Development of a High-Performance, Heterogenous, Scalable Test-Bed for Distributed Spacecraft

Caleb Adams
NASA Ames Research Center
Moffet Field, CA
caleb.a.adams@nasa.gov

Brian Kempa
NASA Ames Research Center
Moffet Field, CA
brian.c.kempa@nasa.gov

Walter Vaughan
NASA Ames Research Center
Moffet Field, CA
walter.b.vaughan@nasa.gov

Nicholas Cramer
NASA Ames Research Center
Moffet Field, CA
nicholas.b.cramer@nasa.gov

Abstract— **To address modern mission challenges, high-performance computation devices (such as SoCs, GP-GPUs, FPGAs/ASICs, and neuromorphic processors) have become increasingly common in spacecraft while simultaneously, satellite constellation size continues to grow. The matched increase in spacecraft capability and cardinality requires concepts of operation that embrace a level of autonomy traditionally avoided in the space domain while tackling the additional complexity and uncertainty of controlling distributed systems. Development of these distributed autonomy capabilities necessitates multi-agent testing at scale for Validation & Verification (V&V) since the group's behavior is inextricably linked to the interactions among the individuals. Under this paradigm, characterizing emergent behavior in large, computationally-intensive satellite networks is paramount to mission success.**

The Distributed Spacecraft Autonomy (DSA) project at NASA Ames Research Center is maturing technology required for larger, autonomous constellations through orbital deployments and simulation studies. Previous work detailed the DSA simulation and test framework, capable of developing, testing, and verifying a four-spacecraft mission in a single developer environment using open-source software. DSA technology is also applied to a Lunar Position Navigation and Timing (LPNT) scenario with 100 spacecraft to further advance the state-of-the-art. Whereas the flight test hardware provided higher fidelity testing of the four-node DSA application before flight, no equivalent set of 100 engineering units exists for these large theoretical swarms, and the existing test framework falls short of the required number of participating agents by an order of magnitude. A higher fidelity test platform capable of scaling to 100 independent agents is required to support the development and characterization of distributed autonomy in large swarms of spacecraft.

DSA is building a heterogenous, many-node, Processor-in-the-Loop (PiL) testbed to aid the development and verification of scalable distributed autonomy capabilities for multi-spacecraft missions, including LPNT, heliophysics, space situational awareness, and collision avoidance. Our three rigorously selected platforms represent a range of computationally-intensive processors. Of the 100 PiLs, 85% are high-performance nodes, 40% are real flight hardware engineering units, and all have been deployed to orbit. Here, we detail the design, implementation, and capability of DSA's 100 processor-in-the-loop scalability testbed, the Distributed Intelligent Spacecraft Simulation Test RACK (DISSTRACK).

TABLE OF CONTENTS

978-1-6654-9032-0/23/$31.00 ©2023 IEEE

1. INTRODUCTION

The Distributed Spacecraft Autonomy (DSA) project at NASA's Ames Research Center is maturing technology necessary for autonomous decision-making by distributed space systems through simulation studies and orbital deployments. The most impactful demonstrations of DSA capability are onboard the Starling 1.0 satellite swarm and the 100 spacecraft lunar navigation study. NASA's Starling mission hosts several technologies for cooperative groups of spacecraft – also known as distributed missions, clusters, or swarms [1]. DSA's primary flight demonstration showcases collaborative resource allocation for multipoint science data collection with several small spacecraft as a payload on NASA's Starling 1.0 satellites [2]. The Starling 1.0 technology demonstration raises the Technology Readiness Level (TRL) of the DSA flight software from 3 (proof of concept) to 9 (flight proven). However, it is limited to four spacecraft. A scenario with 100 spacecraft opportunistically providing Position, Navigation, and Timing (PNT) service to the lunar surface is studied to evaluate distributed autonomy at scale. Initially, numerical simulation with simplified models and idealized communications assumptions validated the algorithm design while bounding expected performance [3, 4]. Guided by the preliminary simulation results, flight-grade DSA software was derived from the Starling 1.0 mission application, which implements the Lunar PNT (LPNT) scenario.

DSA has developed a framework for testing distributed behavior of flight software by leveraging free and open-source Linux container tools to run replicated instances of flight software on a single host computer. Linux containers, managed through Docker tools, create the isolated execution environment expected and often required by flight software without incurring the overhead of more conventional isolation approaches, such as machine virtualization (VMs). In addition to creating, starting, and stopping containers, our test infrastructure includes the ability to supply artificial sensor data, send commands, collect telemetry, measure network utilization, and introduce delay, loss, and bandwidth restrictions

to individual containers' network connections. These features are leveraged in test suites which verify requirements tied to the distributed behavior of the flight software, which we call *swarm tests*. For more general mission requirements, we define specific mission scenarios with scripted sensor inputs, spacecraft state, command sequences, and expected results for each spacecraft, which we call *scenario tests*. We refer to this test infrastructure collectively as the *swarm test framework*.

In this paper, we describe the scientific and technical motivators behind our testbed as well as details of its implementation and capabilities thus far. First, we examine the requirements of testing distributed behavior in Section 2. We then describe the design and infrastructure to address these requirements in Sections 3 and 4 and present preliminary results in Sections 5 and 6. Finally, we conclude with an overview of current and future work for this project in Section 7.

2. Testing Distributed Behavior

Flight software is a critical component of modern spacecraft. Every mission carries its own requirements, including and driving the software-specific requirements. Software requirements formally describe the necessary behavior of the flight software and span many layers of abstraction and scope, from full-system behavior through standalone component descriptions and even individual function definitions. These requirements can be verified through software engineering approaches with unit, functional, and higher-level component and integration tests. But as the size and complexity of software increases, the need to test software behavior earlier in the development process also increases [5]. Furthermore, distributed system architectures necessitate more complex testing configurations because some software requirements can only be verified when executed in multi-agent environments. The flight software developed by DSA is complex but also a reusable technology platform for future distributed spacecraft missions. Therefore, we consider our testing framework to be an inherent component of the overall DSA software technology stack rather than a mission-specific tool.

Swarm Test Framework

The swarm test framework was developed concurrently with the development of the flight software comprising the DSA payload on the Starling 1.0 mission [2]. Developers can use the framework to run swarm tests on their workstations to verify distributed behaviors of the flight software quickly and interactively without requiring additional hardware or specialized commercial simulation software. The framework is also part of a continuous integration (CI) suite that automatically tests all submitted changes to the flight software code and verifies correct behavior in single-agent and multi-agent configurations in addition to traditional peer review methods. In the swarm tests, containers are configured in a *bridge* network, meaning each simulated spacecraft (container) is assigned a virtual Ethernet interface and bridged to every other simulated spacecraft, with a gateway interface assigned to the host operating system for container-host communication - shown in Figure 1. The resulting virtual network inside the host machine can be monitored and manipulated on a per-interface basis, yielding detailed metrics and insights on the network characteristics of flight software under many different configurations.

The simulation fidelity of this approach was briefly examined by quantifying the network performance limits of the

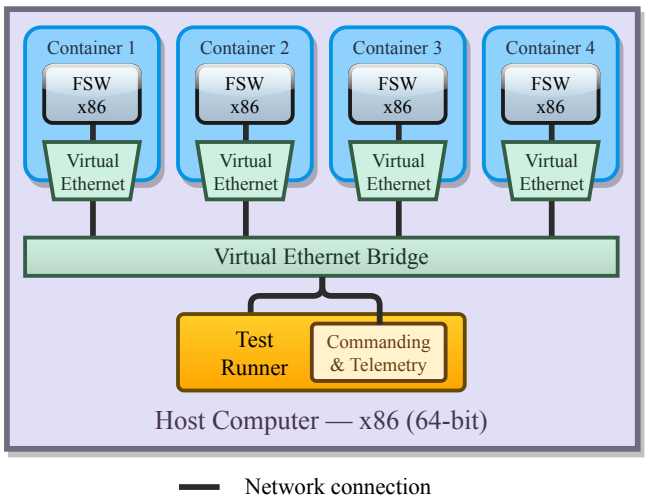


Figure 1. Network structure of the Swarm Test Framework with $n = 4$ spacecraft shown

container-based simulations, but experiments were limited to artificially generated traffic under worst-case topology and total network load [6]. While the results were useful insight into the performance limits of container networks on our particular hardware configuration, they did not model the real communication patterns expected in any scalable distributed space system. The network analysis was also limited to a smaller scale ($n = 32$ spacecraft) than the intended participation of the LPNT services ($n = 100$ spacecraft). Another factor affecting the simulation fidelity is the increased computational load due to non-network-related processing. In the current swarm test framework, the number of spacecraft simulated can be controlled by a simple integer parameter n. However, the computational capacity of the hardware running the swarm test framework is static, meaning that as the simulation scale increases, container processes will eventually saturate the available capacity and introduce artificial application faults.

The PiL testbed is designed to be the logical progression of the swarm test framework, increasing fidelity as a tradeoff with extra development effort and cost. Development for a mission always means eventually using actual flight hardware; the swarm test framework will never replace the fidelity of testing on the final system. But just as the swarm test framework is valuable as a generic interface to test distributed system behavior at the beginning of development, the PiL testbed offers value as a generic tool for testing larger-scale systems and different hardware architectures with many of the same features to explore a wide variety of potential mission configurations.

Network Testing

The network configuration of the PiL testbed is intended to operate at a similar level of control and abstraction as the swarm test framework. On each system, interactions with other spacecraft are handled through an Ethernet network interface rather than a particular hardware device, such as an emulated radio system. A swarm test framework node's network interface is virtual, but PiLs in the testbed have physical Ethernet ports. The equivalent network structure for the PiL testbed is visualized in Figure 2. Instead of tightly controlling for realistic network constraints, our approach to distributed spacecraft simulation primarily prioritizes observ-

ing behavior in ideal conditions, with the ability to introduce loss, delay, and other network degradation as an additional, secondary mechanism. Through the use of traffic shaping tools which are part of the mainline Linux kernel, traffic between different devices can be artificially delayed, rate-limited, or dropped [7]. We use the NetEm traffic shaping facilities, in conjunction with low-level network interface statistics tracked by the kernel, to monitor and precisely modify the network and behavioral characteristics.

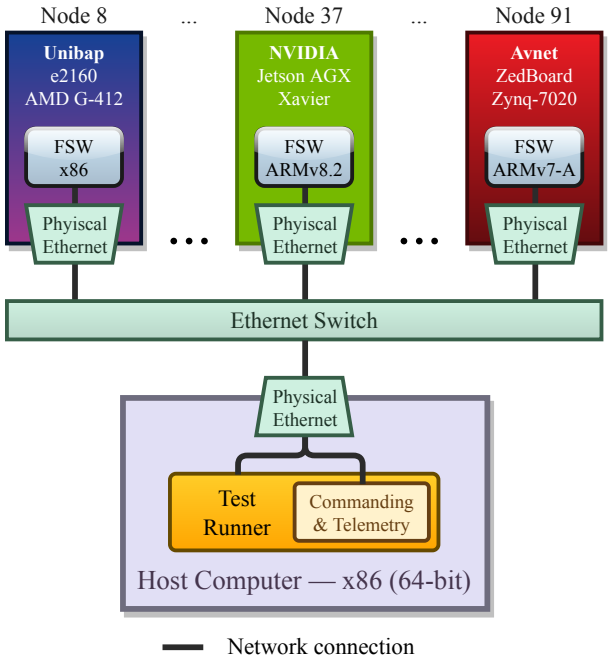


Figure 2. Network structure of the PiL testbed with 3 of 100 facsimile spacecraft shown

3. Hardware Configuration

Our testbed, known as the DISSTRACK (Distributed Intelligent Spacecraft Simulation Test RACK), contains a balanced selection of architectures that we expect to have a growing significance for edge computation onboard satellite systems. For this reason, our PiL rack contains both Nvidia and AMD GP-GPU architectures. Together, Nvidia and AMD accounted for 96% of the market share for discrete graphics processing units in Q1 of 2022. Nvidia accounted for 78% and AMD accounted for 17% [8]. Thus, if a GP-GPU is required in a space environment, it will likely to utilize at least one of these architectures. In fact, research regarding adapting Nvidia GPU/SoC technologies, typically from the Tegra line of products, for usage in small satellites is already occurring [9]. Alternatively, custom AMD-based CPU/GPU SoCs for satellites have matured greatly over the past several years. Now, corporations offer Commercial-off-the-Shelf (CotS) solutions for satellite edge computation devices based on the AMD architecture [10, 11].

In recent years, to balance the complex needs of edge computation, Neuromorphic accelerators have been included within the dye of SoCs. Many computational pipelines can benefit from the low Operations per Watt of Neuromorphic processors. In fact, Neuromorphic processors can be more power efficient and faster than traditional GP-GPU and CPU based approaches when performing object tracking, object classification, inferencing, and other applications that optimize the structure of a neural network [12]. As we built the DISSTRACK, we sought to include a small selection of these as accelerators which may prove to be useful for future applications.

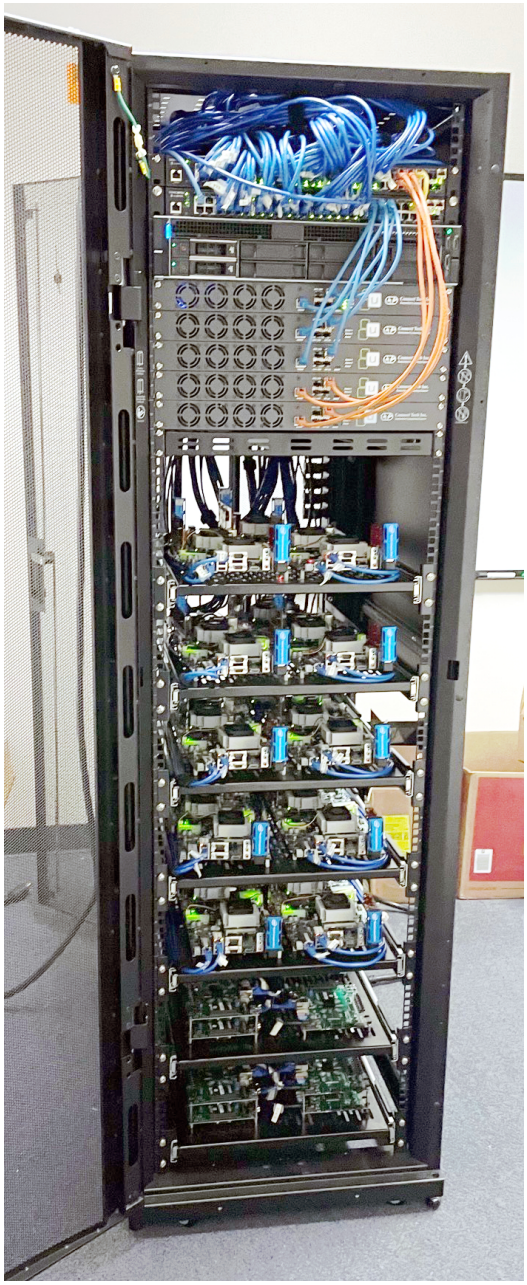


Figure 3. The Distributed Intelligent Spacecraft Simulation Test RACK (DISSTRACK) with 100 Processor-in-the-Loop nodes and supporting test infrastructure integrated into a standard server rack.

The Zedboard Zynq SoC Development boards were selected because they represent a common low-power option for small satellite flight computers, typically for computationally simple missions. However, computationally simple missions may still perform edge computation. For example, the SPOC (Spectral Ocean Color) cube satellite utilizes a Zedboard payload processor for spectral binning [13]. Additionally, the Starling 1.0 mission uses Zedboards for initial PiL integration testing. Thus, low-power Zedboards represent a class of small

satellite computing where high-performance edge computation is not required to accomplish mission objectives.

The DISSTRACK consists of 100 PiL devices, split across three different architectures and integrated into a single server rack: 60 Nvidia Jetson Xavier AGX CPU/GPU SoCs, 25 Unibap e2160 Qseven AMD CPU/GPU SoCs, and 15 Zedboard Zynq SoC Development Boards [14–16]. The server rack also contains two 48 port PoE capable managed switches and a 2U Dell PowerEdge server with 2TB of storage and 378GB of RAM. The 60 Nvidia boards are integrated with five 1U chassis containing 12 Nvidia Jetson Xavier AGX units, an internal managed switch connected to the rack switches, an Out-of-band Management (OOBM) Module, and redundant power supply units [17]. The Unibap Qseven boards are integrated onto five rack shelves with five boards per shelve. In addition to the carrier/breakout board, each Unibap Qseven board has a 1TB NVMe SSD connected via a PCIe adapter and an Intel Neural Compute Stick connected via USB. The boards are powered via PoE, with both interfaces (FPGA and SoC) connected to the rack-mounted network switches. The 15 ZedBoards are integrated onto two rack shelves, use 8GB SD cards for persistent storage, and are connected to the rack-mounted network switches. These boards and their specifications are also enumerated in Tables 1 and 2.

Table 1. Processing unit representation in DISSTRACK

Quantity	Device Model	CPU	GPU	FPGA
60	Nvidia Jetson Xavier AGX	✓	✓	
25	Unibap e2160 Qseven	✓	✓	✓
15	Avnet Zedboard	✓		✓

4. Software Configuration

Standalone small satellite missions typically manually flash software updates to spacecraft as needed. NASA's Starling 1.0 mission currently does not utilize an automated software deployment system to update the flight processors on the four flat-sats. With a few nodes, the engineering time required to manually build software and flash each flight processor before a test run is usually acceptable - about 4 to 8 minutes each. However, when the number of nodes scales to 100, it would require the entire workday to build and flash all flight processors for a single test. Even a slightly larger mission, such as NASA Ames' recently announced HelioSwarm (consisting of 8 nodes and a mothership), may desire an automated system for ease of development [18]. To avoid the overhead, DSA required a solution to manage PiL configuration, build, and deployment automatically - capable of scaling up to 100 nodes.

DISSTRACK operates entirely within a private subnet. All network addresses are static IPv4 addresses and utilize a Virtual Machine, known as the *test-runner VM*, running on the Dell PowerEdge server as the primary gateway and DNS forwarder. Every networked and rack-mounted component is directly accessible from the *test-runner VM* on the Dell PowerEdge server. The test-runner VM uses Salt as an infrastructure management and communications system for managing the 100 PiLs. When Salt is scripted, the *test-runner VM* can build, deploy, and run software on specified nodes [19]. With this software and network configuration, it is possible to expand the DISSTRACK to include more than 100 PiLs. To do this, simply daisy chain the new rack's switch back to the original rack via an upstream Ethernet connection. Only one *test-runner VM* is required per DISSTRACK installation, regardless of the number of racks and PiLs.

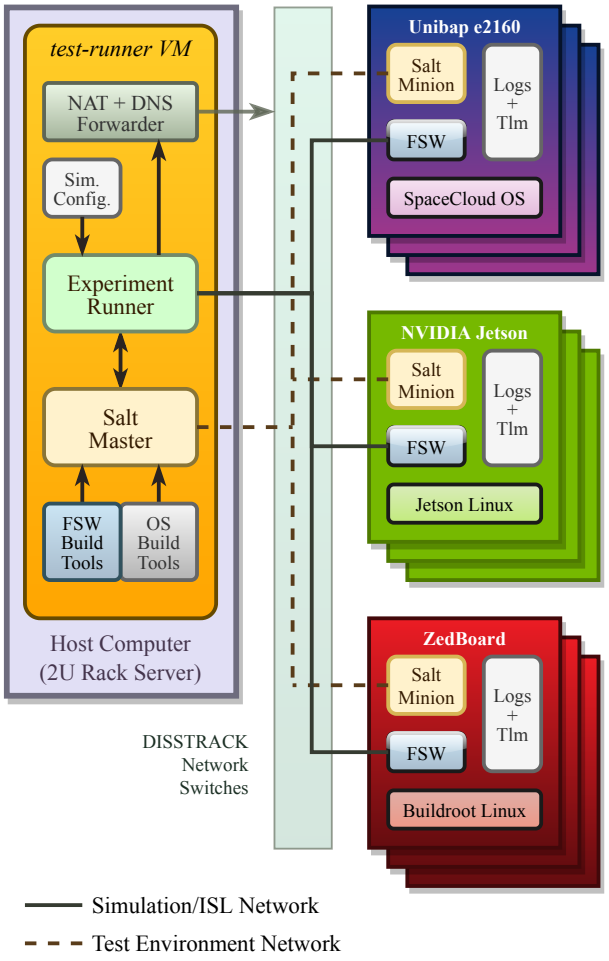


Figure 4. DISSTRACK Test Software Functional Diagram

The test-runner VM contains cross-compilation toolchains to build executables for the Nvidia, AMD, and Zynq targets. NVIDIA's Jetpack SDK provides a proprietary Linux system and toolchain, Linux for Tegra (L4T), based on Ubuntu 18.04 LTS. Unibap provides their ODE SpaceCloud software, based on Ubuntu Server 20.04 LTS, and uses the same toolchain as the test-runner VM. DSA uses Buildroot, a tool to generate embedded Linux systems, to create toolchains and Linux bootable images for the Zedboard PiL platform.

The experiment runner script is designed to be simple and modular while still spanning the configuration space needed for experiments. It implements a variety of maintenance functions and device target options which are enumerated and described in Table 3.

When running a Build, Update, Run, Stop, or Log command through the Experiment Runner, the user can specify a set or subset of nodes. The nodes include each board type, but the user can also specify the node-specific minion IDs of any Salt minion. The Build and Update steps typically take 8-15 minutes to run; this is roughly the time it would take to flash two traditional flat-sat systems.

The DISSTRACK utilizes post-mortem artifact analysis

Table 2. Processor-in-the-loop architecture specification details

Component	Board	Architecture	Processing Cores	Memory	Max Freq.
CPU	Nvidia Jetson Xavier AGX	ARM-v8.2 (64-bit)	8 Logical	32 GB	2.3 GHz
	Unibap e2160 Qseven	x86 (64-bit)	4 Logical	2 GB	1.2 GHz
	Avnet Zedboard	ARM-v7 (32-bit)	2 Logical	512 MB	667 MHz
GPU	Nvidia Jetson Xavier AGX	Nvidia Volta	512 CUDA	shared w/ CPU	1.37 GHz
	Unibap e2160 Qseven	AMD Radeon R3E	2 CU	shared w/ CPU	350 MHz
FPGA	Unibap e2160 Qseven	Microsemi SmartFusion2	57k Logic Elements	512 MB	
	Avnet Zedboard	Xilinx Artix-7	85k Logic Cells		

Table 3. Command Options for the Experiment Runner

argument	**description**
`-help`	displays help options
`-unibap`	targets all unibap boards
`-nvidia`	targets all nvidia AGX boards
`-zedboard`	targets all zedboards
`-container`	targets docker containers
`-nodes`	targets individual nodes by salt minion ID
`-build`	builds executables for the desired boards
`-update`	pushed updated configs to the desired boards
`-run`	runs cFS on the desired boards
`-clear-screen`	clears all existing screen sessions before running
`-stop`	stops cFS on the desired boards
`-log`	fetches the runlogs and DS from cfs
`-reboot`	reboots the desired targets from settings
`-all`	auto sets zedboard, nvidia, unibap, build, update, and run

rather than live supervision to keep network overhead low and PiLs as flight-like as possible. Crucially, we have found all desired behaviors and metrics to be well logged in flight software with the need to add artificial instrumentation for visibility. While in flight, access to these full records might be infeasible due to bandwidth or storage limits; however, the run-record-retrieve lifecycle of a DISSTRACK test allows for "offline" retrieval after execution, bypassing these operational restrictions in order to improve simulation fidelity.

Generating well-calibrated observations of the system state requires that timestamps between arbitrary PiLs be comparable, and therefore time must be well-synchronized across the entire rack. In DISSTRACK we synchronize clocks using the IEEE 1588 Precision Time Protocol (PTP) standard instead of the more common Network Time Protocol (NTP) used by common network devices. PTP uses a series of delay measurements with the master clock on the test server's network interface; this works well in the compact, direct networking environment of the DISSTRACK. We are not concerned with the absolute correctness of the clock time with any external reference; only the simultaneity of clocks within the test rack is necessary for valid interpretation of test artifacts.

5. LUNAR POSITION, NAVIGATION, AND TIMING

The purpose of DSA's Lunar PNT (Position, Navigation and Timing) scalability study is to study the applicability of DSA technologies to the problem of providing PNT services on and around the Moon in the absence of a dedicated constellation. In prior research, DSA developed a scheduler that satisfies user requests for PNT services from ad-hoc, non-dedicated orbital constellations around the Moon [4]. Additionally, DSA developed a cFS application around RTI (Real-Time Innovations) DDS (Data Distribution Service), an implementation of the DDS standard, that operates as a microservice for multicasting and routing arbitrary UDP packets for distributed publish-subscribe communication. This application, known as the COMM App, is on each satellite in the Starling 1.0 mission and is also used for the LPNT scalability studies. For LPNT studies on the DISSTRACK, DSA builds and deploys the entire cFS executable alongside several additional applications built for our mission concept. Though the details of our flight software are not discussed in this paper, it is important to note the intention of this work is to increase TRL and show that the software, in its native form, scales on representative hardware.

Our LPNT localization services make no assumptions about the specific constellations nor the orbits of the service providers. We assume an omnidirectional transmitter on each satellite that can provide PNT services to multiple ground targets within the satellite's field of view. Additionally, we assume each satellite can crosslink via an ISL (Inter-Satellite Link) to two additional satellites within the constellation. Thus, we assume we can generate a large, connected graph of satellites. Our software uses simulated ranging measurements, measured via ISLs, as input to a DEKF (Distributed Extended Kalman Filter) with also distributes measurements over the network.

Without DISSTRACK, validating and testing 100 instances of flight code would require enormous computational resources. To simulate flight code at this scale, the swarm test framework or an equivalent approach could be used on much a much larger computing resource to virtualize the spacecraft but would still lack the application specific interfaces and architecture typical of spacecraft. Furthermore, the cost of such computing resources would approach the cost of the dedicated hardware facsimiles that virtualization would replace during early development.

Our goal in building DISSTRACK was not only to simplify the task of building and deploying flight software to 100 flight processors, but to also test an actual large-scale distributed spacecraft system. Currently, DSA is testing a scenario which involves 25 satellites running the LPNT flight software. We simulate a 3-satellite dropout, lowering the total constellation size to 22, and we are working to evaluate the increase in PNT accuracy as a result. We plan to present more comprehensive

results in future research.

6. Results

To quantify the functional characteristics of the DISSTRACK at its current state in development, we present two tests of relevant test framework components: time-to-deploy updated flight software on the PiLs, and clock synchronization residuals. These tests on their own are not sufficient indicators of the DISSTRACK meeting its intended functional goals, but we believe they still serve as meaningful contextualization of DISSTRACK's abilities towards those goals.

First we show deployment times for the Unibap nodes using our experiment runner script. Because the build step of the experiment runner only needs to run one once per platform, we exclude that from the timing analysis to more clearly isolate the scalability of the system. The results of those measurements are shown in Figure 5. Timing data is the real time elapsed in seconds as reported by the standard *time* shell utility and averaged over 5 update runs for each set of nodes.

Deploying small updates is fast because the flight software is cached on the PiLs and we use an incremental file list, implemented in the standard *rsync* program, to ensure that only changed software files are synced. The software change used in each update is a change to the DSA COMM Application's cFS table. A cFS table contains parameters that can update or reconfigure a cFS Application on the cFS executable's startup or when arbitrarily commanded. Here we updated 8 values that represent a reconfiguration of the network; This type of reconfiguration is common to do many hundreds of times over the duration of integration and testing and is therefore a representative workload for measuring deployment speed.

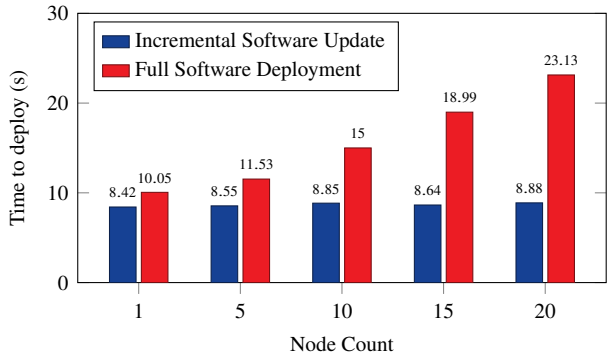


Figure 5. Effects of simulation scale on software deployment time (lower is better)

We see a relatively constant (only ±0.23 seconds) average deployment time when caching is enabled and deployment is scaled from 1 to 20 nodes. In contrast, deploying completely new builds with no caching increases considerably as simulation scale increases. Although time savings is only 14 seconds at most as measured, it is clear that a naïve deployment would result in significant lost time over the many repeated updates required by integration and testing efforts. We expect this relationship will hold as deployment scale increases to 100 nodes.

The distribution of residuals of the PTP clock synchronization are shown in Figure 6. All 20 test nodes used in this test leverage software timestamping, which is inferior to the hardware timestamping available on the Jetson nodes. These

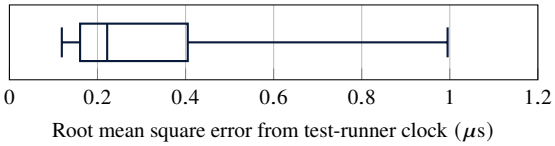


Figure 6. Distribution of clock error for 20 nodes

nodes are therefore the weakest link in the synchronization chain–bounding confidence levels when comparing timestamps. Also of note is that the clock skew is more than ten orders of magnitude smaller than any of the measured clock errors, so the smoothed running adjustments to these clocks can be anticipated to be well-behaved. The synchronization of the timestamps outpaces the temporal resolution of the logs used to generate performance metrics and can be trusted to produce those measurements without introducing significant uncertainty. This result validates our offline retrieval approach to test evaluation.

7. Ongoing and Future Work

The primary demonstration for the DISSTRACK, currently under active development, is the scalability of the DSA flight software for autonomous, multi-agent Lunar Position, Navigation, & Time (LPNT) services, described above and in previous publications from DSA [20]. More specifically, we want to demonstrate the service-providing abilities of the LPNT flight software under multiple conditions: simple idealized network availability, anomalous network conditions, peer spacecraft failures, and under full load leveraging all 100 PiLs in the DISSTRACK. These test conditions are derived directly from the challenges identified by DSA corresponding to the collective complexity, human-swarm interaction, and swarm scale in the context of scientific spacecraft missions [3].

The intermediate milestones for meeting these goals start with running the flight software in partial levels of functionality. We developed many LPNT extensions to the DSA flight software using the swarm test framework at small scales under ideal network conditions.

In the future, we want to verify that our testbed supports network traffic needed by the full load of 100 simulated spacecraft to quantitatively validate that DISSTRACK meets the levels of fidelity desired.

Acknowledgments

This work was funded by the NASA Game-Changing Development Program (GCD) as part of the Distributed Spacecraft Autonomy (DSA) Project.

References

[1] H. Sanchez, D. McIntosh, H. Cannon, C. Pires, J. Sullivan, S. D'Amico, and B. O'Connor, "Starling1: Swarm technology demonstration," in *Proceedings of the AIAA/USU Conference on Small Satellites*, ser. Small Satellite Conference, 2018. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2018/all2018/299/

[2] N. Cramer, D. Cellucci, C. Adams, A. Sweet, M. He-

jase, J. Frank, R. Levinson, S. Gridnev, and L. Brown, "Design and testing of autonomous distributed space systems," in *Proceedings of the AIAA/USU Conference on Small Satellites*, 2021.

[3] D. Cellucci, N. B. Cramer, and J. D. Frank, "Distributed spacecraft autonomy," in *ASCEND 2020*. American Institute of Aeronautics and Astronautics, Nov. 2020. [Online]. Available: https://doi.org/10.2514/6.2020-4232

[4] S. Niemoeller, J. Frank, R. Burton, R. Levinson, and N. Cramer, "Scheduling PNT service requests from non-dedicated lunar constellations," in *2022 IEEE Aerospace Conference (AERO)*. IEEE, Mar. 2022. [Online]. Available: https://doi.org/10.1109/aero53065.2022.9843208

[5] D. Dvorak, *NASA Study on Flight Software Complexity*. NASA, 2009. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2009-1882

[6] W. Vaughan, A. George, B. Kempa, D. Cellucci, and N. Cramer, "Evaluating network performance of containerized test framework for distributed space systems," in *Proceedings of the AIAA/USU Conference on Small Satellites*, ser. Small Satellite Conference, 2021. [Online]. Available: https://digitalcommons.usu.edu/smallsat/2022/all2022/209/

[7] S. Hemminger, "Network emulation with netem," in *Proceedings of the 6th Australia's National Linux Conference (LCA2005)*, 2005, pp. 18–26.

[8] "Pc discrete graphics processing unit (dgpu) shipment share worldwide from 1st quarter 2019 to 1st quarter 2022, by vendor," Oct. 2022. [Online]. Available: https://www.statista.com/statistics/1131242/pc-discrete-gpu-shipment-share-by-vendor-worldwide/

[9] C. Adams, A. Spain, J. Parker, M. Hevert, J. Roach, and D. Cotten, "Towards an integrated gpu accelerated soc as a flight computer for small satellites," in *2019 IEEE Aerospace Conference*. IEEE, Mar. 2019. [Online]. Available: https://doi.org/10.1109/aero.2019.8741765

[10] F. Bruhn, K. Brunberg, J. Hines, L. Asplund, and M. Norgren, "Introducing radiation tolerant heterogeneous computers for small satellites," in *2015 IEEE Aerospace Conference*. IEEE, Mar. 2015. [Online]. Available: https://doi.org/10.1109/aero.2015.7119158

[11] F. C. Bruhn, N. Tsog, F. Kunkel, O. Flordal, and I. Troxel, "Enabling radiation tolerant heterogeneous GPU-based onboard data processing in space," *CEAS Space Journal*, vol. 12, no. 4, pp. 551–564, Jun. 2020. [Online]. Available: https://doi.org/10.1007/s12567-020-00321-9

[12] G. Bersuker, M. Mason, and K. Jones, "Neuromorphic computing: The potential for high-performace processing in space," in *The Aerospace Corporation*, Oct. 2022.

[13] D. Cotten, N. Neel, D. Mishra, M. Madden, C. Adams, S. Ullrich, A. Burd, M. Adams, K. Summey, C. Versteeg, J. Parker, and F. Beyette, "The spectral ocean color imager (spoc) – an adjustable multispectral imager," in *Proceedings of the AIAA/USU Conference on Small Satellites*, 2019.

[14] "Jetson agx xavier series," Sep. 2022. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/

[15] "e20xx / e21xx computer modules," Sep. 2022. [Online]. Available: https://unibap.com/en/our-offer/space/spacecloud-products/e20xx-e21xx/

[16] "Zedboard zynq-7000 product," Sep. 2022. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-8dyf-11.html

[17] "Agx inference server," Sep. 2022. [Online]. Available: https://connecttech.com/product/agx-inference-server/

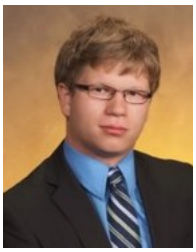
[18] L. Plice, A. Dono Perez, and S. West, "Helioswarm: Swarm mission design in high altitude orbit for heliophysics," in *AAS/AIAA Astrodynamics Specialist Conference*, no. AAS 19-831, 2019.

[19] M. Zadka, "Salt stack," in *DevOps in Python*. Springer, 2019, pp. 121–137.

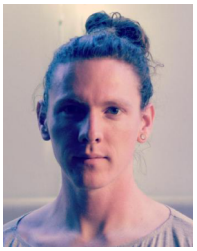
[20] B. Hagenau, B. Peters, R. Burton, K. Hashemi, and N. Cramer, "Introducing the lunar autonomous pnt system (laps) simulator," in *2021 IEEE Aerospace Conference*, 2021, pp. 1–11.

BIOGRAPHY

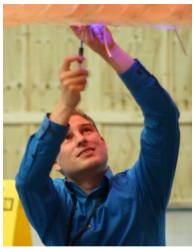
Caleb Adams *is the project manager of Distributed Spacecraft Autonomy at NASA Ames Research Center in the Intelligent Systems division. He was the co-founder of the UGA Small Satellite Research Laboratory, which now has 3 satellite missions. He has a Masters Degree in Computer Science from the University of Georgia and his research focuses on small satellites, distributed computing, and perception systems.*



Brian Kempa *is a Pathways Intern in the Intelligent Systems Division's Intelligent Robotics Group at NASA Ames Research Center and has developed flight and operations systems since 2014. He studied Aerospace Engineering at Iowa State University, receiving his B.S in 2017 and his M.S. in 2019. He is currently working on a Ph.D. in Aerospace and Computer Engineering researching applications of runtime formal methods for trusted autonomy.*



Walter Vaughan *is a Pathways Intern in the Intelligent Systems division's Core Avionics and Software Technologies (CAST) group at NASA Ames Research Center. He received his B.S. in Electrical Engineering from the South Dakota School of Mines & Technology in 2019 and his M.S. in Electrical & Computer Engineering from the University of Pittsburgh in 2022. He is also a Predoctoral Trainee at the NSF Center for Space, High-Performance, and Resilient Computing (SHREC) and is currently pursuing a Ph.D. in Electrical & Computer Engineering at the University of Pittsburgh. His research interests include virtualization technologies for spacecraft, distributed architectures, software reliability, and heterogeneous embedded computing.*



Nicholas Cramer *is a Lead Aeronatics Researcher at Supernal. He was the project manager of Distributed Spacecraft Autonomy at NASA Ames Research Center. His areas of research span, distributed systems, intelligent structures, perception systems, and broad applications of autonomy. Nick occasionally teaches a UAV design class at San Jose State and received his Ph.D. in Computer Engineering with a focus in robotics from the University of California, at Santa Cruz.*