Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Novel Observing Strategies Testbed: A Lightweight, Collaborative Platform for Prototyping Earth Observing Missions

Emmanuel M. Gonzalez
Arizona State University
Hadis Banafsheh
Arizona State University
Divya Ramachandran
Arizona State University
Melissa L. Wrzesien
University of Maryland, College Park
Sujay V. Kumar
Goddard Space Flight Center
Carrie M. Vuyovich
Goddard Space Flight Center
Paul T. Grogan
Paul.Grogan@asu.edu
Arizona State University

Research Article

Keywords: Novel observing strategies, Distributed simulation, Testbed architecture, Earth observation, Cyberinfrastructure, Systems architecting

Posted Date: November 4th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11323528/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Novel Observing Strategies Testbed: A Lightweight, Collaborative Platform for Prototyping Earth Observing Missions

Emmanuel M. Gonzalez[1], Hadis Banafsheh[1], Divya Ramachandran[1], Melissa L. Wrzesien[2,3], Sujay V. Kumar[3], Carrie M. Vuyovich[3], Paul T. Grogan[1*]

[1*]School of Computing and Augmented Intelligence, Arizona State University, Tempe, 85281, AZ, USA.
[2]Earth System Science Interdisciplinary Center, University of Maryland, College Park, 20740, MD, USA.
[3]Hydrological Sciences Laboratory, NASA Goddard Space Flight Center, Greenbelt, 20771, MD, USA.

*Corresponding author(s). E-mail(s): Paul.Grogan@asu.edu;
Contributing authors: emmanuelgonzalez@asu.edu; hbanafsh@asu.edu; dramach6@asu.edu; melissa.l.wrzesien@nasa.gov; sujay.v.kumar@nasa.gov; carrie.m.vuyovich@nasa.gov;

Abstract

Distributed space mission concept development increasingly relies on collaborative, software-based testbeds, yet established distributed simulation standards and mission-operations frameworks impose overhead disproportionate to early-stage prototyping. Operating the first version of the Novel Observing Strategies Testbed exposed five systemic limitations—identity and authorization, reliable and persistent messaging, session resilience, schema governance, and coordinated scenario time control—that hindered long-duration, multi-institutional use. We evolved the testbed across two further versions to address them: Version 2 hardened the identity, messaging, session, and schema layers, and Version 3 added coordinated scenario time control through programmatic freeze, resume, and time-scale orchestration, while keeping the platform lightweight, cloud-native, and standards-aligned. Two independent Snow Observing Strategy campaigns, spanning 577 scenario days and closing the loop with an external land-surface model, validated the evolved architecture at technology readiness level 6. The

campaigns delivered zero message loss, automated session recovery, sub-second coordinated freezes, and event-driven resume that eliminated idle waiting, and they coordinated a multi-sensor observation-selection workflow toward improving snow water equivalent estimation. The testbed thereby offers an accessible, standards-aligned platform for evaluating novel observing strategies before costly development, generalizable well beyond the snow domain.

Keywords: Novel observing strategies, Distributed simulation, Testbed architecture, Earth observation, Cyberinfrastructure, Systems architecting

1 Introduction

Distributed Space Mission (DSM) concept development increasingly depends on the rapid, repeatable evaluation of coordination algorithms, heterogeneous data exchange patterns, and temporally managed execution across institutionally distributed assets. At early development stages, science and engineering teams require infrastructure that: (*i*) minimizes integration overhead; (*ii*) enforces sufficient architectural discipline to avoid semantic drift; and (*iii*) supports controlled experimentation on timing, reliability, and interoperability properties. Fully adopting comprehensive simulation or mission operations standards at this stage typically imposes disproportionate cost in tooling complexity, specialized expertise, and configuration overhead relative to the immediate analytical value obtained. The New Observing Strategies thrust of the National Aeronautics and Space Administration (NASA) Earth Science Technology Office named multi-node software testbeds among the technologies needed to coordinate orbital, airborne, and in situ observing assets (Le Moigne et al. 2019).

The Novel Observing Strategies Testbed (NOS-T) was introduced in a prior publication as Version 1 (V1): a lightweight, rapidly deployable environment that let dispersed research teams prototype DSM behaviors without committing to heavyweight simulation standards or tightly coupled mission operations frameworks (Chell et al. 2023). Sustained operational use of V1 across multi-institutional campaigns surfaced five systemic limitations that both hindered further experimentation and clarified the minimum architectural requirements for a resilient, standards-aligned research testbed.

We evolved the platform into Version 2 (V2) and Version 3 (V3) through *challenge-driven architectural refinement*, in which every design change is justified by an observed operational limitation rather than an anticipated requirement. Generalizing from V1's operational lessons and the V2/V3 design responses those lessons prompted, we define five reusable requirements (R1–R5) that form a technology-agnostic baseline for early-stage distributed interactive simulations, each specifying intent, capability, and acceptance criteria to enable reuse beyond NOS-T. Identity and authorization (R1) answers V1's static credentials and IP-based access control—which lacked role-based granularity, dynamic revocation, and auditability—with federated, token-based access providing short-lived refreshable tokens, role and attribute scoping, revocation at the identity provider, and auditability. Reliable and persistent messaging (R2) answers

V1's best-effort Message Queuing Telemetry Transport (MQTT) delivery without persistence, which risked silent data loss under transient network faults, with durable routing, delivery acknowledgments, and at-least-once delivery over the Advanced Message Queuing Protocol (AMQP). Session resilience (R3) answers V1's lack of automated reconnection and session recovery after network disruptions, with fixed-delay reconnection, token refresh, heartbeat and liveness supervision, automatic subscription restoration, and orderly shutdown signaling. Schema governance (R4) answers V1's ad hoc topic and payload conventions—which lacked versioning and validation and caused semantic drift across teams—with a versioned catalog enforcing semantic major and minor versioning, validation at publish and consume boundaries, and a documented change and deprecation policy. Coordinated scenario time control (R5) answers V1's limited scenario time management—which lacked freeze/resume and fine-grained execution control—with central freeze, resume, and scale commands together with in-application orchestration triggered by scenario-time intervals or events.

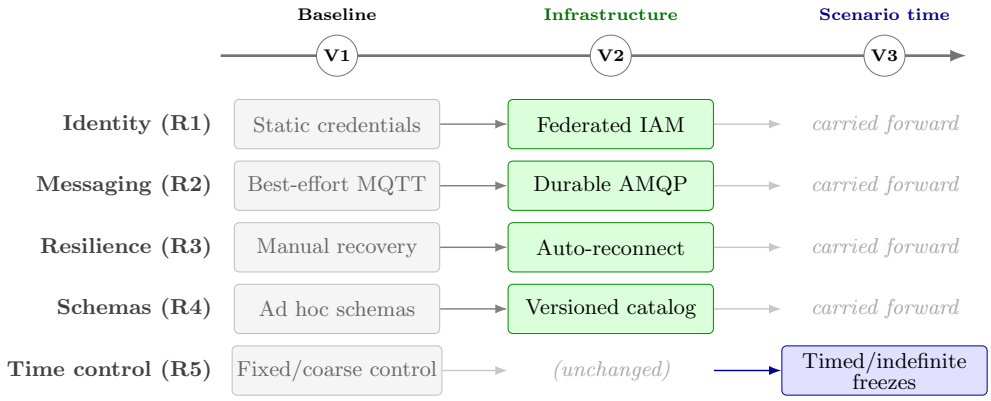


Fig. 1 Staged, challenge-driven evolution of NOS-T across three versions. Each row traces one architectural requirement from its V1 limitation to the version that resolved it. V1 established a multi-institution baseline whose operational use exposed five limitations. V2 hardened four—identity and authorization (R1), reliable messaging (R2), session resilience (R3), and schema governance (R4)—while V3 added coordinated scenario time control (R5), and the four V2 layers carry forward unchanged.

These requirements were realized in two stages: V2 hardened the four infrastructure layers—identity, messaging, resilience, and schema (R1–R4)—and V3 then extended only coordinated scenario time control (R5) (Figure 1, Supplementary Table S1). Open-source components built on open protocol standards implement those layers—Keycloak for federated identity management with OAuth2/OIDC, and RabbitMQ for AMQP message routing with durability guarantees—each independently verifiable to enable incremental validation. The stack integrates with existing research workflows, easing adoption and deployment. V3's programmatic freeze halts scenario time while external models compute, without consuming simulated campaign time—bounded by a fixed duration when the processing time is known, or held open until an external event or data product signals completion. This freeze keeps applications

inside the NOS-T framework synchronized with models running outside it on independently managed infrastructure. This extension left the four V2 layers and V1's core design principles—publish–subscribe event exchange, managed and unmanaged execution modes, and hierarchical topic namespaces—intact. Together, R1–R5 provide an evidence-based trajectory from an ad hoc initial implementation to a governed, measurable, and repeatable infrastructure, reducing integration risk while avoiding the premature complexity that early-stage iteration cannot justify.

2 Background

2.1 Distributed Systems

Systems engineering increasingly designs and integrates distributed systems—architectures composed of multiple interacting components that operate across physical, organizational, or computational boundaries. They are designed to achieve four traits: (*i*) decentralization, where components operate independently but coordinate to achieve system-level goals; (*ii*) scalability, allowing systems to grow in complexity and size without requiring complete redesign; (*iii*) resilience, achieved through redundant components that improve fault tolerance and operational continuity; and (*iv*) heterogeneity, enabling integration of diverse hardware, software, and data sources (Lamport 1978).

In space applications, these traits manifest as DSMs, which coordinate multiple spacecraft to deliver Earth observation, communications, and scientific research capabilities that single platforms cannot achieve, including enhanced spatial coverage, temporal resolution, and system redundancy (Selva et al. 2017). A mission need not be designed as a DSM from inception: spacecraft that begin as independent assets can be coordinated after the fact into an ad hoc, or virtual, mission (Le Moigne et al. 2019). Missions designed as DSMs take three common forms, each with characteristic advantages: (*i*) constellations are independent satellites in coordinated orbits that scale incrementally and provide broad spatial and temporal coverage with redundancy; (*ii*) formations maintain relative positions for joint, multi-point sensing that improves data quality; and (*iii*) fractionated systems distribute functional elements across smaller, specialized platforms that lower development and launch costs (Martinez Marrero et al. 2022). These configurations also introduce challenges—precise timing and data-exchange synchronization, communication overhead across inter-satellite and ground links, and integration of heterogeneous components (Lamport 1978; Martinez Marrero et al. 2022).

Distributed simulation platforms expose these challenges during concept development and pre-deployment evaluation of DSMs, rather than leaving them to appear once assets are in orbit. They model satellite operations and let teams exercise coordination strategies across spacecraft, ground, and analytics nodes. Such platforms must additionally provide distributed execution across multi-institution teams, scalability to large constellations and formations, and security and access control for cross-organization experiments (Selva et al. 2017). Established simulation standards meet several of these needs, but their complexity and adoption barriers limit uptake in early-stage research and development.

2.2 Simulation Standards

Distributed simulation standards emerged from 1980s defense training needs, beginning with the Defense Advanced Research Projects Agency (DARPA) Simulator Networking (SIMNET) project (Miller and Thorpe 1995). Its architecture and protocols matured into a family of IEEE standards maintained by the Simulation Interoperability Standards Organization (SISO) (IEEE 2012, 2023, 2025a). Three standards anchor this family, each addressing a distinct concern. IEEE Std 1278, the Distributed Interactive Simulation (DIS), exchanges state between independently developed simulators in real time through common, implementation-independent messages called Protocol Data Unit (PDU) (IEEE 2012). IEEE Std 1516, the High Level Architecture (HLA), composes larger federations of component simulations through a shared Runtime Infrastructure (RTI) that synchronizes simulation time across them (IEEE 2025a). Its object model template lets each federation define the data its members exchange (IEEE 2025b). IEEE Std 1730, the Distributed Simulation Engineering and Execution Process (DSEEP), adds a vendor-neutral, seven-step process for engineering federations built on either DIS or HLA (IEEE 2023).

Of the three, DIS most directly informs NOS-T. Its PDUs carry entity state, events, and simulation-management commands, the last governing exercise control through start, stop, and freeze (IEEE 2012). NOS-T adapts this simulation-management pattern for coordinated scenario time control as freeze, resume, and stop, carrying it over AMQP rather than the User Datagram Protocol (UDP) multicast transport DIS historically specified (IEEE 2015). The full standards family offers far more than that single pattern: it has anchored mature interoperability across defense, aerospace, and training for over three decades, supported by robust vendor tools and well-defined conformance testing. Its breadth, however, exceeds the needs of early-stage research and development: DIS requires extensive PDU and synchronization support, HLA demands sophisticated federation management and RTI integration, and DSEEP's seven-step process adds overhead disproportionate to early-stage projects, though its sequence of steps remains a sound guide even where full conformance is not pursued. These designs reflect their military and training origins, so applying them to civilian space and Earth-science work requires conceptual translation rather than direct reuse.

2.3 Testbed Landscape

Distributed-systems testbeds span a wide range of approaches and frameworks suited to different needs. No single point on that range serves every need, so a testbed's design follows from what its campaigns must serve.

2.3.1 Taxonomy and Common Elements

The spectrum of distributed testbed approaches spans physical, virtual, hybrid, and federated systems, trading fidelity against cost and flexibility. Physical testbeds offer the highest fidelity and authentic failure modes using real hardware such as constellation mockups and Internet of Things (IoT) device networks, but at high cost and limited scale. Virtual testbeds are software-based and therefore cost-effective,

scalable, and rapidly reconfigurable at some loss of realism. Hybrid testbeds couple the two through standardized interfaces, and federated testbeds distribute across institutions for large-scale collaboration at the cost of coordination, security, and resource-management complexity (Maier 1998).

Regardless of approach, effective testbeds share common elements. Communication infrastructure built on message-oriented middleware (MQTT, AMQP, or proprietary protocols) supports publish–subscribe, request–response, and multicast patterns, while time management enables accelerated execution and coordinated pause/resume. Resource and identity management handles multi-user, multi-organization authentication and authorization, and data integration incorporates real-world and synthetic streams. Monitoring and instrumentation cover performance measurement, logging, and visualization, while configuration and orchestration support declarative scenario specification, automated deployment, and lifecycle management.

2.3.2 Representative Frameworks

Representative frameworks range from comprehensive operational systems built for mission deployment to lightweight platforms designed for early-stage research and development. Two dimensions separate them: integration agility—the setup and expertise a framework demands—and collaboration scope—how well it supports multiple institutions working together. Each established framework is strong on one dimension but not both.

The Goddard Mission Services Evolution Center (GMSEC), the standardized mission-operations infrastructure of NASA, offers a comprehensive message catalog, pluggable middleware, and multi-language Application Programming Interfaces (APIs), giving it broad, multi-institution reach and a long record of operational use (NASA Goddard Space Flight Center 2024). That mission focus, however, weighs against research: its schemas target spacecraft commanding and telemetry rather than exploratory scenarios, it favors reliable data exchange over simulation orchestration such as scenario time control, and its configuration overhead slows early-stage iteration.

Academic space-systems platforms instead focus on detailed spacecraft-level modeling. The NASA Operational Simulator for Small Satellites (NOS3) runs unmodified flight software against simulated spacecraft hardware and dynamics for small-satellite development (Geletko et al. 2019). PAseos Simulates the Environment for Operating multiple Spacecraft (PASEOS) models spacecraft operational constraints—power, thermal, communication windows, and radiation—across both high-performance and real-time edge execution (Gómez et al. 2024). Both deliver deep spacecraft-level fidelity, but as single-user tools they lack the multi-participant coordination that cross-institution experiments require.

The Hierarchical Engine for Large-scale Infrastructure Co-Simulation (HELICS), from outside the space domain, couples independently developed simulators through distributed time coordination and a layered broker hierarchy, maintained as a community-driven, open-source project (Hardy et al. 2024). Its co-simulation model, however, assumes participants join a shared runtime as instrumented federates, and it targets terrestrial power, natural gas, communication, and transportation systems

rather than space missions. The Robot Operating System (ROS), from robotics, routes strictly typed messages over named topics between nodes that need not know one another, using a language-neutral definition scheme and tooling that instantiates an entire node graph from one declarative file (Quigley et al. 2009). It is used in both simulation and operational systems, but its peer-to-peer design assumes a compute cluster under a single owner, so it supplies neither federated identity nor the durable delivery and coordinated scenario time that cross-institutional campaigns require.

None of these five frameworks can be adapted to cover both dimensions: each lacks several of the needed capabilities—cross-institution identity and access control, durable and schema-governed messaging, automated session recovery, and scenario time control that can pause for externally managed models. NOS-T was built to close these gaps.

2.3.3 NOS-T Version 1: Design and Operational Baseline

NOS-T Version 1 began closing these gaps in the testbed landscape, pairing lightweight integration with multi-institutional participation. Its capabilities met many of them, but sustained use surfaced further requirements. It emphasized ease of adoption, loose coupling, multi-language interoperability, and modularity, letting teams incrementally integrate existing analysis scripts and data sources—both synthetic and real—while experimenting with coordinated scenario time control and publish–subscribe messaging patterns. The messaging layer adopted the proprietary Solace PubSub+ Standard Edition broker for publish–subscribe over the lightweight MQTT protocol, supporting on the order of 10^3 concurrent connections and 10^4 messages per second. An event-driven pattern with hierarchical topic namespaces and wildcard subscriptions let modular applications exchange state changes (e.g., orbital states, visibility events, sensor products) while remaining independently deployable. Applications ran in one of two modes: managed applications synchronized to a central scenario time manager, while unmanaged applications advanced on local wall-clock, letting tightly timed components (orbital propagators) coexist with loosely coupled clients (visualizations) that did not require strict temporal coordination (Chell et al. 2023).

The FireSat+ campaign exercised core behaviors of a distributed Earth observation concept—orbital propagation, coverage estimation, and fire-detection fusion—with Python spacecraft and analytics components operating alongside JavaScript/Cesium visualization clients to demonstrate cross-language participation. The campaign validated that loosely coupled components could sustain coordinated exchanges without centralized monolithic orchestration. Its benchmarks showed latency was largely insensitive to message frequency within broker throughput limits but scaled approximately linearly with payload size (˜1.5 ms per kB), so data-rich schemas imposed cumulative delay that constrained achievable time-acceleration factors. V1 therefore established a working baseline rather than a finished platform: widening collaboration across institutions and integrating externally managed models both demanded capabilities it lacked, and R1–R5 specify them.

2.4 Toward Science-Oriented Testbeds

Novel observing strategies resolve one physical process from multiple vantage points and across spatial, spectral, temporal, and radiometric dimensions (Le Moigne et al. 2019). No single observation type meets all measurement requirements for complex variables such as Snow Water Equivalent (SWE), the depth of liquid water held in a snowpack, so contemporary Earth observation combines passive microwave, active radar, optical, and in situ sensing (Hale et al. 2025; Tsang et al. 2022). Sensors differing in temporal resolution, spatial scale, and latency are among the spacecraft, ground, and analytics nodes a DSM testbed must synchronize.

Synchronizing those nodes demands shared meaning as well as shared timing. The Earth-science community solves the equivalent problem for models through model-coupling frameworks, where the Earth System Modeling Framework (ESMF) composes weather and climate models as components with standardized interfaces and the Assistance for Land Modeling Activities (ALMA) convention standardizes variable names and units across land-surface models (Hill et al. 2004). Those conventions bind participating models into a single tightly integrated system, which a testbed coordinating independently managed applications cannot require.

A testbed serving early-stage DSM research and development must meet established science models on their own terms: NASA's Land Information System (LIS), a high-resolution land-surface model constrained to institutionally managed high-performance infrastructure, is the kind of external component such a testbed must integrate rather than replace (Kumar et al. 2006, 2008). It must also balance three competing priorities: sufficient architectural discipline to maintain semantic interoperability as participation scales, minimal integration friction for rapid prototyping, and conceptual alignment with mature standards to preserve migration paths toward operational deployment.

Together, NOS-T V2 and V3 meet these priorities by composing proven, general-purpose open-source components into a lightweight stack. The evolved platform sits primarily between virtual and federated testbeds, where its messaging, identity, and scenario-time coordination mechanisms support distributed participation across institutions and administrative domains. NOS-T shares the model-coupling frameworks' goal of semantic interoperability but governs message schemas across loosely coupled, distributed applications rather than binding them into a single tightly integrated system. Crucially, it couples independently managed models through loose, event-driven exchange and coordinated scenario-time freezes, so a model such as LIS behind a firewall can contribute to a scenario while remaining in its own computing environment—a lightweight, federated combination that no surveyed framework achieves (Figure 2).

3 NOS-T Platform

The NOS-T platform comprises a layered infrastructure and the NOS-T Tools library with three application roles, supporting DSM research workflows such as orbital propagation, coverage estimation, sensor fusion, and environmental data analytics. It treats coordinated scenario time control as a first-class primitive rather than an optional add-on.

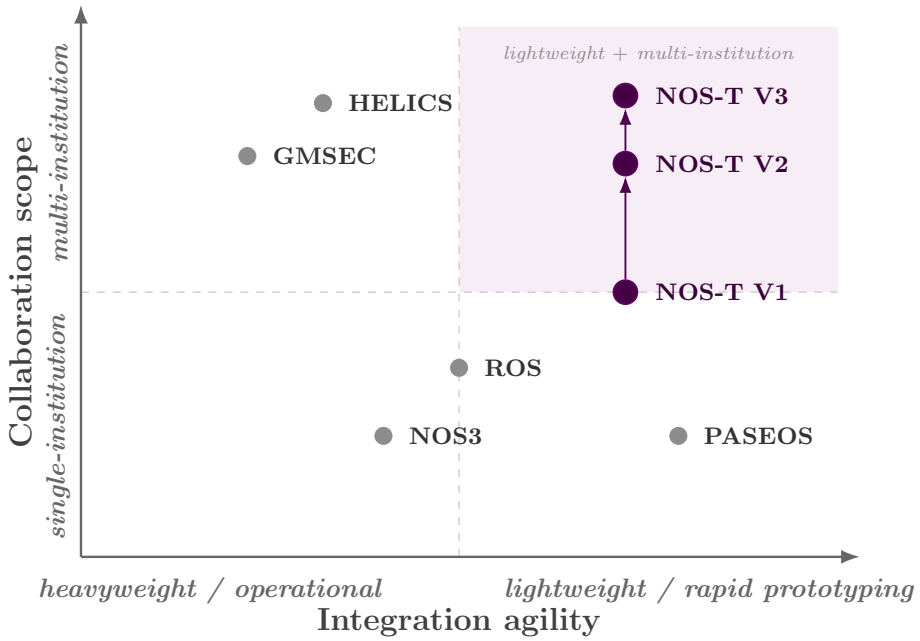


Fig. 2 Approximate positioning of NOS-T relative to representative distributed simulation and testbed frameworks along two axes: integration agility (heavyweight and operational to lightweight and rapid-prototyping) and collaboration scope (single- to multi-institution and federated). Markers place GMSEC, HELICS, NOS3, PASEOS, and ROS. Arrows trace NOS-T's progression across V1–V3 into the otherwise-unoccupied region combining low integration overhead with multi-institution, federated coordination. Placements are qualitative, reflecting the gap analysis rather than measured values.

3.1 System Architecture

The evolved NOS-T architecture is layered, with each layer addressing one validated V1 limitation to preserve traceability from operational evidence to implementation. NOS-T V2 delivered the identity, messaging, session, and schema layers, and NOS-T V3 added the temporal coordination layer (Figure 1, Supplementary Table S1). The infrastructure services—the message broker and identity provider—deploy as containerized components on a shared cloud or on-premises host, so participants from multiple institutions connect to a common testbed over the network from their own computing environments rather than each standing up separate infrastructure, giving the platform a cloud-native, multi-tenant deployment model.

Identity and Access.

The Identity and Access Management (IAM) layer addresses R1 by replacing static credentials with Keycloak-backed, role- and attribute-based access control across multiple organizational realms. Short-lived, refreshable OAuth2/OIDC tokens for service and user principals renew automatically, eliminating manual credential rotation, while central session logging captures audit trails meeting NASA compliance requirements.

Single Sign-On (SSO) and Multi-Factor Authentication (MFA) integration accommodates institutional authentication policies. An application obtains a token from Keycloak and presents it to the broker, which admits the connection and permits message exchange (Figure 3).

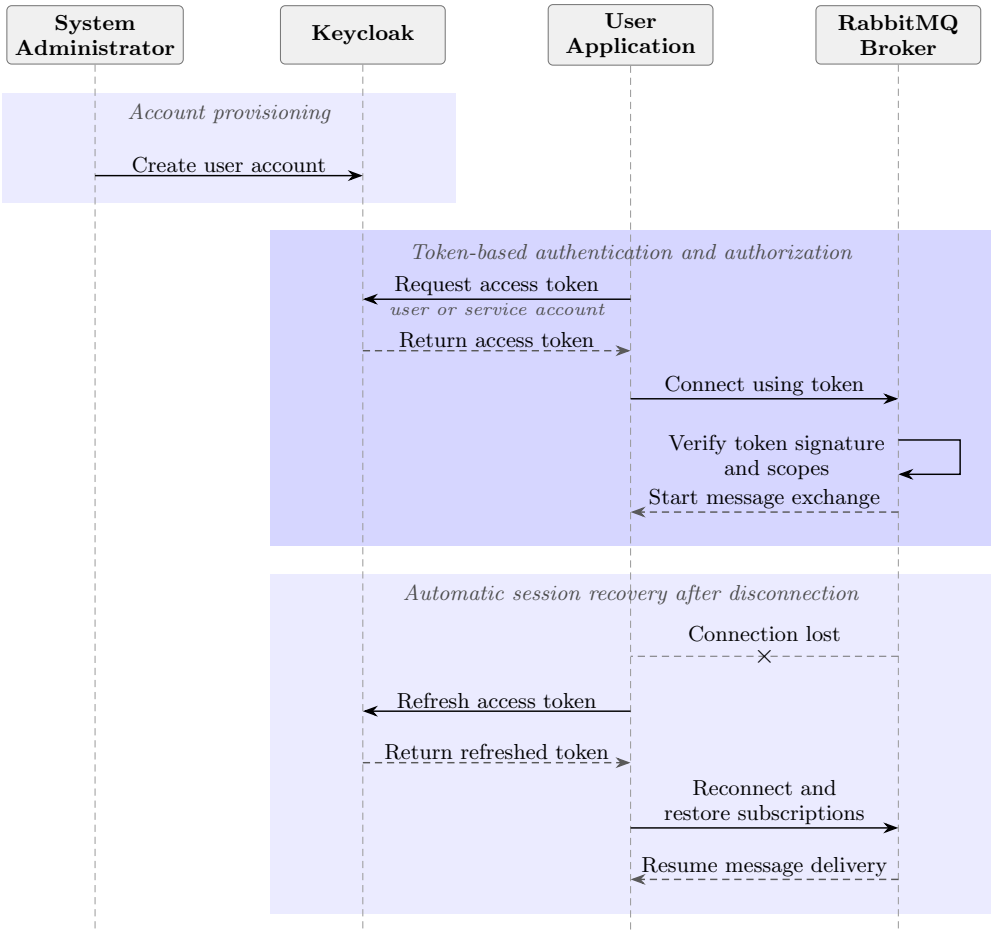


Fig. 3 Token-based authentication and automatic session recovery in NOS-T V2. A system administrator first provisions the account in Keycloak. The user application then requests a short-lived OAuth2/OIDC access token, presenting either a user account or a service account, and presents the returned token to the RabbitMQ broker, which admits the connection and permits message exchange. The broker retrieves and caches Keycloak's token signing keys, then verifies each token locally. When a connection drops, the application waits a fixed, configurable delay, refreshes its access token, and reconnects, restoring its subscriptions. Dashed arrows denote responses.

Messaging.

The messaging layer addresses R2 by replacing best-effort MQTT with AMQP over RabbitMQ, using a topic exchange backed by durable queues. Publisher confirmations

and consumer acknowledgments enforce at-least-once delivery. Durable queues retain messages through unexpected disconnections, and any message an application cannot send during an outage is queued locally and re-sent automatically once the connection is restored, so no data is lost unless an outage leaves more unsent messages than that queue holds, 5000 by default.

Session Resilience.

The session layer addresses R3 by keeping long-running applications connected with minimal operator intervention. A heartbeat monitors each connection, and when one drops unexpectedly the application reconnects automatically after a fixed, configurable delay, first refreshing its Keycloak access token in case it expired during the outage (Figure 3). On reconnection it restores its message subscriptions, resuming delivery without manual reconfiguration, and signal handlers let applications shut down cleanly when interrupted.

Schema Validation.

The schema layer addresses R4 with Pydantic-defined message schemas that can validate payloads at runtime and reject malformed messages, providing the mechanism to keep data consistently structured across the distributed applications. The scenario-control message set follows the DIS simulation-management command pattern (Section 2.2). Standardized topic-naming conventions scope messages by application and function.

Temporal Coordination.

The temporal coordination layer, introduced in V3, addresses R5 through a central Manager that issues freeze, resume, and time-scale commands and collects acknowledgments from participating applications. Managed applications can also request a freeze, resume, or time-scale change, which the Manager arbitrates and broadcasts so that every application freezes and resumes at the same scenario time (Supplementary Figure S1). Scenario time advances independently of wall-clock time, so the Manager can accelerate long campaigns or freeze scenario time while external models compute, without consuming simulated campaign time.

3.2 NOS-T Tools Library

The NOS-T Tools library is a Python package that encapsulates platform capabilities into reusable, high-level abstractions, prioritizing low-friction integration for research workflows while preserving access to platform-level features. It forms the developer-facing interface to the platform infrastructure, so contributors build applications against the library rather than integrating with the message broker and identity provider directly. Advanced users can also stand up their own broker and identity provider, making it straightforward to develop and deploy a self-hosted architecture.

Application Roles.

The library defines three application roles—the Manager, Managed Applications, and Unmanaged Observer Clients—all built on a common base `Application` class.

The Manager, implemented with the `Manager` subclass, is the central coordination point, issuing time-control commands and aggregating health and heartbeat reports to orchestrate the scenario. Managed Applications, implemented with the `ManagedApplication` subclass, are domain-specific simulators or analytics services that honor scenario-time advancement, emit periodic time-status telemetry, and validate their outgoing messages. Unmanaged Observer Clients are deployed as the base `Application` class and subscribe passively, monitoring the scenario without participating in coordinated time advancement. The library provides one such observer, the `LoggerApplication`, which subscribes to a topic and records all messages to file.

Configuration.

A single YAML Ain't Markup Language (YAML) configuration file, validated on load by Pydantic, defines the broker and identity servers, the message channels and exchanges, and the execution parameters. Credentials are read from a local environment file or environment variables, supporting Keycloak user and service accounts as well as non-Keycloak username/password access for local development. Per-application parameters can be supplied in the same file, so each application retrieves its own configuration at startup, with Keycloak tokens acquired and refreshed automatically (Supplementary Figure S2).

Messaging Adapters.

Messaging adapters wrap publish–subscribe interaction so applications exchange typed payloads without calling the broker directly. Routing keys follow a standard `prefix.application.topic` convention, and registered callbacks dispatch incoming messages to application logic. If the connection drops, outgoing messages are queued and delivered automatically once it is restored.

Callbacks and Events.

Three callback types, built on an observer pattern, support distinct coordination needs: property-change callbacks that fire when a tracked value changes, scenario-time interval callbacks that fire at fixed intervals of scenario time, and wall-clock interval callbacks that fire at fixed intervals of real time. A common registration interface manages their lifecycle, reducing integration boilerplate.

Scenario Time.

The library exposes two classes of scenario-time operation. The Manager issues commands directly: initialize, start, and stop bracket an execution, while freeze, resume, and time-scale updates control its progress. Managed applications issue the corresponding requests—to freeze, resume, or change the time scale—which the Manager arbitrates, letting participants drive scenario time cooperatively without managing the underlying coordination themselves.

Diagnostics.

Periodic time-status messages carry timestamps that support end-to-end latency analysis, and each application writes a timestamped `.log` file of its connection events,

message latencies, and state transitions for campaign monitoring. Log files rotate to numbered backups once they reach a configured size, so long campaigns retain a complete record.

4 Methods

4.1 Overview

The Snow Observing Strategy (SOS) demonstration provides integrated system validation of the NOS-T platform under realistic operational conditions, exercising the five architectural requirements (R1–R5) realized in NOS-T V2 and V3 within a mission-representative scenario rather than synthetic benchmarks. It evaluates the platform's ability to coordinate distributed applications, manage long-duration accelerated simulations, and integrate external computational models in a closed-loop observing system—a step toward adaptive Earth science mission planning.

At its core is a scientifically motivated problem: optimizing SWE estimation in the Missouri River Basin through selection of Synthetic Aperture Radar (SAR) observations to complement Passive Microwave (PMW) measurements. The PMW measurements provide frequent global coverage at coarse spatial resolution and saturate in deep snow, whereas SAR measurements offer higher spatial resolution but struggle with wet snow and complex terrain (Tsang et al. 2022). The demonstration integrates NASA's LIS, a high-resolution land surface model, with four distributed NOS-T applications—Manager, Planner, Appender, and Simulator—to evaluate observation strategies for SnowGlobe, a mission concept for a constellation of small satellites equipped with X- and Ku-band SAR designed to improve global SWE estimates (Kumar et al. 2006; Hale et al. 2025; Cho et al. 2023).

Two sequential demonstrations targeted distinct snow seasons and progressively hardened the synchronization and authentication architecture. The first campaign spanned the 2024–2025 accumulation and melt cycle within water year 2025 (October 1, 2024 through June 30, 2025) and validated V2 and V3 capabilities under sustained operational conditions, using user account authentication and time-based synchronization between NOS-T and LIS, in which NOS-T resumed each scenario-time freeze on a fixed timer. The second campaign targeted the 2018–2019 season within water year 2019 (October 1, 2018 through August 1, 2019), selected for its notable snow conditions in the Missouri River Basin, and validated the event-driven synchronization architecture exclusively, in which NOS-T froze indefinitely and resumed only when LIS completed. It replaced user account authentication with service accounts, removing the 12-hour SSO session limit identified in the first campaign. Throughout this section, the two campaigns are referred to as Demo 1, the time-based campaign, and Demo 2, the event-driven campaign (Table 1).

4.2 Integration Architecture

Both campaigns share a loosely coupled integration between four NOS-T applications and NASA's LIS land surface model executing on firewalled supercomputer infrastructure. While the NOS-T applications coordinate via AMQP messaging within the

Table 1 Comparison of the two SOS campaigns: time-based Demo 1 and event-driven Demo 2.

Attribute	Demo 1	Demo 2
Architecture		
Synchronization	Time-based	Event-driven
Authentication	User account	Service account
Scenario		
Period	Oct 1, 2024 – Jun 30, 2025	Oct 1, 2018 – Aug 1, 2019
Duration	273 days	304 days
Acceleration factor	72× → 60×	60×
Freeze mode	Timed	Indefinite
Wall-clock		
Period	Sep 24 – Oct 25, 2025	Feb 25 – Mar 31, 2026
Duration	31 days	34 days
Per scenario-day	20 → 24 min	24 min
Per freeze	1 h → 2 h	Variable (∼27 min)

NOS-T Tools library, LIS integration across institutional boundaries requires indirect communication via Amazon Web Services (AWS) Simple Storage Service (S3) buckets under two synchronization patterns, time-based and event-driven, described below.

Four NOS-T applications form a closed loop with LIS, exchanging data products through S3 while coordinating among themselves over AMQP (Figure 4, Supplementary Figure S3). The **Manager** orchestrates the simulation lifecycle, coordinates temporal synchronization across the applications, arbitrates freeze, resume, and time-scale commands, and monitors application health. The **Planner** is the decision component. On each scenario day it retrieves the latest LIS land-surface outputs from S3, namely SWE change, snow cover fraction, and surface temperature, and combines them with SnowGlobe sensor characteristics and the PMW ground tracks that define the baseline coverage to derive five per-pixel efficiency layers: SWE change, snow cover fraction, surface temperature, sensor saturation, and resolution. These layers are integrated into a per-pixel reward, the observation value that SnowGlobe SAR coverage would add beyond the PMW baseline, and a budget-constrained integer optimization then maximizes total reward subject to a fixed per-cycle selection limit, yielding the high-reward observation regions. The selected regions are published over AMQP and uploaded to S3, and on each scenario-day change the Planner requests a coordinated scenario freeze from the Manager. A companion study developed and evaluated the Planner's reward formulation and optimization outside the testbed (Banafsheh et al. 2026). The **Appender** maintains the shared observation ledger, assigning each selected region a persistent identifier, tracking its lifecycle, and mediating the Planner-to-Simulator handoff. The **Simulator** models SnowGlobe constellation operations—five Ku-band SAR satellites in a sun-synchronous Walker constellation—evaluating observation opportunities by comparing satellite ground tracks and swath geometry against each target region under a daily capacity constraint, and recording feasible observations with their ground-track footprints as Geographic

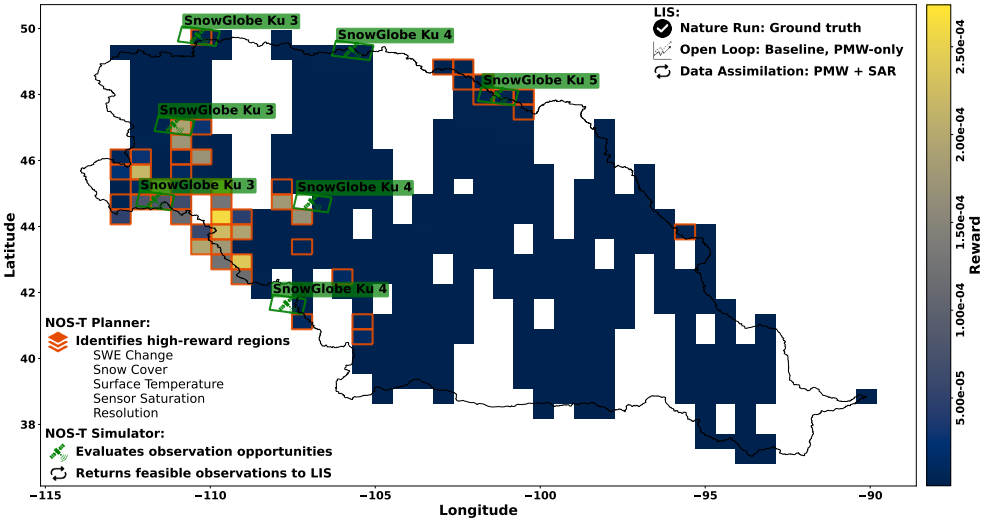


Fig. 4 SOS operational workflow on a representative scenario day, illustrative of both Demo 1 and Demo 2, mapped over the Missouri River Basin. The background heatmap shows per-pixel reward. The NOS-T Planner selects high-reward observation regions (orange) from efficiency layers—SWE change, snow cover fraction, surface temperature, sensor saturation, and resolution—and the Simulator evaluates observation opportunities for the SnowGlobe Ku-band SAR constellation (green footprints), returning feasible observations to LIS. The legend lists the Planner and Simulator actions and the three LIS modes.

JSON (GeoJSON) products uploaded to S3 for LIS assimilation. The results reported here focus on the Manager, Planner, and Simulator, while the Appender, integral to the loop but intermediary in role, remains outside the analyses.

LIS operates in three modes: a Nature Run providing ground truth, a PMW data assimilation baseline, and a joint data assimilation combining PMW data with SAR observations (Wrzesien et al. 2022). This demonstration employs LIS's data assimilation mode, which combines satellite observations with model state estimates—a capability that the NOS-T distributed architecture extends by enabling observation selection based on efficiency metrics derived from LIS outputs. Exchanging those outputs with NOS-T, however, is constrained by where LIS runs: the NASA Center for Climate Simulation (NCCS) Discover supercomputer restricts direct communication protocols such as AMQP messaging. While migrating LIS to an alternative compute environment—such as a laboratory server or AWS Elastic Compute Cloud (EC2) instance—would enable direct messaging, this demonstration deliberately imposed minimal requirements on LIS to preserve its existing workflow. Such constraints reflect operational realities where scientific models execute on compute environments dictated by institutional policy rather than integration convenience. Addressing these challenges strengthens NOS-T Tools' robustness and ensures testing conditions representative of real-world deployment scenarios.

4.3 Experimental Design

Both SOS campaigns shared a common compute environment and instrumentation, differing in configuration, including synchronization approach, authentication model, and target snow season (Table 1). Across both, the four NOS-T applications exercised V2 capabilities (persistent messaging, automated reconnection, schema validation) and V3 enhancements (application-requested scenario freezes, coordinated time control) while integrating LIS executing on external supercomputer infrastructure. The NOS-T applications executed on a Dell Precision 3260 workstation (Intel Core i9-13900, 24 cores and 32 threads, 64 GB RAM, 1 TB NVMe SSD), LIS ran on NCCS Discover, and the NOS-T infrastructure services—RabbitMQ message broker and Keycloak authentication—were hosted on an AWS EC2 m5.xlarge instance (4 vCPUs, 16 GB RAM) in the us-east-1 (N. Virginia) region.

Scenario Specifications.

Each campaign reproduced a complete snow accumulation and melt season in the Missouri River Basin—the 2024–2025 season (October 1, 2024 through June 30, 2025) in Demo 1 and the 2018–2019 season (October 1, 2018 through August 1, 2019), selected for its notable snow conditions, in Demo 2 (Table 1). Scenario time advanced under acceleration, compressing each 24-hour scenario day into roughly 20–24 minutes of NOS-T execution before each freeze. In Demo 1, LIS ran on a cron schedule, so each execution-plus-freeze cycle had to fit a whole number of times into the 24-hour day. When the freeze was lengthened to accommodate LIS processing, execution rose from 20 to 24 minutes to preserve exactly ten cycles per day. Both campaigns ran over comparable wall-clock windows—Demo 1 from September 24 to October 25, 2025 (31 days) and Demo 2 from February 25 to March 31, 2026 (34 days).

SnowGlobe Mission Parameters.

SnowGlobe, a satellite mission concept rather than a deployed constellation, was modeled identically across both campaigns: five Ku-band SAR satellites in 600 km sun-synchronous orbits provided ∼5-day global revisit (1–3 days north of 60°) with 250 m spatial resolution, and performance specifications target SWE error <20% for snowpack >50 mm and ±10 mm for shallow snowpack <50 mm. Neither campaign operated in true taskable mode. Instead, the Planner identified high-reward regions from efficiency metrics calculated on LIS land-surface outputs, the Simulator determined which simulated SnowGlobe observations, generated from modeled orbital coverage, intersected those regions, and those observations were returned to LIS for assimilation (Supplementary Figure S3).

Data Flow and Storage.

In both campaigns, LIS uploaded daily NetCDF files to scenario-date-partitioned S3 locations, where the Planner and Simulator likewise staged GeoJSON observation products for LIS assimilation. These files formed a closed loop that advanced one scenario day at a time: the Planner selected regions from the LIS land-surface state available for that day, the Simulator determined which of those requests SnowGlobe

could meet, and LIS assimilated the resulting observations, updating the state the Planner read on the next scenario day.

Temporal Coordination Strategy.

Integrating LIS batch processing with continuous NOS-T operations required scenario time freezes. Each scenario day followed a four-phase cycle: (*i*) NOS-T applications executed under acceleration for a 20–24 minute processing window, (*ii*) the Manager issued a coordinated freeze command preserving scenario time state, (*iii*) LIS retrieved outputs from S3 and computed within the freeze, and (*iv*) the Manager resumed scenario time. This freeze-resume coordination occurred 273 times in Demo 1 and 304 times in Demo 2, validating precise temporal synchronization under repeated operational stress—a coordination pattern enabled by programmatic, fine-grained time control. The two demonstrations differed in how that resume was triggered (Figure 5). In the **time-based approach**, used in the first demonstration, the freeze ran for a fixed 1–2 hours before applications resumed automatically. This proved inefficient because wall-clock execution times varied on the shared supercomputer—queue wait times exceeded or fell below the nominal allocation, resulting in either insufficient runtime or unnecessary idle waiting. In the **event-driven approach**, adopted exclusively for the second demonstration, applications instead held an indefinite freeze until LIS uploaded its output to the S3 bucket. That upload triggered an AWS Lambda function, which used the NOS-T Tools library to publish a resume request through the testbed broker, and the Manager arbitrated that request into a coordinated resume command received by every participating application. Using data availability rather than fixed time intervals minimized latency between output readiness and workflow initiation while maintaining institutional firewall compliance.

Performance Instrumentation.

In both campaigns, each NOS-T application generated its own `.log` file, timestamped to microsecond precision, recording its connection events, message latencies, and operational state transitions. These included per-application scenario-time and wall-clock latencies sampled at 10-second intervals, and the stage-by-stage timing of each freeze event (request initiation, Manager receipt, internal processing duration, and per-application acknowledgment). The Manager computed each latency as

$$\ell = t_{\text{mgr}} - t_{\text{app}}, \tag{1}$$

where t_{app} is the time an application reported in its status message, t_{mgr} is the Manager's clock when it processed that message, and ℓ is the resulting latency—how stale the report was on arrival—evaluated separately in scenario time and wall-clock time. Every application corrected its wall-clock timestamps against a Network Time Protocol (NTP) server, querying it in a background thread and adding the returned offset to each reported time, so ℓ reflects delivery lag rather than drift between machine clocks. The server and the query interval are configured per deployment, and both campaigns used `pool.ntp.org` at 60-second intervals. These logs provided the disconnection and

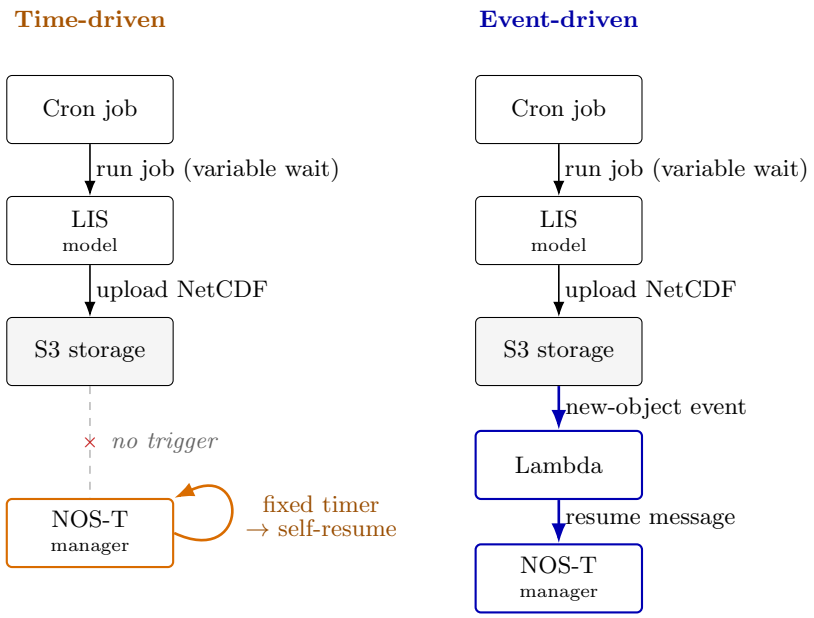


Fig. 5 Synchronization paths contrasting time-driven and event-driven coordination of NOS-T resume with external LIS processing. In both approaches, a cron job runs LIS on the supercomputer—after a variable scheduling wait—and LIS uploads daily Network Common Data Form (NetCDF) outputs to S3. The approaches differ only in how NOS-T leaves its scenario-time freeze. In the time-driven approach, the Manager resumes on an internal fixed timer, independent of LIS completion. In the event-driven approach, a new object in S3 triggers an AWS Lambda function that publishes a resume message, so resume tracks data availability. In both cases LIS–NOS-T data exchange uses S3 as the storage layer, while NOS-T applications coordinate over AMQP messaging on the lab server.

reconnection records, message-queue persistence checks, and freeze timings underlying the latency decomposition and session resilience analyses reported here.

4.4 Data Analysis

All quantitative analyses were performed in Python from the per-application `.log` records, with latencies evaluated in both scenario time and wall-clock time following Equation 1. Log records were parsed and aggregated with pandas 2.2.3 and NumPy 2.2.4, extracting per-application connection, freeze, and message-queue events. Latency and processing-time series were aggregated per scenario day and summarized by the daily maximum (wall-clock latency) or the per-cycle value (freeze propagation and processing time), so that transient excursions remained visible against the baseline.

Per-day trends were smoothed with Locally Weighted Scatterplot Smoothing (LOWESS) (statsmodels 0.14.4, smoothing fraction 0.1, or 0.07 for the observation-selection series), and the band accompanying each trend denotes a rolling interquartile range (25th–75th percentile in a centered window) rather than raw points. For the Demo 1 processing-time and idle-time series, the LOWESS trend was fit piecewise on either side of the 72×→60× acceleration change at scenario day 159 to preserve the

discrete step rather than smooth across it. Per-stage and per-application distributions were rendered as violin plots annotated with the mean, and the wall-clock latency violins omitted reconnection-spike outliers for legibility.

The spatial association between per-cell selection frequency and terrain elevation was quantified with the Spearman rank correlation coefficient (scipy 1.13.1), reported with its two-sided p-value. Upload-interval regime statistics excluded the isolated multi-hour gaps produced by manual restarts while retaining them in the plotted timelines. Figures were produced with Matplotlib 3.9.4.

5 Results

5.1 Simulation Performance

Both campaigns sustained low message latency across their full duration without critical connection failures requiring manual intervention. The latency held a dense near-zero baseline interrupted only by brief, recoverable spikes—maxima of 16.3–16.6 s in Demo 1 and 14.3–16.4 s in Demo 2—that align with disconnect/reconnect episodes rather than sustained degradation (Figure 6). These maxima reflect the configured 15 s reconnection delay in NOS-T Tools—the interval a dropped client waits before retrying—not a performance limitation. Mean wall-clock latency (Equation 1) was 86 ms (Manager), 147 ms (Planner), and 137 ms (Simulator) in Demo 1 and 75 ms, 76 ms, and 128 ms in Demo 2 (Supplementary Figure S4).

5.2 Session Resilience

Automated reconnection eliminated operator intervention in both campaigns. Demo 1 recorded 46 disconnection events (10 Manager, 14 Planner, 22 Simulator) and Demo 2 recorded 59 (23 Manager, 7 Planner, 29 Simulator) (Figure 7A). Recovery averaged 15.4 s in Demo 1 and 15.3 s in Demo 2 (Figure 7B). Per-application mean recovery was 15.26 s (Manager), 15.42 s (Planner), and 15.46 s (Simulator) in Demo 1 and 15.23 s, 15.23 s, and 15.45 s in Demo 2 (Supplementary Figure S5). These recovery times include the configured 15 s reconnection delay, the fixed wait before a dropped application retries, so they are a reconnection cost rather than a floor on latency. Latency stays sub-second in normal operation and rises only for reports queued during an outage, which the Manager processes late by up to the outage length. Outages typically end at the first retry, so the resulting staleness clusters near the 15 s delay. Each application keeps advancing its own scenario clock throughout a disconnect—scenario time never freezes—so the applications do not fall behind, and latency returns to its sub-second baseline once reports flow again.

During these disconnection events, applications could not send messages over the lost network connection. Instead, the NOS-T Tools library queued each undelivered message locally—28 in Demo 1 and 45 in Demo 2. All were delivered once the application reconnected. This yielded zero message loss across both campaigns, in contrast to V1's best-effort MQTT configuration, which offered limited delivery guarantees and no automated reconnection (Figure 7C).

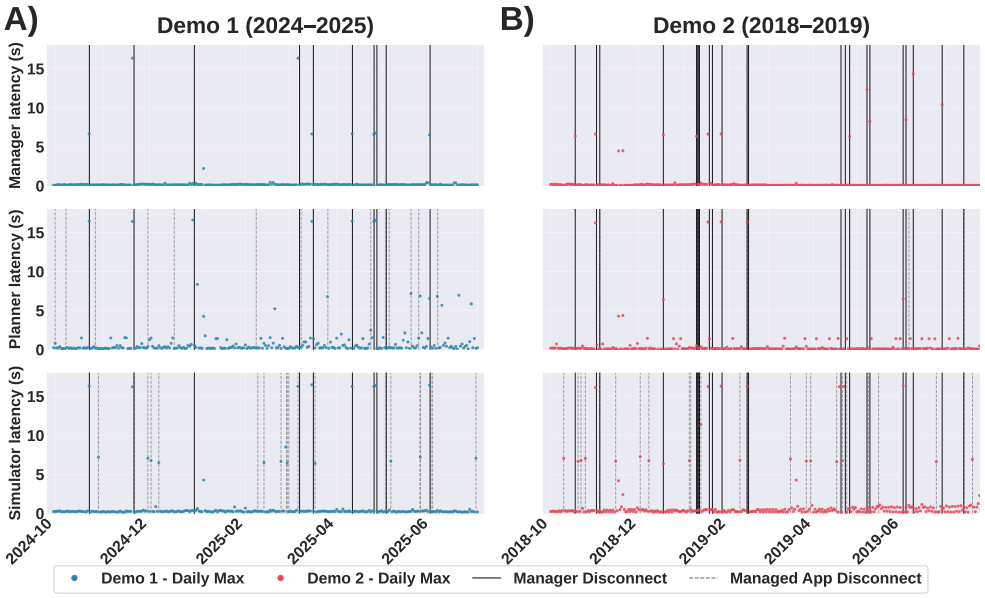


Fig. 6 Latency and connection stability for the Manager, Planner, and Simulator applications across both SOS campaigns—Demo 1 (273-day 2024–2025 season, panel A) and Demo 2 (304-day 2018–2019 season, panel B), each column showing the three applications stacked by row. Wall-clock latency is aggregated per scenario day: points mark each day's maximum (blue for Demo 1, red for Demo 2), so the rare reconnection spikes stand up from the otherwise flat, sub-second baseline of low daily maxima. Thin black vertical lines mark Manager disconnections affecting all managed applications, and gray dashed lines mark individual managed-application disconnections. Transient spikes peak near the configured 15 s reconnection delay and recover automatically.

5.3 Scenario Time Freeze Coordination

Coordinated scenario time freezes (R5) paused the NOS-T applications on 273 scenario days in Demo 1 and 304 in Demo 2 so that LIS could perform its batch processing. Freeze propagation to all applications remained sub-second throughout. Mean end-to-end propagation latency was 163.2 ms in Demo 1 and 125.3 ms in Demo 2 (Figure 8A). These latencies remained tightly bounded, with event-level distributions clustered near their means. Each freeze decomposed into request-to-Manager transit, near-instant Manager processing, and Manager-to-application broadcast. Manager internal processing held at 0.3 ms in both campaigns, and broadcast reception remained comparable at 67.5–72.8 ms in Demo 1 and 58.5–72.9 ms in Demo 2 (Figure 8B). The campaign-to-campaign difference therefore concentrated almost entirely in request-to-Manager transit, at 93.4 ms in Demo 1 versus 58.6 ms in Demo 2 (Supplementary Figures S6 and S7).

5.4 Operational Efficiency

Temporal coordination (R5) produced the most operationally significant contrast between the campaigns. Demo 1 used time-based synchronization and incurred idle-waiting under variable supercomputer queue times, whereas Demo 2 introduced

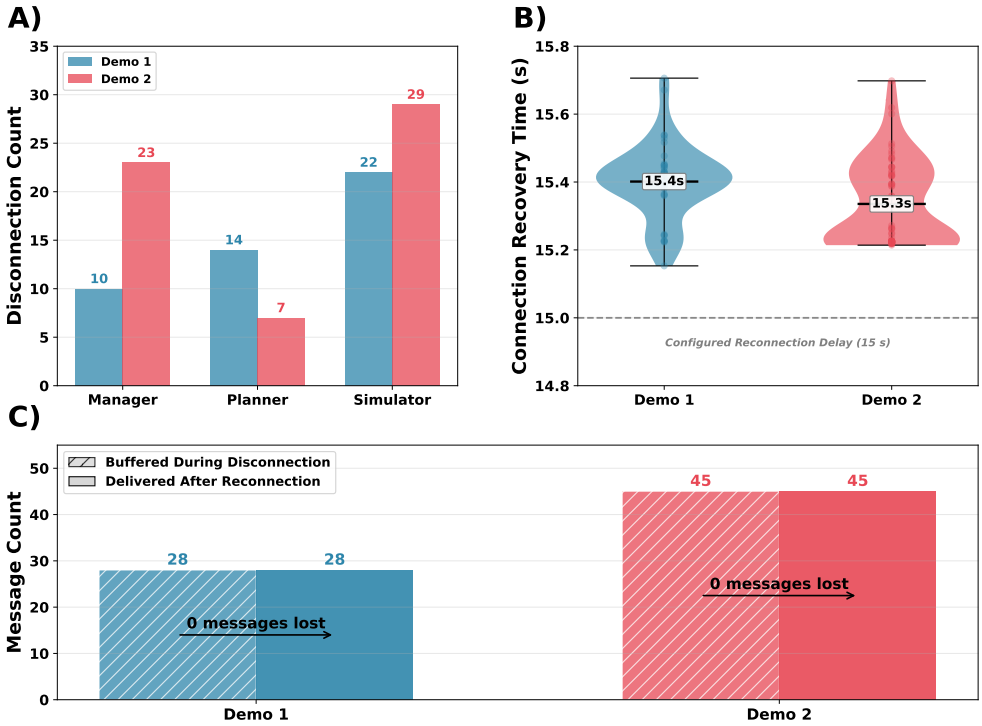


Fig. 7 Session resilience and durable-message validation across the Demo 1 and Demo 2 SOS campaigns. **(A)** Disconnection counts for the Manager, Planner, and Simulator applications over the 31-day and 34-day wall-clock runs. **(B)** Paired reconnection recovery-time distributions. The dashed line marks the configured 15 s reconnection delay and labels show means. **(C)** Buffered messages during outages and messages delivered after reconnection. Both demonstrations delivered every buffered message, yielding zero message loss.

event-driven synchronization, in which the applications entered an indefinite freeze after their execution window and resumed when an AWS Lambda trigger fired. Queue waits on Discover reflect contention among all users of a shared facility and lengthen around conference and proposal deadlines, so the wait a fixed timer must absorb is neither constant nor predictable across a months-long campaign. The LIS workload itself was comparable between campaigns—mean processing 31.6 min in Demo 1 and 26.8 min in Demo 2 (medians 28.1 and 25.0 min)—so the difference came from the synchronization change rather than the model. Demo 1's timed windows left substantial idle time: 26.6 min per scenario day under the stable 1-hour-freeze regime (scenario days 1–129) and 94.2 min under the stable 2-hour-freeze regime (day 157 onward, with a re-run transition spanning days 130–156), totaling 255 h over the campaign, a mean of 57.6 min per scenario day. Demo 2's event-driven resume cut residual idle time to 29.6 s per scenario day and 2.5 h over the campaign, eliminating 99.1% of idle time while preserving the same daily LIS–NOS-T exchange pattern. The 1-to-2-hour freeze extension itself traced to an operational trigger: LIS batch processing intermittently overran its 1-hour allocation, its smoothed processing time peaking at 67.1 min at

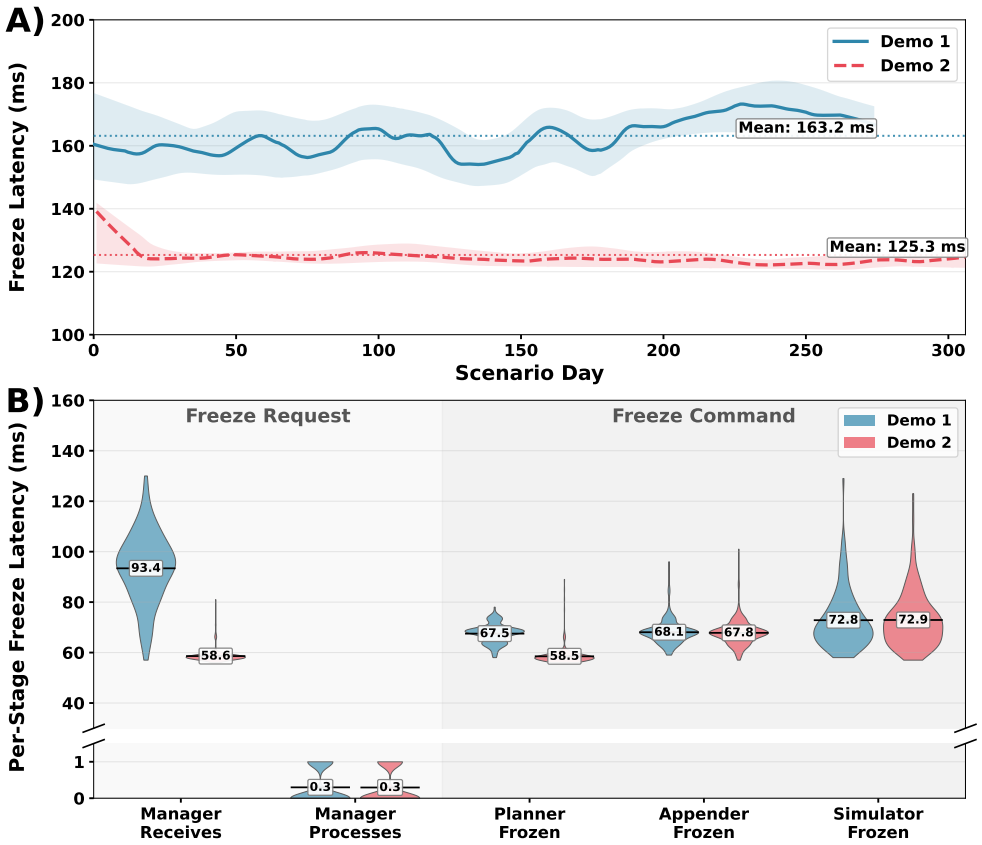


Fig. 8 Comparative scenario time freeze propagation across the Demo 1 and Demo 2 SOS campaigns. **(A)** End-to-end freeze propagation latency per freeze cycle, plotted against relative scenario day (day 1 = each campaign's October 1 start), for Demo 1 (blue) and Demo 2 (red). Trend lines show a LOWESS smoothing (Demo 1 solid, Demo 2 dashed), shaded bands span the interquartile range of the per-cycle values, and dotted horizontal lines mark the campaign mean. **(B)** Per-stage latency distributions (violins, with the mean annotated) decomposing each freeze into the request stage—Planner-to-Manager request transit (Manager Receives) and Manager internal processing (Manager Processes)—and the command stage—Manager-to-application broadcast reception at the Planner, Appender, and Simulator. The broken y-axis resolves the near-instant Manager-processing stage alongside the larger transit stages. Freeze propagation remained sub-second for all target applications in both campaigns, and the campaign-to-campaign reduction concentrated in request-to-Manager transit.

scenario day 141 (Figure 9A). Observing this overrun, we extended the freeze window to 2 hours and the NOS-T execution window from 20 to 24 min (72×→60×), producing the execution-window step at scenario day 159 (Figure 9B) and the idle increase between the two freeze regimes noted above (Figure 9C).

Independent S3 upload logs characterized the LIS–NOS-T handoff in both campaigns. In Demo 1, the mid-campaign freeze extension split the upload cadence into two regimes: mean LIS and Simulator upload intervals rose from 1.34–1.37 h under the 1 h freeze to 2.38 h under the 2 h freeze. The Simulator interval stayed near-deterministic

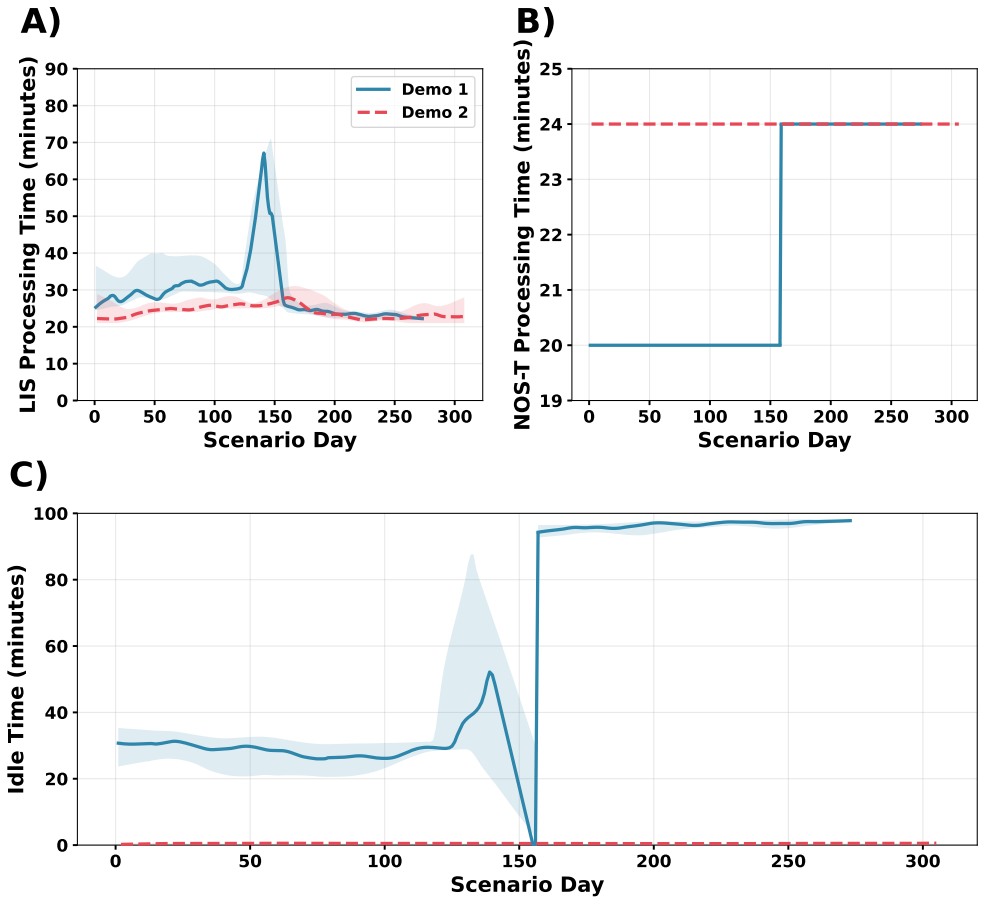


Fig. 9 LIS processing time, NOS-T processing time, and idle time across the Demo 1 and Demo 2 SOS campaigns, plotted against relative scenario day (day 1 = each campaign's October 1 start). All three panels summarize per-day values with a LOWESS trend (Demo 1 solid, Demo 2 dashed) and a shaded interquartile band. **(A)** LIS batch processing time per scenario day. **(B)** NOS-T processing time per scenario day, measured as the execution window from application start to freeze. It tracks the acceleration-set window (20 min at 72×, 24 min at 60×) and steps up at scenario day 159 with Demo 1's 72×→60× acceleration reduction. **(C)** Idle time per scenario day under Demo 1's fixed-duration freeze windows compared with Demo 2's event-driven resume. Demo 1's idle time steps up to the stable 2-hour-freeze regime following the acceleration change (scenario days 130–156 appear as a re-run transition), while Demo 2's residual idle time reflects only the LIS-to-Simulator handoff beyond the NOS-T execution window.

across the campaign, locked to the fixed-timer cycle, while the LIS interval varied more widely with supercomputer processing time. The LIS-to-Simulator gap—the NOS-T execution window (∼24 min) plus any idle wait while NOS-T remained frozen after LIS had finished—doubled from 0.87 h to 1.80 h under the longer freeze, the growth being idle time that accumulated above the execution floor (Supplementary Figure S8A). In Demo 2, the single event-driven configuration held the Simulator and LIS upload intervals at a steady 2.40 h, both varying with LIS processing time rather than a fixed

timer. The LIS-to-Simulator gap averaged 0.42 h (∼25 min), essentially the NOS-T execution window alone: event-driven resume fires the moment LIS uploads, leaving negligible idle and eliminating the wait that grew under Demo 1's fixed freeze (Supplementary Figure S9A). Manual restarts were required eight times in Demo 1 and five times in Demo 2—when LIS processing exceeded its allocated window (Demo 1), when application errors interrupted execution, or when prolonged NOS-T disconnections occurred—each producing an isolated multi-hour gap, excluded from the regime statistics above but retained in the upload timeline (Supplementary Figures S8B and S9B).

5.5 High-Reward Observation Selection

Across both campaigns, the Planner identified high-reward regions from efficiency layers derived from LIS SWE change, snow cover fraction, and surface temperature, and the Simulator recorded which simulated SnowGlobe observations intersecting those regions were observed. Demo 1 produced 1,122 high-reward SAR observations (4.1 per scenario day) and Demo 2 produced 4,584 (15.0 per scenario day) (Figure 10A). The higher Demo 2 rate follows a change made between the campaigns in how the Simulator tracked which requests remained eligible for observation, and the two campaigns reproduce different snow seasons, so matching counts were never expected. Neither difference is analyzed further here, since this paper evaluates the testbed's coordination performance rather than the observation yield the campaigns produced.

High-reward observations clustered during active snowmelt, peaking at 160 observations in May 2025 in Demo 1 and 543 observations in March 2019 in Demo 2. Observations were balanced across the five SnowGlobe satellites, each contributing 19.3–21.2% in Demo 1 and 19.5–20.2% in Demo 2. The daily observation series showed no sustained gaps in either campaign, confirming uninterrupted Planner–Simulator–LIS coordination (Figure 10).

Spatially, selection frequency increased with terrain elevation (Spearman $\rho = 0.71$ for Demo 1 and $\rho = 0.54$ for Demo 2, both $p < 0.001$), concentrating the most persistently selected cells over the high, mountainous western basin, where PMW retrievals saturate in deep snow (Tsang et al. 2022). Where and when to observe were nevertheless not fixed in advance. The Planner scores cells on SWE change, snow cover, and surface temperature and carries no notion of elevation, so when snow arrived on the eastern plains, the plains scored. In Demo 2, a record February snowfall over Nebraska raised plains SWE in LIS to a 35 mm plateau by early March, and the Planner moved its selections onto the plains for three weeks, its median selection elevation falling from about 1,900 m to near 1,000 m. It returned to the mountain front within days of the mid-March cyclone, whose rain and thaw melted the plains bare by the end of the month (Zou et al. 2025; Velásquez et al. 2023). Demo 1 traced a shallower excursion in December as its first plains snow arrived. In both campaigns the Simulator's observed cells followed the Planner's path at lower median elevation, and the cells last selected on the plains froze at their winter dates while the mountain-front cells stayed active into May (Supplementary Figure S10).

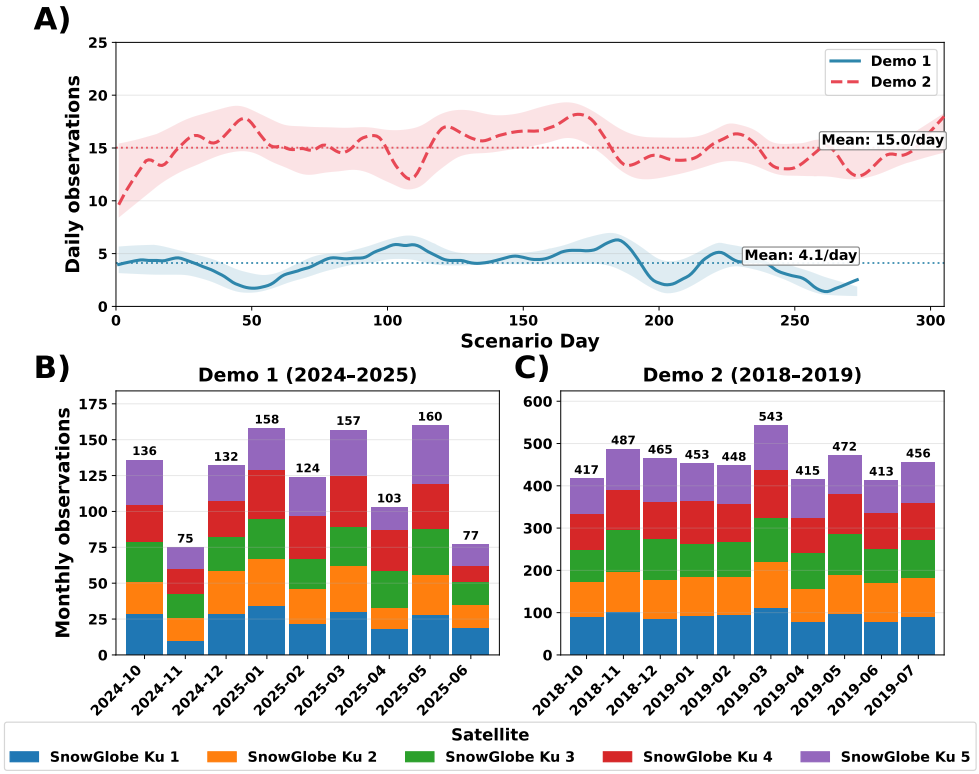


Fig. 10 High-reward SAR observations across both SOS campaigns—Demo 1 (2024–2025 season) and Demo 2 (2018–2019 season). **(A)** Daily observation counts per scenario day (day 1 = each season's October 1 start) for Demo 1 (blue, solid) and Demo 2 (red, dashed), each shown as a LOWESS trend with a shaded 25–75% interquartile band. Dotted horizontal lines mark the mean daily observation rate. **(B, C)** Monthly observations stratified by SnowGlobe satellite for Demo 1 and Demo 2, respectively. Observations are balanced across the five SnowGlobe satellites.

5.6 Validation Summary

Across 577 total scenario days, the two SOS campaigns collectively exercised all five architectural requirements (R1–R5) under realistic operational conditions. Two of them operated continuously in support rather than as measured outcomes: token-based authentication (R1) ran uninterrupted across both campaigns, while schema governance (R4) enforced validated message formats for observation requests, satellite state updates, and simulation results, preventing semantic mismatches among the four applications.

6 Discussion

The evolution of NOS-T across three versions demonstrates that grounding architectural change in empirically observed limitations, rather than anticipated requirements, yields a resilient and standards-aligned testbed without sacrificing conceptual integrity.

Each of the five drivers addressed in V2 and V3—identity and authorization, reliable and persistent messaging, session resilience, schema governance, and coordinated scenario time control—originated in a constraint encountered during V1 operations rather than in a specification drafted in advance (Chell et al. 2023). Staging the infrastructure hardening of V2 ahead of the temporal coordination of V3 allowed each layer to be validated against operational evidence before the next was introduced, thereby preserving a coherent architecture as capability accrued. We regard this challenge-driven refinement as the principal architectural lesson of the work, transferable to other long-lived research platforms that must mature under continuous use.

Notably, the loosely coupled integration of LIS with NOS-T, in which a land-surface model executing on a firewalled high-performance computing system exchanged data through cloud object storage under coordinated scenario time control, illustrates a pattern that generalizes beyond the snow domain. Coupling heterogeneous, independently managed models through a shared messaging fabric and a minimal set of interface conventions, rather than through direct integration, lets each contributor retain its native computing environment while participating in a coordinated scenario. This low-friction coupling reflects the testbed's deliberate middle ground between ad hoc integration and full simulation-standard compliance, which adopts the proven message semantics, federation-coordination principles, and systematic engineering process of DIS, HLA, and DSEEP while trading their comprehensive interoperability guarantees for the integration ease that early-stage concept development rewards (IEEE 2012, 2023, 2025a).

The two SOS campaigns serve as an extended example of the platform's computational capabilities under realistic, long-duration operation. Across both, reliable messaging held message loss at zero, automated recovery restored every interrupted connection, and coordinated freezes propagated in under a second. Demo 2's event-driven resume additionally eliminated the idle waiting imposed by Demo 1's fixed-duration freezes (Supplementary Figures S8 and S9). Because NOS-T supports both modes, each deployment can resolve this trade-off for itself: fixed-interval freezes configure simply but leave idle time when external processing finishes early, whereas event-driven resume removes that idle at the cost of an event-trigger path. Importantly, the campaigns produced more than system metrics: the coordinated multi-sensor SAR–PMW workflow selected high-reward observations that, in a fully taskable version of the campaign—with SnowGlobe or other constellations—may help improve SWE estimation. Independent replication across two distinct snow seasons strengthens confidence that these results reflect the platform's capabilities rather than the particulars of a single scenario.

Neither latency measure revealed a coordination bottleneck. Freeze propagation averaged 163.2 ms in Demo 1 and 125.3 ms in Demo 2, the campaign-to-campaign difference concentrated in request-to-Manager transit rather than Manager processing, which held at 0.3 ms in both. Freeze latency is therefore governed by broker placement—a wide-area hop between the on-premises applications and the cloud broker—rather than by coordination overhead, and stays negligible against LIS processing windows measured in tens of minutes. Message wall-clock latency likewise held a sub-second baseline broken only by transient spikes (Figure 6): a 15-second band

as a reconnecting client re-sends the messages it queued locally during the outage, and a 6–8-second band from partial re-sends and from the Planner's compute-heavy per-cycle work—retrieving LIS layers from S3 and solving the observation-selection optimization—briefly stalling its status-reporting loop. Because the NOS-T Tools library re-sent every buffered message on reconnect, both bands are delay, not loss. Latency-sensitive deployments could shrink both by co-locating the broker with participating applications and offloading synchronous per-cycle computation to worker threads.

The present campaigns selected observations from SnowGlobe's pre-simulated coverage rather than commanding a constellation in real time, yet they exercise end to end the planning, simulation, and time-coordination loop that genuine dynamic tasking requires. In each cycle the Planner identified high-reward regions from evolving LIS land-surface state, the Simulator resolved which modeled observations intersected them, and the closed loop returned those observations for assimilation under sub-second coordinated freezes. A truly taskable system would close this same loop against a live constellation, adjusting acquisitions in near-real time as snowpack conditions evolve. Such dynamic tasking would concentrate a limited SAR duty cycle on the regions where each observation yields the greatest scientific return—deep mountain snowpack, rapid melt onset, and terrain where passive microwave saturates—rather than on uniform or pre-planned coverage, thereby improving SWE estimation per observation while avoiding acquisitions over low-information areas. These campaigns demonstrate the coordination substrate on which such responsiveness depends.

Beyond the reliance on pre-simulated coverage, two further constraints bound the present results and indicate where the platform requires further maturation. First, schema governance enforced message validation within the SOS campaigns, but its catalog is not yet comprehensive across domains. NOS-T is designed to generalize across DSM concepts and scientific domains, so integrating mission concepts beyond SOS is the natural next step, and that broader experience will consolidate a unified schema catalog spanning projects. Second, session resilience was validated against the disconnections that arose naturally during the campaigns rather than through systematic fault injection. Introducing controlled fault injection would let NOS-T emulate the operational disruptions characteristic of fielded missions, which early concept-stage demonstrations do not naturally exhibit. Neither constraint affected the integrity of the reported campaigns, each of which recovered from every interruption and completed without critical failure.

The evolved platform also lowers the barrier to participation for research teams. The NOS-T Tools library provides Python abstractions through which a contributor builds an application and connects it to the testbed without reconstructing the underlying messaging, identity, and time-management infrastructure. A single declarative configuration file specifies the connection and scenario parameters, and the resulting application can be containerized for portable deployment. Such accessibility matters most to organizations such as NASA, where candidate constellation architectures, sensing modalities, and coordination schemes must be repeatedly prototyped before

any commitment to development. By lowering the engineering effort required to assemble and run a credible distributed simulation, NOS-T lets mission designers compare alternatives and set aside unpromising ones early, when the cost of change is lowest.

Sustained adoption ultimately depends on the surrounding community as much as on the architecture. By aligning with established simulation standards rather than diverging from them, NOS-T lets applications developed on the platform retain migration paths toward more comprehensive frameworks as concepts mature. Open-sourcing the tools library, together with a collaborative model that minimizes the cost of entry and specifies clear conformance conventions, treats interoperability as a network good whose value grows with each additional participant. Fostering the incentives for institutions to contribute, rather than mandating their participation, is consistent with the voluntary, owner-free character of the collaborative systems NOS-T is designed to support (Maier 1998; Selva et al. 2017).

7 Conclusions

The NOS-T platform has evolved through challenge-driven refinement from an operational baseline into a resilient, standards-aligned testbed for distributed space mission prototyping. V2 hardened the platform's identity, messaging, session, and schema layers in response to four operational constraints, and V3 added coordinated scenario time control through programmatic freeze, resume, and time-scale orchestration. Two independent SOS campaigns, spanning 577 scenario days and closing the loop with an external land-surface model, validated the evolved architecture at Technology Readiness Level (TRL) 6—a system prototype demonstrated in a relevant environment.

These capabilities address the gap that motivated the work: early-stage DSM concept development has lacked an accessible testbed that balances architectural discipline, low integration friction, and alignment with simulation standards. By providing a lightweight, cloud-native platform that multi-institutional teams can adopt and extend, NOS-T offers a route to evaluating novel observing strategies before committing to costly development. Separating these five concerns makes the architecture's benefits generalizable well beyond the SWE application that exercised them here.

Future work will extend the platform from observation selection toward closed-loop dynamic tasking—in which observing systems adapt their acquisitions to evolving environmental conditions in near-real time—and will continue to harden its robustness against the realistic disruptions that operational missions encounter. The publicly available tools library, coupled with the interfaces established across V2 and V3, positions NOS-T for adoption by a growing community across Earth observation disciplines in Earth science and beyond, accelerating the transition of novel observing strategies from concept to capability.

Supplementary information. Supplementary Figures S1–S10 and Supplementary Table S1 accompany this article.

Acknowledgments. This research has made use of the NASA Science Cloud, which is a service of the Computational & Information Sciences and Technology Office at the NASA Goddard Space Flight Center.

Statements and Declarations

Competing interests. The authors declare no competing interests.

Funding. This material is based on work supported, in whole or in part, by the NASA Earth Science Technology Office under the New Observing Strategies Testbed (NOS-T) Development and Demonstration Grant No. 80NSSC24K0875.

Ethics approval and consent to participate. Not applicable.

Consent for publication. Not applicable.

Data availability. The datasets generated and analysed during the current study are available from the corresponding author on reasonable request.

Materials availability. Not applicable.

Code availability. The NOS-T Tools library is publicly available at https://github.com/code-lab-org/nost-tools.

Author contributions. Emmanuel M. Gonzalez: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization. Hadis Banafsheh: Methodology, Software, Validation, Writing – review & editing. Divya Ramachandran: Methodology, Software, Validation, Investigation, Writing – review & editing. Melissa L. Wrzesien: Software, Investigation, Resources, Data curation, Writing – review & editing. Sujay V. Kumar: Supervision, Writing – review & editing. Carrie M. Vuyovich: Supervision, Writing – review & editing. Paul T. Grogan: Supervision, Project administration, Funding acquisition, Writing – review & editing. All authors read and approved the final manuscript.

References

Banafsheh H, Wrzesien ML, Ramachandran D, et al (2026) A dynamic reward-based optimization framework for satellite task planning in Earth observation: Application to snow mass observation. SSRN preprint 6142417, https://doi.org/10.2139/ssrn.6142417

Chell B, LeVine MJ, Capra L, et al (2023) New observing strategies testbed: A digital prototyping platform for distributed space missions. Systems Engineering 26(5):519–530. https://doi.org/10.1002/sys.21672

Cho E, Vuyovich CM, Kumar SV, et al (2023) Evaluating the utility of active microwave observations as a snow mission concept using observing system simulation experiments. The Cryosphere 17(9):3915–3931. https://doi.org/10.5194/tc-17-3915-2023

Geletko DM, Grubb MD, Lucas JP, et al (2019) NASA operational simulator for small satellites (NOS3): The STF-1 CubeSat case study. arXiv preprint arXiv:1901.07583, https://doi.org/10.48550/arXiv.1901.07583

Gómez P, Östman J, Magal Shreenath V, et al (2024) PASEOS simulates the environment for operating multiple spacecraft. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 17:17398–17411. https://doi.org/10.1109/JSTARS.2024.3445506

Hale K, Meyer J, Tarricone J, et al (2025) What's next for snow: Insights from the NASA Terrestrial Hydrology Program community snow meeting. Earth's Future 13(12):e2025EF006460. https://doi.org/10.1029/2025EF006460

Hardy TD, Palmintier B, Top PL, et al (2024) HELICS: A co-simulation framework for scalable multi-domain modeling and analysis. IEEE Access 12:24325–24347. https://doi.org/10.1109/ACCESS.2024.3363615

Hill C, DeLuca C, Balaji V, et al (2004) The architecture of the earth system modeling framework. Computing in Science & Engineering 6(1):18–28. https://doi.org/10.1109/MCISE.2004.1255817

IEEE (2012) IEEE Std 1278.1-2012: Standard for distributed interactive simulation—application protocols. Institute of Electrical and Electronics Engineers, New York, NY, USA, https://doi.org/10.1109/IEEESTD.2012.6387564, revision of IEEE Std 1278.1-1995

IEEE (2015) IEEE Std 1278.2-2015: Standard for distributed interactive simulation (dis)—communication services and profiles. Institute of Electrical and Electronics Engineers, New York, NY, USA, https://doi.org/10.1109/IEEESTD.2015.7459689, revision of IEEE Std 1278.2-1995

IEEE (2023) IEEE Std 1730.1-2023: Recommended practice for distributed simulation engineering and execution process multi-architecture overlay (dmao). Institute of Electrical and Electronics Engineers, New York, NY, USA, https://doi.org/10.1109/IEEESTD.2024.10439055, revision of IEEE Std 1730.1-2013

IEEE (2025a) IEEE Std 1516.1-2025: Standard for modeling and simulation (m&s) high level architecture (hla)—federate interface specification. Institute of Electrical and Electronics Engineers, New York, NY, USA, https://doi.org/10.1109/IEEESTD.2025.11134777, revision of IEEE Std 1516.1-2010

IEEE (2025b) IEEE Std 1516.2-2025: Standard for modeling and simulation (m&s) high level architecture (hla)—object model template (omt) specification. Institute of Electrical and Electronics Engineers, New York, NY, USA, https://doi.org/10.1109/IEEESTD.2025.11112812, revision of IEEE Std 1516.2-2010

Kumar SV, Peters-Lidard CD, Tian Y, et al (2006) Land information system: An interoperable framework for high resolution land surface modeling. Environmental Modelling & Software 21(10):1402–1415. https://doi.org/10.1016/j.envsoft.2005.07.004

Kumar SV, Peters-Lidard CD, Tian Y, et al (2008) An integrated hydrologic modeling and data assimilation framework. Computer 41(12):52–59. https://doi.org/10.1109/MC.2008.475

Lamport L (1978) Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7):558–565. https://doi.org/10.1145/359545.359563

Le Moigne J, Little MM, Cole MC (2019) New observing strategy (NOS) for future Earth science missions. In: IGARSS 2019 – 2019 IEEE International Geoscience and Remote Sensing Symposium. IEEE, pp 5285–5288, https://doi.org/10.1109/IGARSS.2019.8898096

Maier MW (1998) Architecting principles for systems-of-systems. Systems Engineering 1(4):267–284. https://doi.org/10.1002/(SICI)1520-6858(1998)1:4⟨267::AID-SYS3⟩3.0.CO;2-D

Martinez Marrero L, Merlano-Duncan JC, Querol J, et al (2022) Architectures and synchronization techniques for distributed satellite systems: A survey. IEEE Access 10:45375–45409. https://doi.org/10.1109/ACCESS.2022.3169499

Miller DC, Thorpe JA (1995) SIMNET: The advent of simulator networking. Proceedings of the IEEE 83(8):1114–1123. https://doi.org/10.1109/5.400452

NASA Goddard Space Flight Center (2024) Goddard mission services evolution center (GMSEC) overview. NASA Goddard Space Flight Center, Software Engineering Division, briefing, July 2024, URL https://www.nasa.gov/goddard/gmsec/

Quigley M, Conley K, Gerkey B, et al (2009) ROS: An open-source robot operating system. In: ICRA Workshop on Open Source Software, URL https://api.semanticscholar.org/CorpusID:6324125

Selva D, Golkar A, Korobova O, et al (2017) Distributed Earth satellite systems: What is needed to move forward? Journal of Aerospace Information Systems 14(8):412–438. https://doi.org/10.2514/1.I010497

Tsang L, Durand M, Derksen C, et al (2022) Review article: Global monitoring of snow water equivalent using high-frequency radar remote sensing. The Cryosphere 16(9):3531–3573. https://doi.org/10.5194/tc-16-3531-2022

Velásquez N, Quintero F, Rasiya Koya S, et al (2023) Snow-detonated floods: Assessment of the U.S. Midwest March 2019 event. Journal of Hydrology: Regional Studies 47:101387. https://doi.org/10.1016/j.ejrh.2023.101387

Wrzesien ML, Kumar S, Vuyovich C, et al (2022) Development of a "nature run" for observing system simulation experiments (OSSEs) for snow mission development. Journal of Hydrometeorology 23(3):351–375. https://doi.org/10.1175/JHM-D-21-0071.1

Zou X, Cordeira JM, Bartlett SM, et al (2025) A case study of an exceptional atmospheric river and explosively deepening cyclone over the US central plains in March 2019. Journal of Geophysical Research: Atmospheres 130:e2024JD042309. https://doi.org/10.1029/2024JD042309

Supplementary Information

Novel Observing Strategies Testbed: A Lightweight, Collaborative Platform for Prototyping Earth Observing Missions

Earth Science Informatics

Emmanuel M. Gonzalez[1], Hadis Banafsheh[1], Divya Ramachandran[1], Melissa L. Wrzesien[2,3], Sujay V. Kumar[3], Carrie M. Vuyovich[3], Paul T. Grogan[1]

1 School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ 85281, USA
2 Earth System Science Interdisciplinary Center, University of Maryland, College Park, MD 20740, USA
3 Hydrological Sciences Laboratory, NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA

Corresponding author: Paul T. Grogan
Email: Paul.Grogan@asu.edu
Orcid ID: https://orcid.org/0000-0001-8986-4806

Supplementary Figures

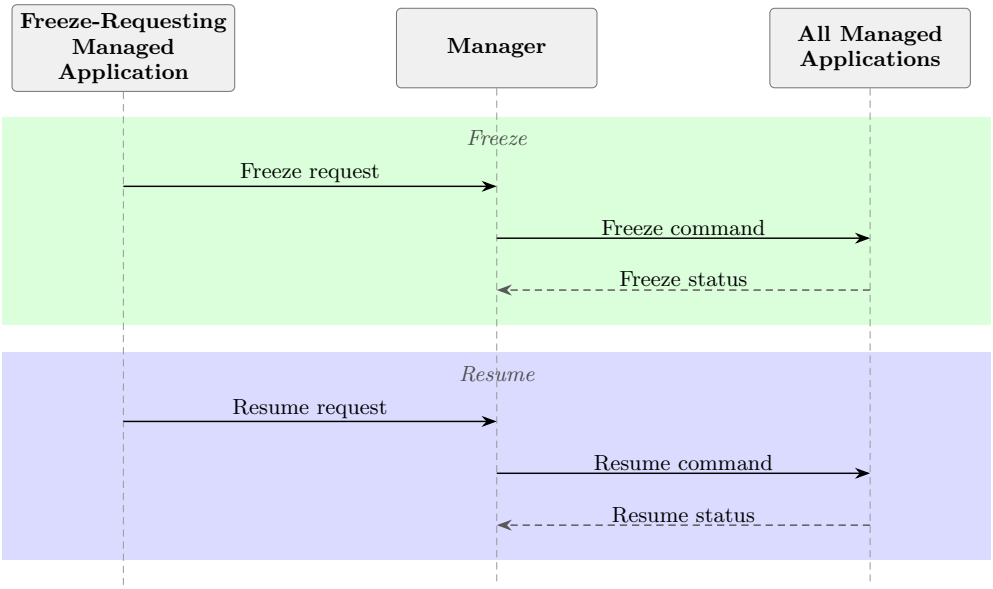


Fig. S1 Coordinated scenario time freeze and resume in NOS-T V3. A managed application requests a freeze, the Manager arbitrates the request and broadcasts a freeze command to all managed applications, and each acknowledges with a freeze status. Resume follows the same pattern, so every application freezes and resumes at the same scenario time. The requesting application is itself one of the managed applications and so also receives the broadcast command. Dashed arrows denote acknowledgments.

```yaml
info:
  title: Novel Observing Strategies Testbed (NOS-T) YAML Configuration
  version: '1.0.0'
  description: Version-controlled configuration document for RabbitMQ event
    broker with Keycloak authentication within NOS-T
servers:
  rabbitmq:
    keycloak_authentication: False
    host: "localhost"
    port: 5672
    tls: False
    virtual_host: "/"
    message_expiration: "60000"
    heartbeat: 600
    blocked_connection_timeout: 300
execution:
  general:
    prefix: greenfield
    wallclock_offset_refresh_interval: 60
    ntp_host: "time.nist.gov"
  application:
    shut_down_when_terminated: True
  manager:
    sim_start_time: "2019-12-31T23:59:59+00:00"
    sim_stop_time: "2020-01-01T10:00:00+00:00"
    time_step: "0:00:01"
    time_scale_factor: 60
    time_status_step: "0:00:01"
    command_lead: "0:00:05"
    required_apps: [manager, fire, constellation, ground]
    init_retry_delay_s: 5
    init_max_retry: 5
  managed_applications:
    constellation:
      time_scale_factor: 60
      time_step: "0:00:01"
      time_status_step: "0:00:10"
      set_offset: True
      shut_down_when_terminated: True
      manager_app_name: "manager"
      configuration_parameters:
        FIELD_OF_REGARD: [110, 110, 112.56, 112.56, 20.6, 20.6]
    # fire and ground follow the same pattern
```

Fig. S2 Abridged NOS-T Tools YAML configuration, taken from the FireSat+ example. A single file defines the broker and identity server settings, the execution parameters including scenario start and stop times and the time-scale factor, and per-application parameters such as the constellation field of regard. Credentials are not stored here but are read from a local environment file or from environment variables. The `fire` and `ground` managed applications are omitted, each following the same pattern as `constellation`.

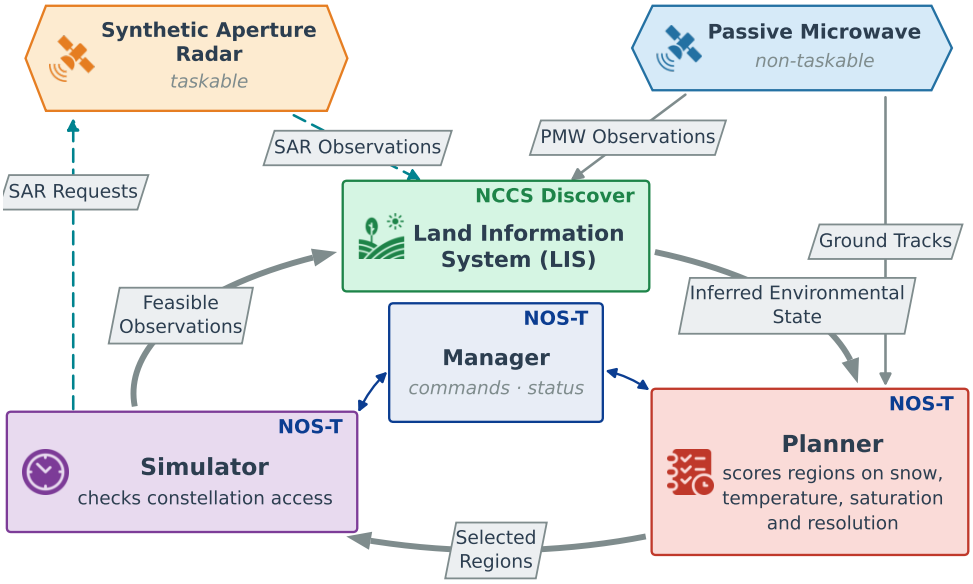


Fig. S3 Intended SOS operational concept, drawn as a closed loop. NASA's LIS, running on NCCS Discover, hands its inferred environmental state to the NOS-T Planner, which scores regions on snow, temperature, sensor saturation, and resolution and passes the selected regions to the NOS-T Simulator. The Simulator checks constellation access and returns the feasible observations to LIS for assimilation. The NOS-T Manager exchanges commands and status with the Planner and the Simulator. Non-taskable PMW observations feed LIS continuously and PMW ground tracks feed the Planner. Dashed legs mark the SAR requests and observations that a taskable constellation would carry. The two campaigns reported here exercised the Planner–Simulator selection loop against *pre-simulated* SnowGlobe coverage rather than commanding a live, taskable constellation.

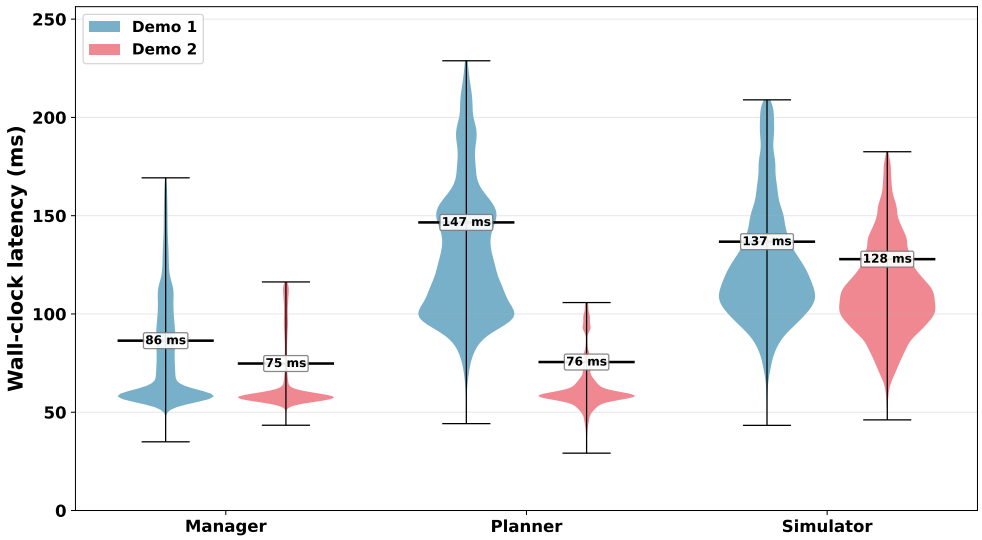


Fig. S4 Distribution of wall-clock application latency for the Manager, Planner, and Simulator across the two SOS campaigns—Demo 1 (273-day 2024–2025 season) and Demo 2 (304-day 2018–2019 season)—shown as violin plots (distribution body with black min–max extrema bars) with the mean marked as a black line and labeled. Reconnection-spike outliers are omitted for legibility and shown over time in Figure 6. The managed Planner and Simulator sit above the Manager, and the Planner's latency fell markedly between campaigns.



Fig. S5 Per-application connection recovery intervals for the Manager, Planner, and Simulator across both SOS campaigns—Demo 1 (blue) and Demo 2 (red). Violin plots show the recovery-interval distribution for each application, with individual reconnect episodes overlaid as jittered points and the group mean marked (black line, labeled). The dashed horizontal line marks the configured 15 s reconnection delay. Every application recovered close to that delay in both campaigns.

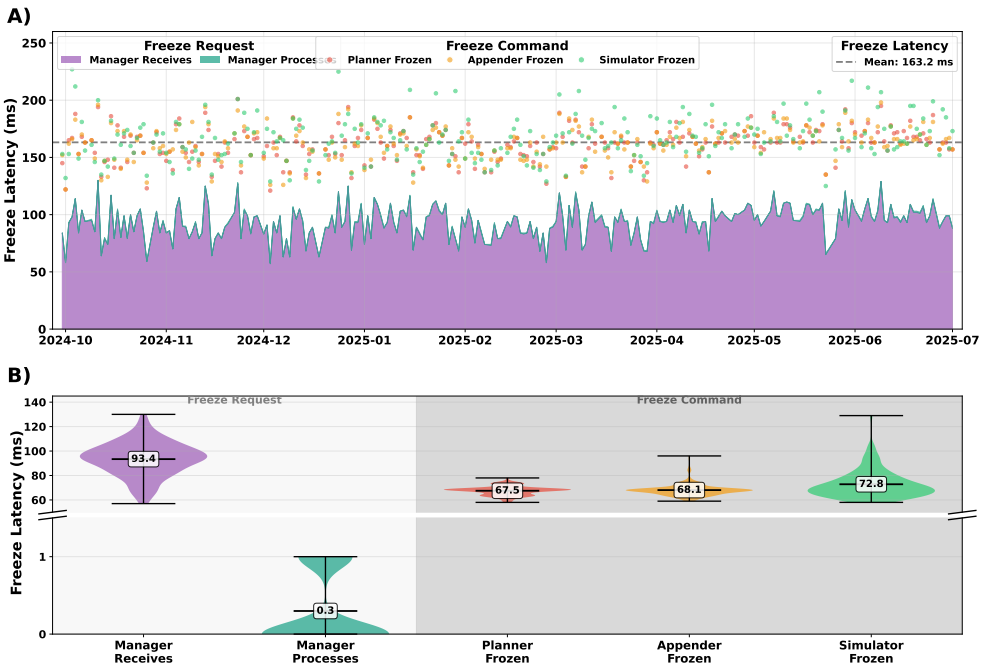


Fig. S6 Scenario time freeze propagation latency analysis across distributed applications during the 273-day Demo 1 demonstration. **(A)** Event timing breakdown for the Demo 1 freeze cycles, showing stacked latency components from Planner freeze request through Manager receipt and processing and broadcast receipt by Planner, Appender, and Simulator. Gray dashed line indicates mean end-to-end propagation latency. **(B)** Latency distribution by event step, presented as violin plots with embedded mean values.

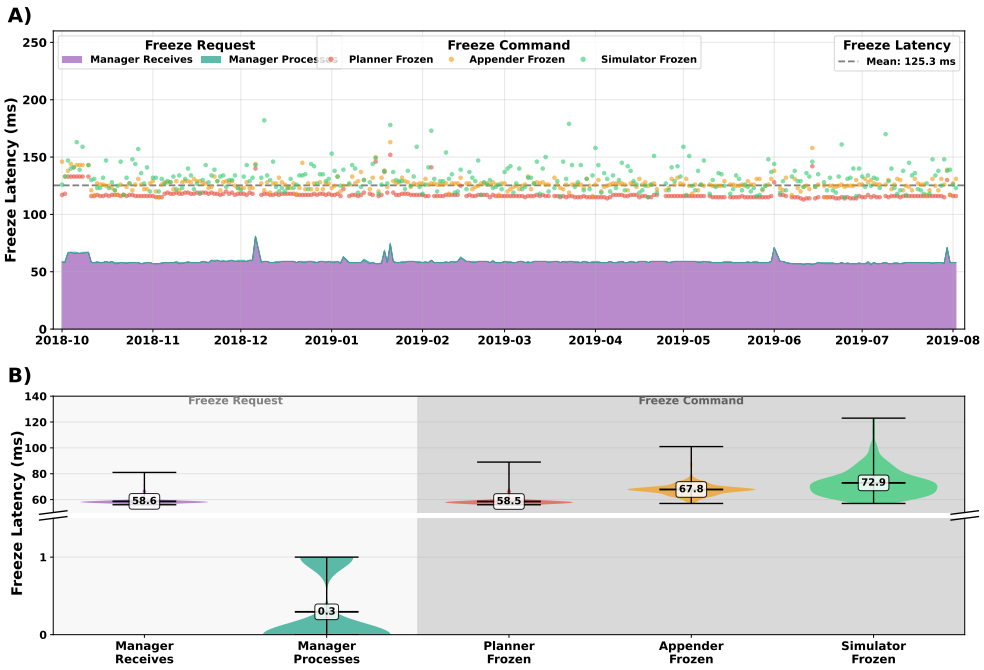


Fig. S7 Scenario time freeze propagation latency analysis across distributed applications during the 304-day Demo 2 demonstration. **(A)** Event timing breakdown for the Demo 2 freeze cycles, showing stacked latency components from Planner freeze request through Manager receipt and processing and broadcast receipt by Planner, Appender, and Simulator. Gray dashed line indicates mean end-to-end propagation latency. **(B)** Latency distribution by event step, presented as violin plots with embedded mean values.

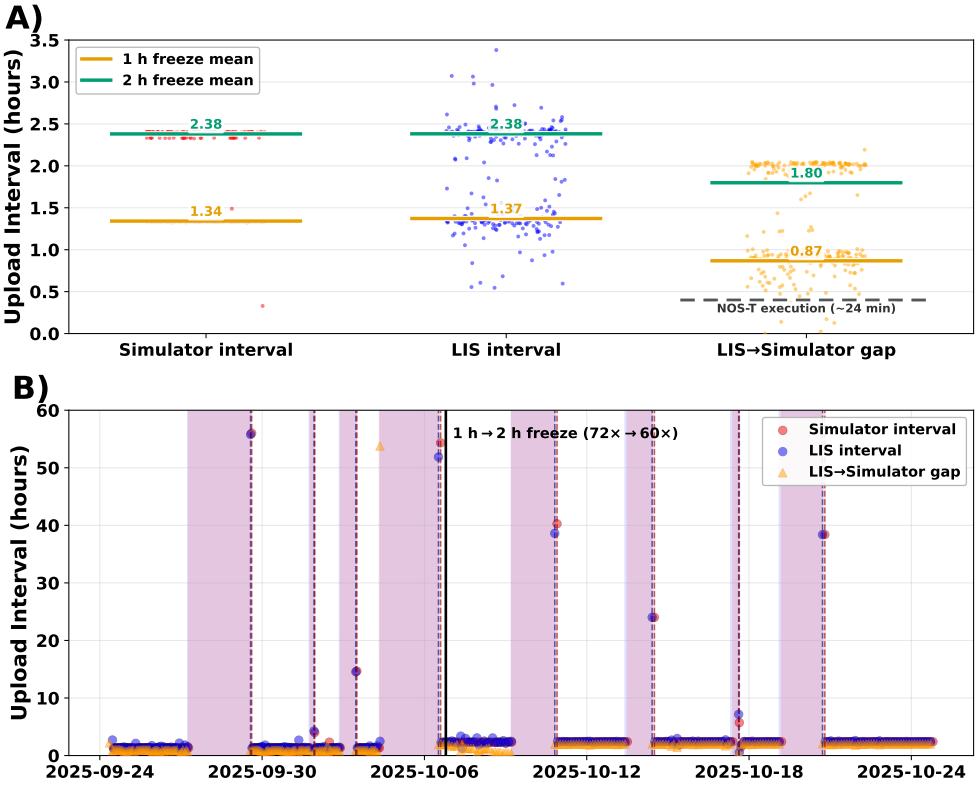


Fig. S8 LIS–Simulator upload timing via S3 during Demo 1, split by freeze regime. **(A)** Distributions of the Simulator upload interval, LIS upload interval, and LIS→Simulator handoff gap. Horizontal lines mark the per-regime mean (amber for the 1 h freeze, teal for the 2 h freeze), computed after excluding restart gaps. The Simulator interval is near-deterministic, locked to the fixed-timer cycle, while the LIS interval varies with supercomputer processing time. The mid-campaign configuration change—freeze window 1 h→2 h and 72×→60× acceleration—stepped both upload intervals upward. The LIS→Simulator gap comprises the NOS-T execution window (dashed reference line, ∼24 min) plus any idle wait while NOS-T remained frozen after LIS had finished. The excess above the line is idle, which grew when the freeze was extended. **(B)** The same intervals over wall-clock time, with the configuration-change instant marked (solid black line) and manual restarts—triggered when the timed NOS-T or LIS windows were not met—shown as shaded bands and dashed vertical lines (red for Simulator, blue for LIS).

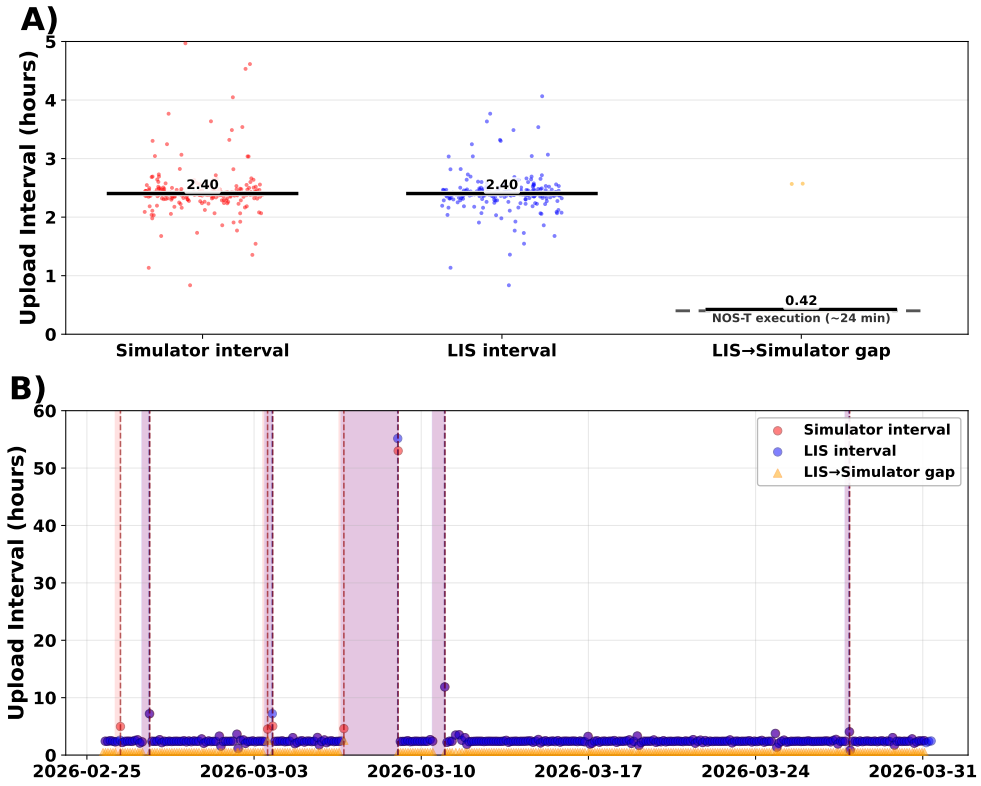


Fig. S9 LIS–Simulator upload timing via S3 during Demo 2, under the single event-driven configuration. **(A)** Distributions of the Simulator upload interval, LIS upload interval, and LIS→Simulator handoff gap. Horizontal lines mark the mean (restart gaps excluded). Unlike Demo 1, the LIS→Simulator gap sits on the NOS-T execution reference line (∼24 min): event-driven resume fires the moment LIS uploads, so the gap is essentially the execution window with negligible idle, eliminating the idle wait that grew with Demo 1's fixed freeze. **(B)** The same intervals over wall-clock time, with manual restarts shown as shaded bands and dashed vertical lines (red for Simulator, blue for LIS).

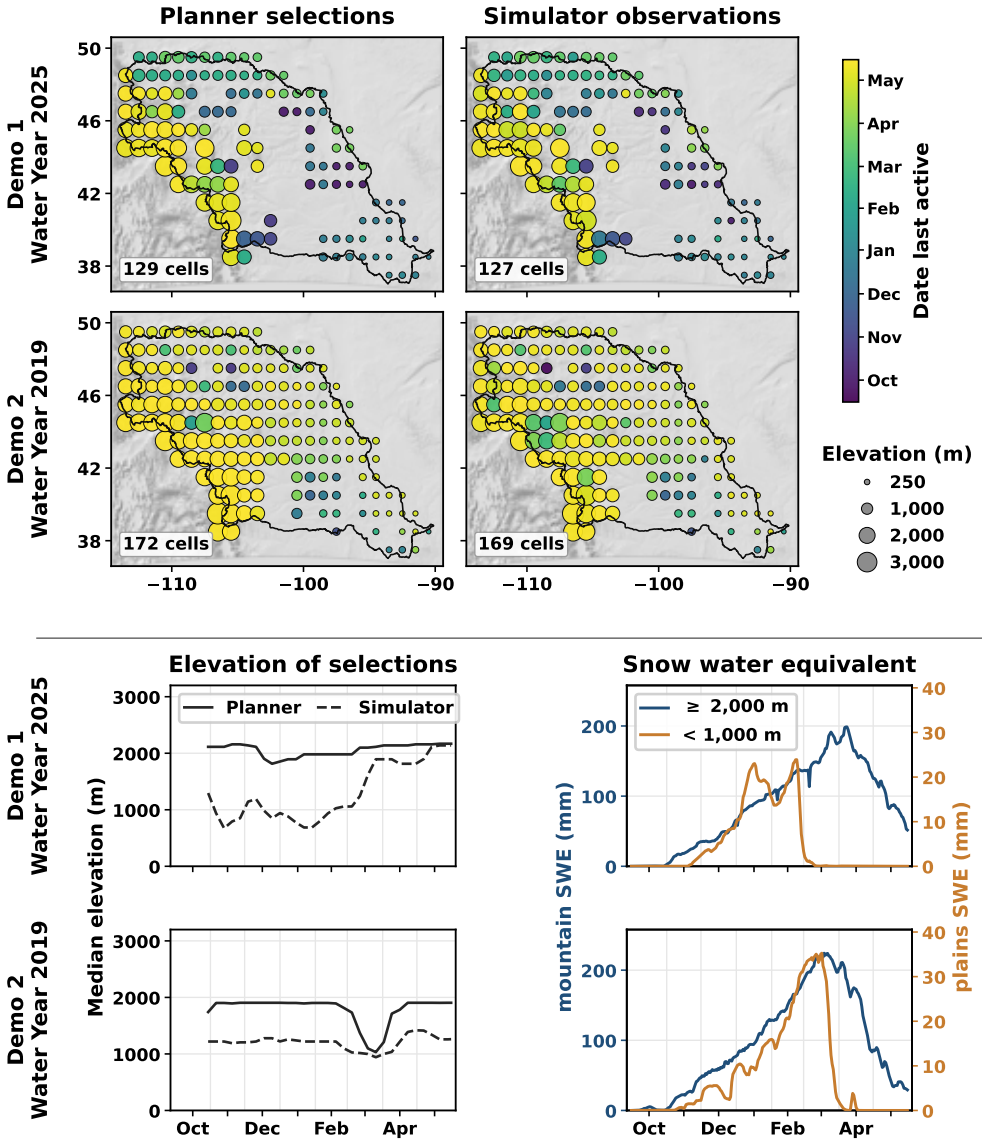


Fig. S10 Planner selections and Simulator observations across both SOS campaigns, Demo 1 (2024–2025 season) and Demo 2 (2018–2019 season). **Upper block:** 1° grid cells selected by the Planner and observed by the Simulator, each cell colored by the last scenario day it was active and sized by its elevation, with the cell count given in each panel. The shaded background is terrain elevation and the black line traces the basin boundary. **Lower block:** for the same campaign rows, the median elevation of the cells active in a trailing 30-day window, Planner solid and Simulator dashed, and the LIS snow water equivalent averaged over cells at or above 2,000 m and below 1,000 m. Each season spans October 1 to May 31.

Supplementary Tables

Table S1 Detailed architectural evolution across five components in NOS-T versions 1–3.

Component	V1	V2	V3
Identity & Authorization	Static Solace PubSub+ accounts with IP whitelisting on EC2; no role granularity or audit trails	Federated, token-based IAM provided by Keycloak with role/realm scoping and multi-factor authentication support	–
Reliable & Persistent Messaging	Best-effort MQTT over Solace PubSub+ with limited delivery guarantees and message persistence	AMQP over RabbitMQ with message acknowledgments, durable queues and exchanges, and delivery guarantees	–
Session Resilience	Manual intervention required for broker connection losses, and no state re-synchronization	Automated broker reconnection, heartbeat monitoring, and state re-synchronization	–
Message Schema & Configuration Governance	Ad hoc topic and payload conventions without versioning or validation; parameters defined in code	Schema catalog with validation hooks and change policies; parameters defined in version-controlled YAML configuration files; catalog continues to expand as new projects are integrated	–
Coordinated Scenario Time Control	Fixed/coarse scenario time control; no freeze/resume or fine-grained control	–	Programmatic, scenario-aware orchestration of scenario time freezes (timed or indefinite) and time-scale updates; in-application event-driven control