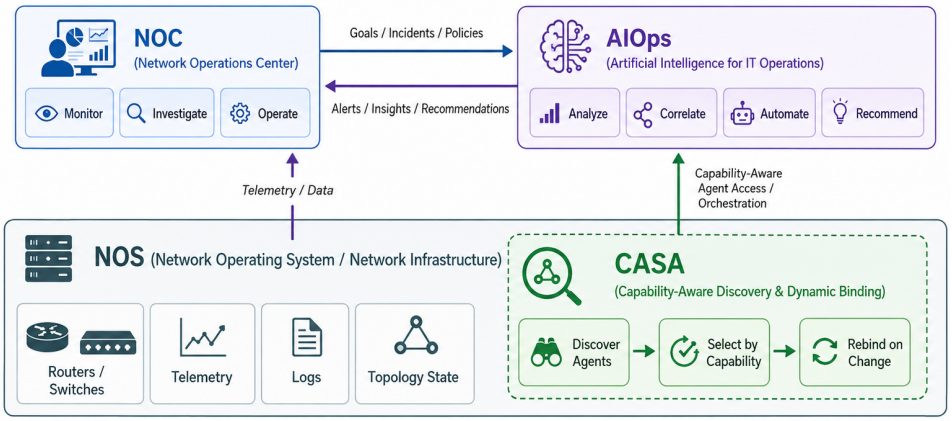

Figure 1: Overview of CASA and its position in the larger AIOps/NOC ecosystem.

3.1 System Model and Components

We model the agent environment as a dynamic collection of agents $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$. Each agent a_i has a profile $P_i = (C_i, I_i, D_i, Q_i, T_i, S_i(t))$, where C_i denotes its capabilities, I_i its interfaces, D_i its data and tool dependencies, Q_i its quality attributes, T_i its trust, provenance, and authorization attributes, and $S_i(t)$ its dynamic state, including availability, load, latency, and data freshness. We represent a capability as $c = (g, I, D)$, where g specifies the operational goal, I the task interface and input/output types, and D the required network data. For example, a task for identifying networks affected by an incident may require $c_{\text{incident}} = (\textit{identify_affected_networks}, \textit{prefixes/ASes}, \textit{telemetry+BGP})$. Multiple agents may provide the same capability while differing in their implementations, data sources, and runtime properties. The distinction between capability and runtime state is important for network operations. Two agents may both provide BGP analysis while differing in data freshness, response latency, measurement coverage, or authorization. An agent that satisfies a capability requirement at binding time may also become unsuitable later as its runtime state changes.

Capability Discovery. A network operation specifies a task requirement $r = (G, X, \Omega)$, where G specifies the required goal, X describes the available context and inputs, and Ω specifies operational requirements. We distinguish hard constraints Ω_h, which must be satisfied for an agent to execute the task, from soft preferences Ω_s, which are used to rank otherwise eligible agents. Hard constraints may include authorization, network scope, required data access, maximum latency, and minimum data freshness, while soft preferences may include cost, load, or relative quality.

CASA first evaluates whether an advertised agent capability is compatible with the task requirement using $M(r, a_i) \in \{0, 1\}$. The discovery layer constructs the capability-compatible candidate set $\mathcal{A}_r^{\text{cap}} = \{a_i \in \mathcal{A} \mid M(r, a_i) = 1\}$. For example, an incident investigating a sudden traffic shift may require topology analysis, BGP analysis, telemetry analysis, and active measurement. CASA independently discovers agents capable of executing each task without requiring the incident-management operation to know which agents are available.

Capability Matching. CASA intentionally separates the binding abstraction from the mechanism used to determine whether an advertised capability satisfies a task requirement. The matcher $M(r, a_i)$ may use a shared capability vocabulary, structured schemas, semantic retrieval, or an LLM-based matcher. Our current prototype assumes normalized capability descriptions and therefore does not evaluate semantic matching accuracy. Handling ambiguous or overlapping capability descriptions remains an important direction for future work.

Constraint-Aware Agent Selection. Discovery identifies agents that provide the required capability. CASA next determines which of those agents are currently eligible for the task. We define $F(r, a_i, t) \in \{0, 1\}$ as a feasibility predicate that evaluates the hard constraints Ω_h against the agent profile and current state. The eligible candidate set is therefore $\mathcal{A}_r(t) = \{a_i \in \mathcal{A}_r^{\text{cap}} \mid F(r, a_i, t) = 1\}$. CASA then selects the most suitable eligible agent: $a^* = \arg\max_{a_i \in \mathcal{A}_r(t)} U(a_i \mid r, t)$, where $U(a_i \mid r, t) = w_q Q(a_i) + w_s S(a_i, r, t) + w_t T(a_i, r) - w_k K(a_i, r)$. Here, $Q(a_i)$ captures agent quality, $S(a_i, r, t)$ captures runtime suitability, $T(a_i, r)$ captures trust and provenance among eligible agents, and $K(a_i, r)$ captures execution cost. Authorization and other mandatory constraints are enforced by F rather than traded against these ranking criteria. Separating discovery from selection distinguishes capability compatibility from current eligibility. Several agents may provide BGP analysis, for example, while only those satisfying freshness, latency, authorization, and capacity constraints are currently eligible.

Runtime (Re)Binding. After selection, CASA establishes a runtime binding between the task requirement and the selected agent: $b = (r, a_i, \epsilon, \theta)$, ϵ contains invocation parameters, and θ contains runtime monitoring information. The binding isolates the network operation from the identity of the agent executing its task. An incident-management operation can therefore request BGP analysis without directly addressing a particular BGP-analysis agent.

CASA monitors active bindings using the agent's dynamic state $S_i(t)$. A binding remains valid while the selected agent continues to satisfy the task's capability and hard constraints: $V(b, t) = \begin{cases} 1, & \text{if } M(r, a_i) = 1 \text{ and } F(r, a_i, t) = 1, \\ 0, & \text{otherwise.} \end{cases}$ When $V(b, t) = 0$, CASA invalidates the binding and triggers rediscovery. A replacement agent is then selected from the updated eligible candidate set: $a_{t+1}^* = \arg\max_{a_i \in \mathcal{A}_r(t+1)} U(a_i \mid r, t+1)$. For example, if a BGP-analysis agent exceeds the

latency constraint during an ongoing routing incident, CASA can rediscover alternative BGP-analysis agents, filter those that do not satisfy the task's hard constraints, and bind the task to the most suitable remaining agent. The incident operation continues to request BGP analysis without implementing agent discovery or replacement itself.

Multi-Capability Operations. Incidents commonly require multiple complementary tasks. Consider an incident involving a sudden traffic shift. The investigation may require *topology* $\rightarrow$ *BGP* $\rightarrow$ *telemetry* $\rightarrow$ *measurement*. A topology-analysis agent can identify affected networks and paths, a BGP-analysis agent can identify routing changes, a telemetry-analysis agent can identify traffic shifts or anomalous links, and a measurement agent can validate reachability and path performance.

CASA discovers and selects an agent for each task, producing a composition such as $W = (a_{\mathrm{topo}}, a_{\mathrm{BGP}}, a_{\mathrm{telemetry}}, a_{\mathrm{measure}})$. The operation therefore depends on the required capabilities rather than on the identities of the agents that provide them. If the BGP-analysis agent becomes unsuitable, CASA can replace that component without requiring the operation to reconstruct the remaining workflow. Discovery therefore becomes a foundational NOS function for managing agents as dynamic computational resources. As the number and diversity of network agents increase, encoding agent identities and replacement logic directly into operations becomes increasingly brittle. CASA instead allows those decisions to be handled by the NOS runtime.

Operational Safety. CASA also distinguishes capability discovery from authorization. This is important in a NOS because some capabilities are read-only while others can modify network state. Read-only capabilities include BGP analysis, topology inference, telemetry analysis, and active measurement, while state-changing capabilities include route modification, traffic engineering, ACL changes, and interface configuration. Authorization and network scope are therefore expressed as hard constraints in Ω_h and enforced through $F(r, a_i, t)$. The agent attributes T_i provide the trust, provenance, and authorization information needed to evaluate these constraints; trust and provenance may additionally inform selection among agents that are already eligible for the task.

Capability advertisements should not themselves establish trust. In a deployment, attributes such as identity, authorization, provenance, and network scope in T_i must be established by the administrative domain through registration policy, signed metadata, attestation, or another trusted mechanism. CASA can use these attributes during selection, but verification of an agent's claimed competence or the semantic correctness of its outputs is a separate problem.

3.2 Putting CASA into Practice

CASA can be implemented as a control-plane service within the NOS rather than as part of individual network agents. Each agent registers a capability descriptor containing its capabilities C_i, invocation interface I_i, data dependencies D_i, and trust and authorization attributes T_i. The NOS maintains the dynamic portion $S_i(t)$ using agent heartbeats, health checks, and runtime observations such as latency and data freshness. These descriptors and states form a registry that can be maintained alongside other NOS state, such as topology, telemetry, and policy.

Network operations interact with CASA through a capability interface rather than an agent endpoint. For example, an operation submits a requirement $r = (G, X, \Omega)$ and receives an opaque binding handle. CASA resolves the requirement against the registry, applies the hard constraints, selects an eligible agent, and invokes that agent through its registered interface. A binding monitor continuously evaluates the constraints associated with active handles; if one becomes invalid, CASA reruns discovery and atomically updates the handle to a replacement agent. The network operation therefore continues to invoke the same capability without learning that the underlying agent has changed. For state-changing capabilities, invocation additionally passes through the NOS authorization and policy layer before an agent's output can modify network state.

Rebinding does not imply migration of an in-progress agent execution. CASA changes the agent used for subsequent invocations through the binding handle; an invocation already executing may complete or fail according to the semantics of that agent. Stateful handoff would require task-specific transfer of context or intermediate state and is outside the current abstraction. Similarly, production deployments should use hysteresis, dwell times, or backoff before invalidating a binding to avoid repeated switching when a runtime metric oscillates near a threshold.

Finally, CASA is intended primarily for an administrative domain in which a NOS can enforce common policy and observe relevant runtime state, such as a campus, enterprise, cloud, or provider network. It is not intended to replace Internet-scale agent search. Discovery across administrative domains would require additional mechanisms for federated identity, trust, policy, and capability verification; we leave such federation to future work.

4 Preliminary Evaluation

We evaluate CASA as a capability-aware discovery and dynamic binding layer for network operations. The evaluation addresses three questions: (1) can CASA resolve network-operation requirements to appropriate agents, (2) can CASA dynamically replace an agent when it violates a runtime constraint without changing the operation, and (3) what overhead does CASA introduce as the agent population grows? Note that our prototype evaluates read-only network operations and does not automatically authorize agents to modify network state. In a production NOS, state-changing operations would additionally require explicit authorization and other safety mechanisms to mitigate potentially disruptive actions (§ 5).

Network Incident Capability Discovery. We first demonstrate CASA's capability-level abstraction using a representative network incident: investigating a sudden traffic shift affecting a target network. The operation specifies the tasks required for the investigation, including topology analysis, BGP analysis, telemetry analysis, and active measurement, but does not identify the agents that should execute them. CASA discovers agents that satisfy each task's capability requirement and selects an eligible agent under the task's operational constraints. For example, the resulting bindings may map topology analysis to `TopologyAnalysisAgent-2`, BGP analysis to `BGPAnalysisAgent-1`, telemetry analysis to `TelemetryAnalysisAgent-3`, and active measurement to `MeasurementAgent-1`. The operation therefore depends only on the required capabilities, while CASA determines the agents that execute them at runtime.

Dynamic Rebinding. We next evaluate CASA's rebinding when a bound agent becomes unsuitable during execution. We use BGP analysis as the representative task and initially bind it to `BGPAnalysisAgent-1`. During execution, the agent's latency increases beyond the task's 300 ms threshold.

CASA detects the constraint violation, invalidates the binding, rediscovers agents capable of BGP analysis, evaluates the currently eligible candidates, and selects `BGPAnalysisAgent-2`. The operation continues to request BGP analysis without knowing the identity of, or explicitly invoking, the replacement agent.

Figure 2 shows the measured sequence of constraint violation, detection, rediscovery, and rebinding, together with the active binding over time. The latency threshold is crossed at approximately 5.00 s, CASA detects the violation at 5.02 s, begins rediscovery at 5.05 s, and establishes the replacement binding at 5.08 s. The transition from the observed constraint violation to the new binding therefore takes approximately 80 ms. These timestamps are measured from the implementation rather than assumed a priori. This experiment demonstrates CASA's central dynamic-binding property: when a bound agent no longer satisfies a task's runtime constraints, CASA can replace it without exposing agent identity or replacement logic to the network operation.

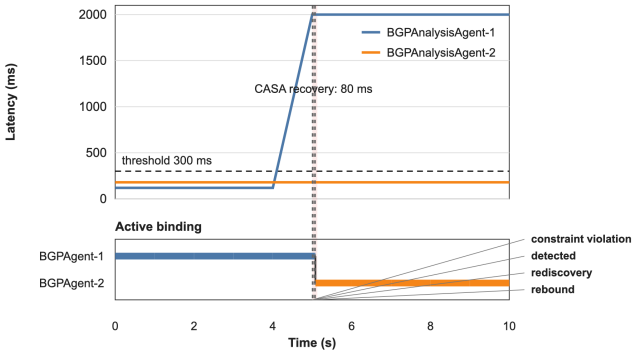

Figure 2: Rebinding during a latency-constraint violation with corresponding change in active binding (lower panel).

Runtime Overhead. Finally, we evaluate CASA's overhead as the number of registered network-operation agents increases. We construct registries containing 10, 50, 100, 500, and 1,000 heterogeneous agents providing capabilities such as BGP analysis, topology analysis, telemetry analysis,

and active measurement. For each registry size, we execute the same capability request and measure four stages: discovery, selection, binding, and rediscovery. Discovery measures the time to identify capability-compatible agents; selection measures the time to evaluate the discovered candidates and choose an eligible agent; binding measures the time to establish the selected agent as the task's active binding; and rediscovery measures the time to repeat discovery after an active binding becomes invalid. We report the median and 95th-percentile latency for each stage.

Table 1 shows the results. Median discovery latency increases from 0.7462 ms for 10 agents to 77.3712 ms for 1,000 agents, with a corresponding p95 increase from 0.8342 ms to 82.7698 ms. Rediscovery follows a similar trend: its median increases from 0.6737 ms to 77.6519 ms, while its p95 reaches 81.4868 ms at 1,000 agents. Median selection latency increases from 0.0249 ms to 2.3271 ms, with a p95 of 2.5880 ms at 1,000 agents. Binding remains comparatively small, with a median of 0.0301 ms and a p95 of 0.0373 ms at 1,000 agents.

Agents	Disc. med.	Disc. p95	Sel. med.	Sel. p95	Bind med.	Bind p95	Redis. med.	Redis. p95
10	0.7462	0.8342	0.0249	0.0517	0.0063	0.0180	0.6737	0.8675
50	3.8536	4.3353	0.1169	0.1517	0.0110	0.0235	3.7796	4.3879
100	7.6639	7.8616	0.2233	0.2595	0.0127	0.0175	7.6130	7.9416
500	38.6025	48.4296	1.1103	1.3514	0.0265	0.0362	38.6148	45.3436
1000	77.3712	82.7698	2.3271	2.5880	0.0301	0.0373	77.6519	81.4868

Table 1: CASA runtime overhead (in ms) as the number of registered agents increases.

Discovery and rediscovery therefore dominate the runtime cost of the current prototype. Their latency grows approximately linearly over the evaluated range, consistent with the prototype's registry search mechanism, which examines the available agent population during discovery. Selection also increases with registry size but remains below 2.6 ms at the 95th percentile for 1,000 agents, while binding remains below 0.04 ms. These results identify (re)discovery as the primary performance bottleneck and motivate indexed or hierarchical discovery mechanisms. Within the evaluated range, however, constraint-aware selection and binding remain lightweight relative to discovering candidate agents.

5 Discussion and Future Outlook

Our evaluation is intended to validate the CASA abstraction and quantify the overhead of the current prototype; it does not compare semantic matching accuracy, diagnosis quality, or lookup performance against alternative discovery mechanisms. The current experiments assume normalized capability descriptions and do not evaluate semantic matching, agent-result quality, consistency across replacements, or stateful handoff. Similarly, the observed 80 ms replacement time is a measurement of the current prototype and monitoring configuration, not a general recovery-time guarantee. Several systems challenges also remain. Multi-capability operations require reasoning about compatibility and constraints across an entire workflow, while distributed deployments introduce authorization, provenance, trust, and data-locality concerns. Larger registries will require indexed, hierarchical, or decentralized discovery.

Rebinding AI agents is riskier than conventional service failover because agents advertising the same capability may behave differently. This risk is lower for read-only tasks but substantial for state-changing actions, where a poor replacement could worsen an incident. This motivates a *verification boundary* between an agent's recommendation and its execution. We therefore envision extending CASA's hard constraints in Ω_h and the trust attributes T_i to encode capability action classes, such as read-only, advisory, and state-changing. State-changing capabilities could require higher trust and provenance thresholds for automatic rebinding, or human approval before a replacement binding is allowed to execute.

Acknowledgements

We thank the anonymous reviewers for their constructive feedback. This work is supported by the National Science Foundation (CNS-2145813). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF.

References

A2A Protocol Working Group. Agent2Agent (A2A) Protocol Specification. Version 1.0.0, 2026. URL https://a2a-protocol.org/v1.0.0/.

AGNTCY. AGNTCY Directory: Distributed Announce and Discovery of Multi-Agentic Systems. Software and technical documentation, 2026a. URL https://github.com/agntcy/dir.

AGNTCY. Open Agentic Schema Framework (OASF). Technical specification and software, 2026b. URL https://github.com/agntcy/oasf.

Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. A Framework for QoS-Aware Binding and Re-Binding of Composite Web Services. *Journal of Systems and Software*, 81(10):1754–1769, 2008. doi: 10.1016/j.jss.2007.12.792.

Ramaswamy Chandramouli and Zack Butcher. Building Secure Microservices-Based Applications Using Service-Mesh Architecture. Technical Report NIST SP 800-204A, National Institute of Standards and Technology, May 2020.

Stuart Cheshire and Marc Krochmal. DNS-Based Service Discovery. Technical Report RFC 6763, Internet Engineering Task Force, February 2013. URL https://www.rfc-editor.org/rfc/rfc6763.

Foundation for Intelligent Physical Agents. FIPA Agent Management Specification. Technical Report SC00023J, Foundation for Intelligent Physical Agents, December 2002. URL https://www.fipa.org/specs/fipa00023/SC00023J.html.

Pouya Hamadanian, Behnaz Arzani, Sadjad Fouladi, Siva Kesava Reddy Kakarla, Rodrigo Fonseca, Denizcan Billor, Ahmad Cheema, Edet Nkposong, and Ranveer Chandra. A Holistic View of AI-Driven Network Incident Management. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pages 180–188. Association for Computing Machinery, 2023. doi: 10.1145/3626111.3628176.

Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A Distributed Control Platform for Large-Scale Production Networks. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*. USENIX Association, 2010.

E. Michael Maximilien and Munindar P. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93, 2004. doi: 10.1109/MIC.2004.27.

Changjie Wang, Mariano Scazzariello, Alireza Farshin, Simone Ferlin, Dejan Kostić, and Marco Chiesa. NetConfEval: Can LLMs Facilitate Network Configuration? *Proceedings of CoNEXT*, 2024. doi: 10.1145/3656296.

Zhaodong Wang, Samuel Lin, Guanqing Yan, Soudeh Ghorbani, Minlan Yu, Jiawei Zhou, Nathan Hu, Lopa Baruah, Sam Peters, Srikanth Kamath, Jerry Yang, and Ying Zhang. Intent-Driven Network Management with Multi-Agent LLMs: The Confucius Framework. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pages 347–362. Association for Computing Machinery, 2025. doi: 10.1145/3718958.3750537.

Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. NetLLM: Adapting Large Language Models for Networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 661–678. Association for Computing Machinery, 2024. doi: 10.1145/3651890.3672268.

Hussein Zaid, Behnam Shariati, Pooyan Safari, Johannes Karl Fischer, and Ronald Freund. Empowering Network Management with a Multi-Agent LLM Architecture. *Journal of Optical Communications and Networking*, 18(6):568–580, 2026. doi: 10.1364/JOCN.585043.

Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004. doi: 10.1109/TSE.2004.11.