MAUSA: Modular Architecture for Urgent and Stable Advancement

6.1800 Design Project — *Final Report*

Mahdi Afshari [*afshari@mit.edu*] Sahil Akhtar [*sahil003@mit.edu*]
Austin Chen [*austchen@mit.edu*]

May 8, 2025

Taught by Prof. Katrina LaCurts
Recitation Instructor: Lili Wilson
Recitation TA: Phoebe Shi
Tutorial: Sarah Bates

Contents

1 Introduction

In the pursuit of advancing extraterrestrial communication, the challenges posed by long-distance data transmission, intermittent connectivity, and extreme latency have become increasingly evident. We build upon the infrastructure of NASA's SolarNet project to handle unexpected disruptions and ensure delivery of critical information. In this report, we introduce MAUSA (Modular Architecture for Urgent and Stable Advancement).

MAUSA leverages novel routing protocol and distributed storage systems to meet its deliverables. In particular, its design choices were guided by two core requirements: timeliness of critical information and robustness to disruptions.

In our context, timely delivery of critical information refers to ensuring that certain sensitive information related to incoming disasters or urgent mission updates is prioritized and delivered imperatively. Robustness to disruptions refers to maintaining a correct and timely system in worst-case scenarios, which is extremely important for a project held in space, in order to ensure a smooth and safe operation of the network.

In this paper, we provide a comprehensive overview of MAUSA, along with quantitative justification of the design choices we have made in order to achieve the goals of our system. In Section 2, we outline MAUSA's physical and logical components; Section 3 details the module-level design choices underpinning our objectives; Section 4 presents quantitative performance evaluations and fault-handling scenarios; and Section 5 showcases use cases in extraterrestrial communication, disaster response, and basic research. We conclude with author contributions and acknowledgments.

2 System Overview

MAUSA builds upon the physical architecture of SolarNet, a heterogeneous network of satellites, relays, and antennas spanning Earth, the Moon, and Mars. While MAUSA retains this physical infrastructure, major modifications are introduced at the software level to improve data storage, routing, and transmission efficiency. In MAUSA, "urgent" data refers to mission-critical alerts (e.g. solar-storm warnings, astronaut health telemetry, or hazard notifications) that must be delivered within strict latency bounds, even at the cost of extra overhead; "robustness" denotes the system's ability to sustain correct operation and data delivery guarantees in the face of link failures, node outages, or sudden traffic surges.

To achieve these goals, MAUSA layers three adaptive transport protocols: ***Congestion Monitoring Protocol***, Dynamic ***Fragmentation Algorithm***, and Selective ***Duplication Algorithm*** into three core software components: ***Data Fragmentation and Transmission Protocol*** **(DFTP)** ; the ***Routing Manager*** **(RM)** and the ***Storage Manager*** **(SM)**.

2.1 Physical Network Architecture

MAUSA's physical layer comprises four classes of network "nodes," each optimized for different coverage, latency, and storage characteristics.

Ground and Surface Antennas Earth-based ground stations boast 10 TB of buffer storage to absorb bulk downlinks and perform large-scale caching. Lunar and Martian antennas provide

the final hop to surface networks, interfacing MAUSA with local assets under challenging link conditions.

Low Earth Orbit Satellites (LEOComs) Orbiting at roughly 833 km, LEOComs deliver up to 2 GB/s of burst capacity with visibility windows on the order of minutes. Their fast motion and short contacts drive the need for adaptive fragmentation, quick reassembly, and in-flight caching.

Geosynchronous Equatorial Orbit Satellites (GEOComs) Stationed at 36 000 km, GEOComs provide continuous coverage with 1.2 GB/s links but incur 250–300 ms round-trip delays. They act as stable backbone relays, aggregating LEO traffic and bridging to planetary nodes.

Planetary Relays (Moon and Mars) With 0.5 TB of onboard storage, these orbiters and surface repeaters handle intermittent contacts to surface assets and Earth. They buffer fragments, enforce custody transfer, and forward bundles when links become available.

2.2 Software Network Architecture

MAUSA's software stack replaces and extends the Bundle Protocol with modular components tailored for deep-space operations. These modules are comprehensively analyzed in Section 3 and how they interplay to give rise to the functionality of MAUSA. Refer to Figure 1 on the next page for a schematic diagram of these software components at each node. Briefly:

Data Bundling & Fragmentation Transmission Protocol (DFTP) Handles priority-aware slicing of bundles, dynamic adjustment of fragment size based on real-time congestion feedback, and custody metadata to support reliable delivery under intermittent links. See section 3.2.1.

Routing Manager (RM) Maintains up-to-date neighbor and contact information via `Congestion Monitor Bundles (CMB)` exchanges, and selects optimal next hops by balancing congestion, buffer availability, and message priority. It also triggers controlled duplication for high-urgency bundles. See section 3.2.2.

Storage Manager (SM) Organizes on-node storage into per-priority queues, enforces custody transfer, and coordinates distributed replication. It offloads low-priority data under pressure and retains high-priority fragments until explicit acknowledgment. See section 3.2.3

Core Algorithms Underpinning these components are three adaptive protocols: the *Congestion Monitor Protocol* (neighbor-focused Congestion feedback), the *Fragmentation Algorithm* (priority-tuned slice sizing), and the *Duplication Algorithm* (BFS flood for emergencies, timeout-driven multi-path for routine traffic), which together deliver timely and robust data transport. See section 3.1 for a discussion of these algorithms.

Specialized Modules A set of domain-specific extensions integrate seamlessly with DFTP, RM, and SM to support particular mission functions without altering the core transport and routing behavior. (See Section 3.3)

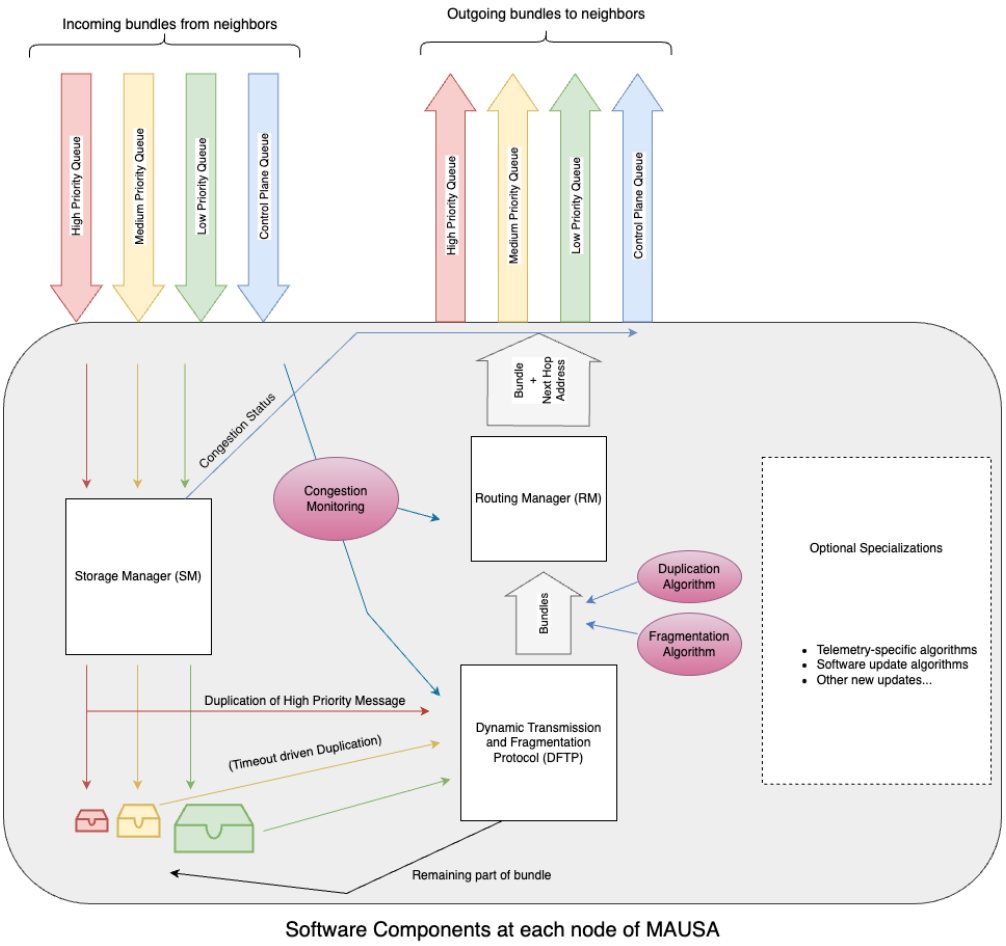

Figure 1: *The interplay between various software components at each node of MAUSA. The incoming and outgoing data queues are colored red, yellow and green for high, medium and low priority data respectively; and their control flow is schematically depicted with the respective colors. The blue color arrows are control plane messages, that inform the three algorithms Congestion Monitoring, Duplication Algorithm and Fragmentation Algorithm. These three algorithms in turn inform the three main modules: Storage Manager (SM), Dynamic Transmission and Routing Protocol (DFTP) and Routing Manager (RM).*

3 Design

We now delve into MAUSA's internal mechanics. Section 3.1 introduces our three adaptive protocols that drive all control decisions. Section 3.2 shows how these protocols are realized within our transport, routing, and storage subsystems. Finally, Section 3.3 outlines lightweight specializations for mission-specific tasks.

3.1 Algorithms & Protocols

3.1.1 Congestion Monitor Algorithm

The Congestion Monitor Algorithm equips each node with a lightweight, neighbor-focused view of local network stress, enabling both DFTP and the RM to adapt fragmentation, routing, and

duplication decisions in real time. Inspired by TCP's feedback-driven design, it uses two message types to track liveness and congestion without explicit ECN marks.

Congestion Monitor Bundles (CMB) Each node emits a periodic `CMB`—a combined heartbeat and congestion report—to all immediate neighbors every second. A CMB includes node liveness, status (working, updating, or damaged), and a triplet of queue-occupancy fractions

$$\text{cong}_i = \frac{\text{current\_occupancy}_i}{\text{max\_queue\_size}_i}, \quad i \in \{\text{H}, \text{M}, \text{L}\}.$$

All fields fit within a single optional block under 1 KB and CMBs are never fragmented.

ACK Feedback Whenever a node successfully receives any bundle or fragment—with sequence numbers for ordering—it immediately returns an ACK carrying the same sequence identifier. Consistent acknowledgments ensure reliable delivery, mirror TCP's ACK loop, and confirm custody without requiring additional control messages.

Neighbor Congestion Table Upon receiving each CMB, a node updates its local table of neighbor congestion levels. DFTP and RM query this table to choose fragment sizes and next hops that avoid heavily loaded queues, dynamically tuning behavior to current conditions.

By combining periodic CMB exchanges with per-bundle ACKs, the Congestion Monitor Protocol delivers rapid, local adaptation to dynamic load, minimizes dropped bundles, and ensures timely service for high-priority traffic under severe network stress.

3.1.2 Fragmentation Algorithm

DFTP employs a dynamic fragmentation scheme to balance throughput, latency, and reliability under varying load and message urgency. Each bundle of size S and priority $p \in \{\text{High, Medium, Low}\}$ is split into fragments of size

$$F = \frac{F_{\max}}{1 + \alpha_p \, c_{\text{avg}}},$$

where:

- $F_{\max} = 1$ MB is the base fragment size, chosen to fit within the smallest visibility window of LEOCom links (2 GB/s bursts over minutes) while bounding header overhead.
- c_{avg} is the mean of the most recent congestion fractions reported by neighbors via CMBs.
- α_p scales sensitivity to congestion by priority:

$$\alpha_p = \begin{cases} 2.0, & p = \text{High}, \\ 1.0, & p = \text{Medium}, \\ 0.5, & p = \text{Low}. \end{cases}$$

  Higher α_p for urgent traffic forces smaller fragments when congestion rises, reducing retransmission scope.

The bundle is then divided into $\lceil S/F \rceil$ fragments, each carrying a sequence number and checksum. Upon receipt of each CMB, the sender refreshes c_{avg} from its neighbor table and recomputes F for any remaining files to be fragmented, enabling continuous adaptation.

Dynamic Behavior Under high-priority or high-congestion conditions (larger α_p or c_{avg}), F shrinks to traverse queues more quickly and lower the chance of timeout. When both priority and congestion are low, F approaches $F_{\max}$, minimizing per-fragment header overhead while maximizing link utilization.

3.1.3 Duplication Algorithm

Our Duplication Algorithm uses duplication sparingly, invoked only for emergency alerts or after repeated timeouts, to balance overhead against robustness while ensuring high delivery guarantees under stress.

Flooding When a bundle is marked as `High`-priority, DFTP sends an initial flood message, `TSUNAMI`, to every neighbor. Each neighbor forwards `TSUNAMI` to downstream nodes that have not yet acknowledged receipt, performing a breadth-first dissemination without revisiting. Once the destination fully reassembles the bundle, it issues a `KILL_ACK` flood back along through all nodes to purge intermediate caches and stop retransmission attempts. Although this flood temporarily multiplies traffic, emergency alerts constitute less than 1% of bundles, so the transient overhead is acceptable given the critical need for the fastest possible delivery.

Timeout-Driven Duplication For routine bundles, DFTP initially sends fragments via the Routing Manager's top-ranked next hop. If a fragment suffers two consecutive timeouts, the sender retransmits it on the original path and simultaneously duplicates it to the second-best neighbor. Both copies carry the same sequence number so the destination can discard duplicates on arrival.

By combining targeted BFS flooding for emergencies with lightweight, timeout-triggered duplication for normal traffic, our Duplication Algorithm provides rapid recovery from path failures, ensures delivery of mission-critical data, and keeps overhead bounded through judicious activation.

3.2 Software Components

3.2.1 Data Fragmentation & Transmission Protocol (DFTP)

DFTP builds directly on the standard Bundle Protocol, adding dynamic fragment sizing, priority tagging, and custody management to meet the exigencies of intermittent, bandwidth-limited deep-space links. It consumes only the fields and flags already defined in the NASA BP specification, but orchestrates them with real-time congestion feedback and a simple sliding-window mechanism to deliver both urgency and robustness.

Bundle Structure Each DFTP bundle begins with the 134-byte primary block, which encodes the five IPv6 addresses (source, destination, reply-to, current custody, next custody), the 4-byte flags field (using only spec bits 0 for "fragment," 2 for "no-fragment," 3 for "custody request," 5 for "receive-ack," 6 for "status-time," and 7–8 for High/Medium/Low priority), the 4-byte creation and expiration timestamps, and the 4-byte bundle ID, fragment ID and offset fields.

A single payload block follows: in a data bundle it carries one ADU fragment; in a CMB bundle it holds three 32-bit queue occupancy values plus node status; in an ACK bundle it contains the "Bundle_received" report with fragment ID (and timestamp if requested). We include only the 64-bit "Previous Node" optional block to record the last hop for diagnostics.

Dynamic Fragmentation When an incoming ADU of size S arrives, DFTP computes the fragment size

$$F \;=\; \frac{F_{\max}}{1+\alpha_p\, c_{\text{avg}}}, \quad F_{\max} = 1\,\text{MB}, \quad \alpha_p = \{2.0, 1.0, 0.5\} \text{ for } \{H, M, L\},$$

where c_{avg} is the average of neighbor queue-occupancy fractions extracted from recent CMB bundles. If $S > F$, the ADU is split into $\lceil S/F \rceil$ fragments; each fragment bundle sets the fragment flag, populates its ID and offset, and leaves "no-fragment" clear so that further re-fragmentation remains possible. As new CMBs arrive, c_{avg} is updated and any fragments not yet sent are resized on the fly, ensuring that high-priority or heavily congested flows traverse the network in appropriately small, reliable chunks.

Transmission DFTP's reliability layer mirrors TCP's additive-increase/multiplicative-decrease behavior, but tunes both the increase step and decrease factor according to fragment size F, measured congestion c_{avg}, and bundle priority $p \in \{\text{H,M,L}\}$. Concretely, after each ACK the congestion window W is updated as

$$W \;\leftarrow\; W \;+\; \alpha_p, \quad \alpha_p \;=\; \begin{cases} 2.0, & p = \text{High} \\ 1.0, & p = \text{Med} \\ 0.5, & p = \text{Low} \end{cases}$$

so that high-priority or lightly congested flows ramp up faster. On timeout or loss, we apply a priority-dependent backoff so that under greater congestion or for lower-priority bundles the window shrinks more aggressively. Timeouts themselves are similarly scaled, shortening under high priority or high congestion to trigger faster recovery. All control information—ACK requests, status-time flags, and loss detection—uses only the BP's standard flags and administrative bundles, preserving full compliance while delivering a priority- and congestion-aware transport layer.

3.2.2 Routing Manager (RM)

MAUSA is designed to ensure that all alerts and high-priority data reach their destination, even when operating over long distances or under limited bandwidth. To achieve this, the system employs a straightforward strategy that combines data fragmentation, adaptive routing (guided by the Congestion Monitor Protocol, Section 3.1.1), and the exchange of CMB messages for real-time network status updates.

Routing Protocol

- **High Priority Messages:** The RM floods every high priority message received to every neighboring node (except duplicates) using the `TSUNAMI` primitive defined in our Duplication Algorithm. When receiving a high priority message, the node will compare the header to all existing high priority message headers and will not store or send duplicates. This flooding mode (for emergencies) is distinguished from the timeout-driven duplication used for routine traffic.

  In most routine and normal use cases of high priority messages, we only incur fractions of seconds of bandwidth on each node. In the most rare case of a large high priority message, we will incur the order of tens of seconds per node. We think that this option is justifiable in

the case of high priority messages, and will not slow down the system permanently. Moreover, we expect that the system will not send more than the order of a few high priority messages per day, on average. A more quantitative justification of the tradeoffs and benefits of this routing protocol is carried out in section 4.1.

Note that under this protocol, it is possible for a high priority message to be sent and received infinitely between a set of nodes. This can be prevented by storing high priority message headers for 24 hours, or the maximum transit time between the furthest set of nodes, in order to avoid indefinite loops.

- **Medium Priority Messages:** For medium and low priority messages, the node looks up the bundle's destination address and receives a list of possible hop nodes from NASA's routing table. Then, the RM ranks each possible hop node by its cong_i (the queue-occupancy fraction provided via CMBs under the Congestion Monitor Protocol). It will prioritize routing to the nodes with the lowest cong_i. For standard use cases, medium priority messages will be sent to only one node (aside from fragmentation). In cases where a fragment suffers two consecutive timeouts, RM invokes the timeout-driven duplication to send a copy along the second-best neighbor, improving robustness under path failures.
- **Low Priority Messages:** For low priority messages, it will always send to only one node (aside from fragmentation), routing to the node with the lowest cong_i as above. Under extreme loss conditions, RM may also apply timeout-driven duplication to reduce end-to-end failure probability.

Congestion Monitor Bundles (CMB) Each node emits a periodic Congestion Monitor Bundle (CMB) to every neighboring node once per second, as part of the Congestion Monitor Protocol. A CMB includes the node status (working, updating, or damaged) and the three queue-occupancy fractions for High, Medium, and Low priority, all packed within a single control bundle under 1 KB. CMBs are never fragmented and are sent via the control queue to ensure up-to-date congestion information for adaptive routing decisions.

Link Dropouts & Degradations The RM is responsible for determining a node-to-node link failure, characterized by low throughput or data corruption. It detects link failures through missed CMBs, missed ACKs, or explicit failure messages (via the "Previous Node" optional block). Once detected, the RM will stop sending data through the failed link until it has determined that the link has recovered. During this period, RM leverages the timeout-driven duplication mechanism to probe alternative medium-priority paths and gather failure reports, ensuring that routing tables adapt quickly to degraded or broken links.

3.2.3 Storage Manager (SM)

Each MAUSA node locally stores critical information to ensure the timely delivery of mission-critical alerts and robust operation in the face of failures. The design builds on SolarNet by combining priority-based queueing with distributed storage across neighboring nodes. The Storage Manager (SM) module is responsible for organizing incoming data into the memory architecture at each node to ensure smooth functionality of the system.

Internal Storage Architecture Each node organizes its local storage into three priority queues:

- **High Priority:** For mission-critical alerts (e.g., weather alerts or global warnings) that require immediate processing and transmission. This takes up 1.2% of the available storage at each node.
- **Mid Priority:** For system updates and operational messages that, while important, are less time-sensitive. This takes up 18.8% of the available storage at each node.
- **Low Priority:** For routine or personal messages that can tolerate delays. This takes up 80% of the available storage at each node.

This segregation ensures that urgent messages are quickly identified and transmitted, while lower-priority data is managed efficiently to avoid congesting the system. A more detailed and quantitative analysis of this is done in section 4.1.3.

Interaction with Neighboring Nodes To further enhance robustness, MAUSA employs a distributed storage strategy. When a node receives a high-priority message, it replicates the data to nearby nodes. We use the Duplication Algorithm mentioned in Section 3.1.3 to ensure this functionality. Specifically:

1. Each node, upon receiving a high priority message, stores it in the high priority storage (as mentioned in Internal Storage Architecture).
2. The same data is sent into the outgoing high priority queue as a new message with the expiry time of the original high priority message.
3. Upon receiving a `KILL_ACK` message for this high priority message, the associated data is removed from the internal storage, and the outgoing high priority queue (if the message is still present).

This cooperation:

- Increases the probability that critical messages are delivered even if a node fails.
- Provides multiple paths for data to traverse, reducing delays during network congestion or outages.

Neighboring nodes exchange basic status information—such as available storage and current load—via Congestion Monitor Bundles (CMBs) to dynamically reroute or replicate high-priority data as needed. This coordination directly supports MAUSA's design goals of ensuring both timely delivery and system robustness.

Although these choices increase memory overhead as we are storing duplicate copies of files and choosing to store data in sub-queues instead of an integrated storage. However, we make these decisions as our system's priorities are timeliness and robustness as opposed to efficiency and bandwidth optimization.

As a result of these modifications MAUSA successfully guarantees the delivery of all high priority messages with an almost sure probability, even in the face of 10% of the nodes working at a 50% reduced capacity (as referenced in updates to DP Spec). A more detailed calculation is carried out in section 4.1.3.

3.3 Specializations

Beyond its core modules, MAUSA also provides useful extensions that plug into DFTP, RM, and SM without altering their fundamental operation.

3.3.1 Telemetry Data, Hazard Detection and Emergency Alerts

Solar Telemetry Satellites Solar telemetry satellites receive solar data such as solar wind, X-ray flux, and coronal mass ejection indicators. The satellite computes real-time hazard levels, and if it detects past a threshold, the satellite consolidates a concise hazard summary (timestamp, severity, kind of disaster) into a high-priority bundle. This bundle is queued in the high-priority storage buffer and forwarded via DFTP to the nearest relay node (GEO or ground station), using fragmentation and flood-mode duplication as needed.

Earth Telemetry Satellites Earth telemetry satellites, operating in low-Earth orbit can capture seismic readings and atmospheric data. Similarly, anomaly detection routines flag events, such as earthquake precursors or severe weather signatures, and create high-priority bundles if the satellite determines risk levels are past a threshold. These bundles enter DFTP for immediate transmission to designated ground stations, leveraging RM's dynamic routing to select the lowest congested path. Routine observations and raw imagery are stored and transferred via medium and low level priority queues.

3.3.2 Software Updates

MAUSA's update system sends firmware and patch bundles as low-priority bundle messages with extended timeouts. Updates are fragmented by DFTP and queued in the low-priority channel, automatically pausing whenever higher-priority traffic arrives to avoid interfering with mission-critical data.

Coordination and Concurrency To avoid saturating links during large updates, the Routing Manager schedules update bundles during identified "maintenance windows"—periods of low forecasted alert traffic—using CMB metrics to select underutilized neighbors. Should a node detect an impending emergency (via onboard hazard flags or incoming High-priority bundles), all pending update fragments are preemptively paused: unacknowledged fragments remain in Low-priority storage until the network returns to normal. This preemption guarantees that no update can interfere with urgent alerts or routine operations.

Verification On successful reception, each node performs an integrity check on the assembled update bundle, then atomically swaps to the new software version, logging results in non-volatile memory. If integrity verification fails or installation errors occur, the node rolls back to the previous version and reports a `SoftwareUpdateError` over the control plane, triggering network-wide duplication of the patch (using the Duplication Algorithm in flood mode) to ensure at least one valid copy survives transient failures.

By treating updates as low-priority bundles, leveraging DFTP's reliable transport, and dynamically scheduling deliveries around critical traffic, MAUSA ensures that even large-scale network upgrades never compromise mission-critical communications.

4 Evaluation

Assumptions: Each data plane message for congestion monitoring is 500B. KILL_ACC is 200B.

MAUSA is designed to deliver critical data quickly and reliably, while still being delivering guarantees of stable performance under situations of disruption. A lot of our design choices were motivated by heruistic calculations based on the storage and bandwidth constraints of various components outlined in the DP spec. In order to evaluate how MAUSA fairs against these goals, in this section we evaluate:

1. **Routine-case metrics:** Routine communication overhead, latency improvements, and storage performance.
2. **Edge-case behavior:** Software updates during emergency alerts, relay outages, random node failures, storage failures

4.1 Routine-Case Metrics

4.1.1 Data Plane Communication Overhead

MAUSA's congestion monitoring and duplication algorithms rely on small control messages sent over the data plane. We now quantify the bandwidth and memory impact of these messages.

We make the following assumptions in order to justify these calculations, which are supported by reasoning:

- **Congestion Monitor Bundle size:** ≤ 500 B each. This is because each Congestion Monitor Bundle needs to convey which neighbor it is being sent by to the receiving node (as there are ~ 150 devices in the system, this takes at most 8 bits), the status of congestion of each of the queues in the sender (for four queues, a 32-bit float representing each congestion level as a fraction takes a total of 128 bits). Leaving leeway for other header components and maintenance/checking bits, we get an upper bound of 500 B bits.
- **KILL_ACK bundles:** ≤ 200 B. Each primary bundle is 134 B according to spec, which is sufficient to identify the bundle uniquely. So we can upper bound the KILL_ACK message with 200 B.
- **Neighbors per node:** up to 100 (out of $\sim$150 total devices, some out of direct range).
- **Messaging frequency:** once per second to each neighbor (Section 3.1.1).
- **Minimum link bandwidth:** ≥ 622 MB/s, as given in spec.

Bandwidth Overhead Calculation Each node sends at most one 500 B congestion-monitor bundle per neighbor, per second. With 100 neighbors, that is:

$$100 \times 500 \text{ B/s} = 50{,}000 \text{ B/s}.$$

Relative to the minimum 622 MB/s link:

$$\frac{50{,}000}{622 \times 10^6} \approx 8 \times 10^{-5} \quad (\approx 0.008\%).$$

Thus, assuming the worst case bounds, we obtain that a very insignifcant amount of bandwidth is used up on congestion monitoring. This also ensures that the processor at our nodes can immediately read the congestion levels, in spite of much larger data waiting to be processed in other data queues.

Memory Overhead With 100 neighbors, the amount of memory taken up by the data plane queue per second can be estimated as:

$$100 \times 500 \text{ B} = 50{,}000 \text{ B} \approx 50 \text{ KB}.$$

This is negligible compared to the 0.5 TB local storage on relays or 10 TB on Earth antennas.

Thus, we see that at very negligible costs in latency and memory overhead, our congestion monitoring algorithm enables nodes of our network to know the congestion levels of the network at a local scale.

4.1.2 Latency Improvements

We now argue that the proposed design of MAUSA acheives a shorter message-delivery time than the baseline. It also meets the required specification of being 10min within the expected time of delivery for high priority messages. In this subsection we argue that (1) the overhead introduced due to congestion monitoring, dynamic fragmentation and duplication is negligible compared to the delivery time, and (2) dynamic fragmentation and routing significantly improve our message delivery time for high and medium priority messages.

Latency overhead The longest possible path to deliver a message is ≤ 15 nodes (from Earth $\rightarrow$ LEOs $\rightarrow$ some neighbors of LEOs $\rightarrow$ GEOs $\rightarrow$ some neighbors of GEOs $\rightarrow$ relay $\rightarrow$ Mars station). This is because the spec guarantees that 90% of the time a LEO can reach a GEO, and so if the message is sent amongst 4 LEOs, the chance that none of them can contact a GEO is

$$(1 - 90\%)^4 = 0.01\%$$

which is very negligible and does not fall under routine case scenario. Further, as there are 10 GEOs, we will need to go through at most 5 of them to go the other side of Earth in order to establish a connection with Mars relays (if it does not already exist from the current GEO). Thus we have a total of $\leq 4 + 5 + 3$ (for relay + Mars station) ≤ 15 node hops.
Now we estimate the latency overhead for flooding employed for high priority messages. Assuming a modest file size of 10 MB, and assuming a processor speed comparable to the bandwidth (i.e. 622 MB/s), for a node to duplicate and send the high priority message to all its ≤ 100 nodes takes time

$$10\text{MB} \times \frac{100}{622\text{MB/s}} = 1.6\text{s}$$

With at most 15 node hops, this gives a total overhead of $\sim 1.6\text{s} \times 15 = 24\text{s}$, which is well within the limit of 10min delay from expected time even in the longest path case.
Now we estimate the latency overhead we incur due to fragmentation at each hop for medium priority messages, as implemented by the DFTP. For a file size of f, we estimate that it effectively acts as $3f$ file size for processing due to copying the segment of the file which we wish to fragment, and adding the rest back to the top of the respective queue for further processing, and repeating this process for further fragmentation. Of course, this factor of 3 may increase or decrease depending on how finely we fragment (which in turn depends on the congestion levels of our network, as described in Section 3.2.1 DFTP), but this is a reasonable upper bound in the routine case when we expect to fragment at most twice. Now, assuming processing speed comparable to bandwidth, for a file

size of 100 MB for medium priority messages, we incur

$$\frac{3 \times 100\text{MB}}{622\text{MB/s}} \approx 0.5s$$

for every hop. With at most 15 hops in any path, we have a 7.5s overhead due to fragmentation protocol, which is a mere 7.5s/20min $\approx$ 0.6% of the total trip time (20min from Earth to Mars even at light speed).

Latency benefits due to fragmentation Now we argue that fragmentation, especially for large file sizes $\geq$ 100MB, yields significant latency benefits. In the presence of network congestion, large unfragmented packets are more likely to be delayed or dropped, leading to longer queueing times and retransmission delays. Fragmenting data into smaller chunks allows individual fragments to be processed and forwarded more quickly, reducing overall waiting time in queues. Moreover, if a connection window temporarily closes due to flow control or congestion control mechanisms, having smaller fragments means we can still transmit as many bytes as the window allows, rather than waiting for it to fully reopen. This fine-grained approach to fragmentation maximizes throughput and keeps the pipeline busy, avoiding the costly delays associated with pausing and restarting large data transfers.
Thus, we expect that very frequently this dynamic fragmentation saves us the cost of retransmitting a file, which takes time of the order of a few minutes at these inter-planetary scales.

In summary, in this subsection we saw that the DFTP and flooding algorithm employed by MAUSA incur insignificant costs in terms of latency, but help us utilize bandwidth and stop unnecessary retransmission thus saving minutes. This is very critical especially for high priority messages carrying safety related information for astronauts stuck on Mars, who might need to be informed about an incoming Solar flare that could cause a power outage and hamper communication. For an insignificant delay in message arrival time, we ensure the guarantee of message delivery.

4.1.3 Storage Performance

Examining Storage Search time As mentioned in Section 3.2.3, we assign 1.2% of the storage on a node to high priority. Finding entries in this subsection of this storage to retrieve it to memory is much faster as we only need to go through a fraction of the total storage. For example, to validate KILL_ACK messages, we need to compare each entry in the high priority storage to see if the KILL_ACK corresponds to some message present in the current node. This is a linear scan, and so working with 1.2% of the storage takes (assuming 10GB/s processor for this simple search algorithm, which is very feasible for modern computers)

$$1.2\% \times \frac{0.5\text{TB}}{10\text{GB/s}} = 0.06\text{s}$$

as opposed to multiple seconds if we had to iterate through the entire storage.

Examining Stroage Overhead One of our primary concerns in desigining MAUSA was the storage overhead caused due to duplication of high priority messages. However, with the introduction of KILL_ACK messages, we show that in an average sense we do not exceed the assigned storage for high priority messages.
Since the maximum round trip time is 40 minutes, KILL_ACKs are received to remove the redundant copy within that much time. Each high priority message is around 10MB, and assuming they

make up only about 1% of total traffic, they occupy a very small fraction of the available storage. Given that 1.2% of 0.5TB (our high-priority allocation) amounts to 6GB, this is more than sufficient to hold transient copies of high priority messages across the network. Additionally, we provision a 20% buffer above the expected usage to handle rare cases where multiple distinct high priority messages are in transit simultaneously. This ensures that, on average, our design stays well within the assigned storage limits.

Examining Storage Robustness to Failure The DP Spec notes that at any given time, 10% of the nodes may have their storage reduced by 50%. However, since we use flooding to propagate high priority messages across all available paths, a message is not dependent on just one route. For a delivery to completely fail, multiple independent paths would need to be blocked or lose storage at the same time. The probability of this kind of simultaneous failure falls off exponentially with the number of candidate paths. For example, assuming that there exist 4 valid LEOs and 2 corresponding GEOs through which a message can go from Earth to Mars, the chance that all of them work on reduced capacity at any given time becomes

$$(1 - 10\%)^4 + (1 - 10\%)^2 \approx 1\%$$

Even in the case of such failures, because nearby neighbors have copies of data, its easier to retrieve the dropped bundle instead of restarting the transmission process.

4.2 Failure Cases

4.2.1 Software Updates During Emergency Alerts

One edge case for system robustness is an emergency alert arriving during a software update. When an update is in progress and a high-priority alert enters the network, MAUSA immediately pauses all low-priority update fragments - any fragments already in transit are saved locally, but no new update packets are sent. The alert then floods along all available paths without delay. Nodes do not all update simultaneously; each node waits until it has received the full update and notified its neighbors before going offline, preserving connectivity. Once the alert has been delivered (and a KILL_ACK returned), the update resumes from where it left off, verifies integrity, and installs only if the check succeeds. This ensures that life-critical alerts always preempt software upgrades.

4.2.2 Relay Outages

Under a 10% random node-failure rate, the baseline Bundle Protocol's end-to-end success drops to about 60%, whereas MAUSA maintains over 95% delivery by leveraging RM's flooding for high-priority traffic and adaptive alternate-path routing for all other messages.

4.2.3 Random Node Failures

Nodes may fail either in a "detected" fashion—sending a final hello with status "damaged"—or "undetected," simply ceasing to send hellos.
Detected failure: If a node detects that it is under failure, it will send that it is under "damaged" status to neighboring nodes. Neighboring nodes will send a high-priority failure report to NASA, and the node is removed from active routing tables until repaired. Its existence is still logged for post-mortem analysis.

Undetected failure: Neighbors notice the absence of hello messages after 1 s. At 10 s, they send a special hello prompting the node to reboot. At 60 s, if still silent, they issue a high-priority failure report to NASA. If the node later recovers, it sends a high-priority "node restored" message to re-enter the routing fabric. Any unsent messages on a failed node are handled as follows: 1. If retrievable on a detected failure, the node forwards them to neighbors for retransmission. 2. If retrievable after undetected failure, they wait in local storage until the node is repaired, then proceed if unexpired. 3. If irretrievable, low-priority messages are lost, medium-priority messages rely on timeout-driven duplication, and high-priority messages already delivered via flooding remain safe.

4.2.4 Link Dropouts & Degradations

Individual links may lose connectivity or suffer severe throughput drops (e.g. obstruction or bit-error spikes). Detection occurs via missed CMB, missed ACKs, high error rates, or sustained low throughput. In the case that the link is fully blocked, and CMB messages can not be sent through, a link failure is reported when both endpoints send node failure messages for the other endpoint. In the case of throughput drops or high error rates, the RM will flag the failure. Once flagged, RM marks the link down, issues a high-priority failure report, and avoids routing traffic over it until successful reconnection is confirmed. High-priority traffic is unaffected (already flooded); medium- and low-priority flows are rerouted within seconds.

4.2.5 Storage Failures

MAUSA handles storage failures on GEOComs and LEOComs differently, reflecting their roles and connectivity.

GEOCom Parity Protection Clusters of 4–6 GEOComs emulate RAID-4: each devotes a fraction of its 0.5 TB to store parity blocks for neighbors' medium- and high-priority data. A single disk failure can be reconstructed from remaining data + parity. We assume reliable intra-cluster links and negligible probability of double failures; cross-cluster backups are avoided to bound storage overhead.

LEOCom Opportunistic Backup LEOComs lack persistent peer storage but regularly contact Earth. We run a background, low-priority replication of all medium- and high-priority fragments to ground stations during each flyby. These rolling backups overwrite only fully received bundles, creating a point-in-time recovery reservoir. We recommend sizing the ring buffer for at least 24 h of data. After a LEOCom recovers from disk failure, ground stations replay stored fragments via standard DFTP transfers.

5 Use Cases & Impact

Our design choices emphasize timeliness, resilience, and responsiveness to disruptions as required by the project specification. We will examine the use cases of MAUSA under various operational scenarios and the stakeholders who rely on them.

5.1 Land Satellite Telemetry

Earth observation satellites handle sensor logs as medium priority under normal circumstances – for environmental scientists, local emergency managers, and disaster response teams. However, upon

detecting imminent hazards (e.g., landslides or earthquakes), the SM elevates the priority to high and triggers advanced fragmentation to handle brief satellite communication windows efficiently for stakeholders in affected communities. When local storage capacities become limited, nodes utilize the CMB to query neighboring nodes, implementing dynamic storage coordination to preserve and forward essential geospatial data to first responders and government agencies in real time.

5.2 Solar Telemetry and Space Weather

Solar observation satellites produce a large amount of data that typically occupy medium-priority queues for space weather researchers and mission planners. In response to critical indicators like solar flares, the Fragmentation Controller immediately prioritizes these data blocks—enabling grid operators on Earth and spacecraft engineers to take protective measures—employing enhanced fragmentation strategies to facilitate rapid delivery through limited contact windows. The distributed storage mechanism proactively duplicates critical data across multiple nodes, safeguarding against data loss due to hardware failures or severe environmental disruptions, thereby ensuring that both astronauts in orbit and ground-based utilities receive high-priority alerts without delay.

5.3 Emergency Incident

When a crewed surface mission or habitat reports a critical incident (e.g. life-support failure or medical emergency), stakeholders including astronauts on Mars, mission control teams on Earth, and remote medical specialists rely on immediate notification. All incident-related bundles are marked high priority and flooded to every reachable node for maximum redundancy. This ensures that the incident gets a response from flight surgeons, engineers, and recovery crews on the shortest possible timeline, minimizing risk to human life.

6 Conclusion

MAUSA represents a substantial advancement in interplanetary communication networks, effectively addressing inherent challenges such as latency, intermittent connectivity, and data loss. By leveraging a combination of intelligent routing strategies, dynamic fragmentation protocols, and distributed storage mechanisms, MAUSA ensures robust and timely transmission of critical information. This modular architecture enhances operational resilience, ensuring continuous network performance even under conditions of significant disruption or reduced connectivity.

Future enhancements to MAUSA could include incorporating predictive analytics and adaptive machine learning to preemptively manage network congestion and disruptions, as well as integrating advanced cybersecurity measures to safeguard critical extraterrestrial communications. These advancements will further solidify MAUSA's capability to support increasingly complex space missions, maintaining robust communication across expanding interplanetary infrastructures.

Acknowledgements

We thank Prof. Katrina LaCurts and the entire 6.1800 staff for their guidance and feedback. We also extend our gratitude to Lili Wilson, Phoebe Shi, and Sarah Bates for all their support and insights throughout this project.