SNORT (SolarNet Optimized Routing Transmission) 6.1800 Design Project Report

Cameron Holt, Jennifer Kim, Kayli Requenez

Recitation Instructor: Katrina LaCurts
WRAP Instructor: Jessie Stickgold-Sarah

May 5, 2025

---

1. Introduction

SolarNet, NASA's extraterrestrial communication network, is shifting from point-to-point links to a satellite-based architecture. To support this shift, our team has developed SolarNet Optimized Routing Transmission (SNORT)—a system designed to provide reliable and timely data delivery across space.

SNORT implements a distributed storage system, an enhanced forwarding protocol, and an extended API to accomplish two main goals: (1) **reliable transmission** between endpoints despite environmental volatility and (2) **timely delivery of bundles** to ensure critical data remains relevant and actionable. Since speed and reliability are often seen as contradictory design goals, we define timely delivery explicitly as bundles reaching their destination before their expiration date. We argue that SNORT's design can simultaneously achieve high reliability and beat these expiration deadlines.

SNORT's capabilities impact a diverse range of stakeholders, including astronauts whose quality of life and safety depend on reliable routine communications with ground control. Additionally, communities vulnerable to natural disasters rely on timely satellite telemetry for effective emergency response, directly benefiting from SNORT's optimized data delivery.

Section 2 outlines our system and its components. Section 3 details each component and design tradeoffs. Section 4 analyzes the impacts of our design choices across use cases. Section 5 evaluates our system under its use cases, and we reiterate our design goals and tradeoffs. Section 6 discusses additional considerations and future improvements to SNORT. Sections 7 and 8 conclude the paper with author contributions and acknowledgements.

2. System Overview

SolarNet provides the physical foundation for space communication, consisting of users (e.g., astronauts, scientists) and hardware infrastructure (e.g., LEOComs, GEOComs, relays, antennas). On top of this foundation, SNORT introduces virtual components to enable reliable data transfer: an API interface, a distributed storage system, and enhanced forwarding mechanisms for the Bundle Protocol (BP). SNORT's modules are shown in Figure 1, and Figure 2 shows how the modules interact to transmit a bundle.

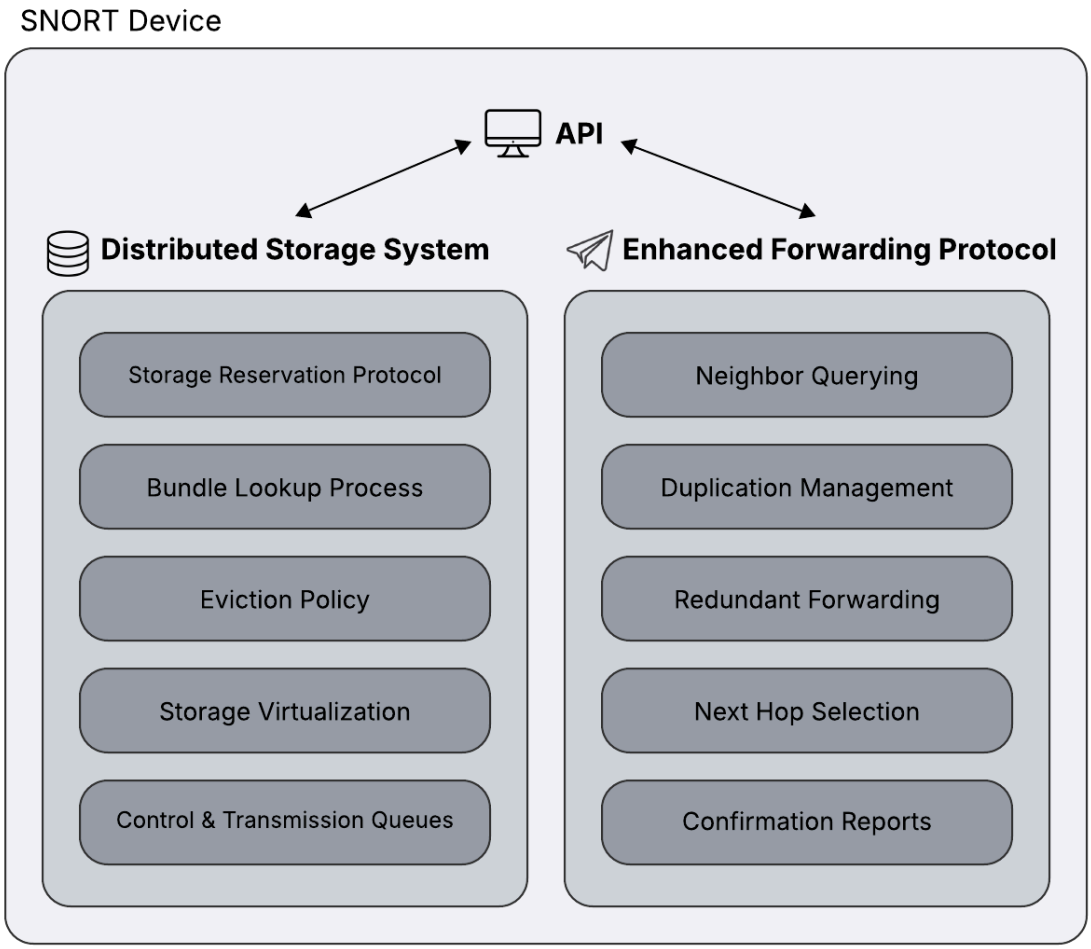

FIGURE 1: SNORT's Modules and Submodules

Each of SNORT's modules is composed of submodules. The storage system uses the reservation protocol to allocate space before receiving bundles, with a bundle lookup process for fast metadata access and an eviction policy to manage storage pressure. Bundles and transmission queues are stored in a virtualized long-term storage layer across multiple devices for fault tolerance. The forwarding protocol integrates neighbor querying—coordinated with the reservation protocol to ensure space is secured before sending—and uses a next hop selector to determine the order of queries. It employs redundant forwarding and confirmation reports for reliability, while the duplication management system discards excess bundles created to maintain timely delivery. Finally, the API enables usage of these modules. Together, these submodules ensure SNORT maintains performance and reliability across volatile network conditions.

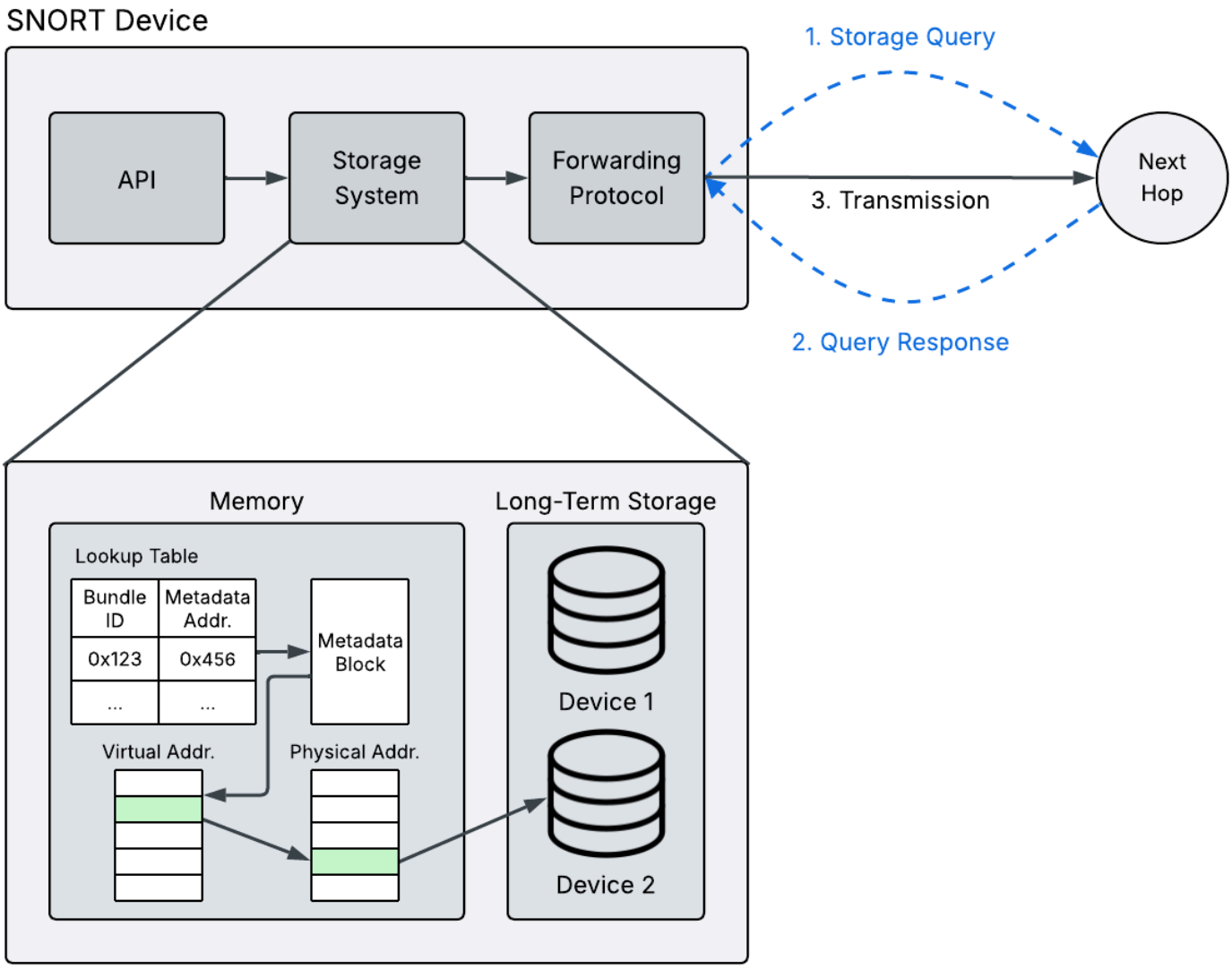

FIGURE 2: Example Bundle Forwarding

This figure details SNORT's basic bundle transmission process after a bundle is popped from the queue. The bundle lookup process retrieves the bundle's metadata using its ID, which contains forwarding information and a virtual storage address. This virtual address is then translated to a physical location specifying the storage device and position. The bundle is transmitted once the node confirms that the next hop has sufficient available space.

3. System Description

This section is organized into two parts: the storage system (3.1) and enhanced forwarding protocol (3.2). Both utilize new and extended API calls detailed in Appendix 9.1.

To avoid ambiguity, we define some commonly used terms. A *small bundle* is any bundle of size at most 250MB, primarily covering administrative and routine communication bundles. The largest known small bundle is solar flare messages (200MB). Accordingly, we define a *valid remaining capacity* from the routing table to be of at least *small bundle* size. This ensures that extremely small fragmentation won't congest the network and prevent timely delivery.

3.1 Storage System

SNORT's storage system consists of several key modules designed to ensure reliability and efficiency. The storage reservation protocol allocates space for incoming bundles, while the eviction protocol handles pressure when space is limited. The bundle lookup process supports fast access to bundle metadata used to identify each bundle's forwarding state, and a virtualization layer abstracts multiple devices to tolerate hardware failures. A memory failure protocol enables recovery of critical tables after unexpected memory loss.

3.1.1 Storage Reservation Protocol

The storage reservation protocol supports the bundle querying process, where a node requests storage space from a neighbor before sending a bundle, ensuring it maintains sufficient capacity.

To do this, each node tracks two values: *Remaining Storage* and *Free Storage Threshold*. Upon receiving a query, a node must:

1) **Check Available Storage**: A node calls `get_storage_available()` to determine the maximum space it can reserve without dropping below the *Free Storage Threshold*. It then calls `get_amount_transmissible()`, which factors in this value along with the remaining bandwidth capacity between the two nodes. If the bundle size exceeds either limit, the node checks whether it can be fragmented into two valid pieces, each at least the size of a *small bundle*. If so, it reserves space for the largest valid piece. If the node doesn't have enough space, it will follow the eviction protocol (3.1.6).
2) **Reserve**: This simply involves decrementing *Remaining Storage*, as no specific physical location is allocated yet. The node marks the bundle as an *Expected Bundle* in memory (3.1.5) and calls `estimate_round_trip_time()` to determine the reservation period, which accounts for both the time to respond to the query and receive the actual bundle, and internally multiplies the estimate by two to provide a safety margin for potential delays. It then `calls deallocate_space()`, which restores the space and removes the bundle's entry from memory if it hasn't arrived by the end of the reservation period.
3) **Reorganize**: Once the bundle arrives, the node searches for a suitable location in long-term storage. If fragmentation has left no large enough contiguous space, the node performs coalescing to compact storage and create room for the bundle.

Nodes maintain the *Free Storage Threshold* to guard against late-arriving bundles that miss their reservation window. The threshold is set to the size of the largest known emergency bundle (2GB for landslide alerts), allowing the node to handle critical messages even under storage pressure.

3.1.2 Bundle Lookup Process

Bundle metadata is stored in memory to support fast retrieval without accessing full bundles from long-term storage, which is often unnecessary for routine operations.

The metadata is organized in the Bundle Lookup Table, which maps each Bundle ID to the address of its metadata block. For fragmented bundles, multiple Fragment IDs are associated with one Bundle ID. In that case, the table maps the Bundle ID to a fragment block, which lists all known fragments stored on the node. Each Fragment ID then maps to its own metadata block. A metadata block includes the primary block of a bundle, a pointer to the bundle in long-term storage, and fields that track its delivery state (3.1.5). This process is illustrated in Figure 3.

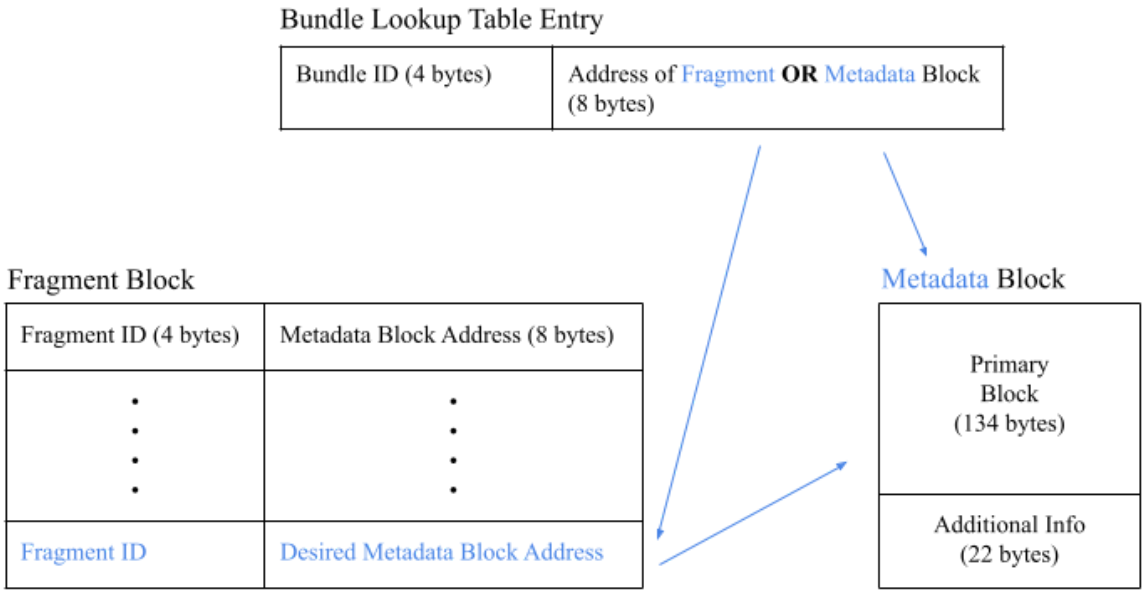

FIGURE 3: The bundle lookup process.

3.1.3 Storage Virtualization

LEOComs and GEOComs use two storage devices, but one may fail intermittently. To simplify the bundle lookup process and ensure continued operation during partial failures, SNORT introduces a storage virtualization layer that presents both devices as a unified logical space.

The system prefers using the more stable device to reduce risk of data loss. If a failure occurs, the node temporarily decreases *Remaining Storage* and removes the entries of any bundles stored on the affected

device. This appears as a brief transmission loss for those bundles, but SNORT's redundant forwarding and confirmation reports ensure they are still likely to reach their destinations via alternate paths.

3.1.4 Long-Term Storage

To enable seamless recovery after disruptions—including system failures, restarts, and updates—bundles and transmission queues are always kept in long-term storage. Among these queues is the control-plane queue, which manages system-generated bundles (transmission reports, queries, etc.) and is prioritized above all other queues to keep the system up to date and responsive. Memory maintains pointers to the head of each queue to efficiently identify the next bundle for transmission.

Each queue stores pointers to the metadata blocks of each bundle, allowing the node to check relevant information without needing to retrieve it from long-term storage, improving efficiency.

3.1.5 Bundle Storage States

A node can identify each bundle as *Stored*, *Expected*, or *Recently Delivered* based on how it is referenced in memory. These states reflect the bundle's forwarding status and guide how to handle it. To make state-based decisions, the node references the bundle's metadata block, which contains both the primary block and additional information unique to SNORT (Figure 4).

Field Name	Size (bytes)	Description
Address in Storage	8	The address of this bundle in long-term storage.
Expected Size	6	The expected size of this bundle which this node has reserved space for.
ACK Retransmission Time	4	The earliest time this bundle can be retransmitted if no ACK has been received, recorded in seconds since Jan. 1, 1990.
Next Critical Retransmission Time	4	The earliest time this bundle can be retransmitted if no custody transfer confirmation has been received, recorded in seconds since Jan. 1, 1990.
Total	22	

FIGURE 4: Additional information stored in metadata block.

In examining each state, note that both metadata block addresses and storage addresses are considered invalid if set to -1.

There are three bundle states:

1. Stored Bundles: They have both a valid metadata block address and storage address, ensuring they are retrievable before potential forwarding. Some bundles may remain in storage because they require transmission reports before they can be safely removed. This is where additional metadata becomes important, as it includes two key timing values associated with the following scenarios:

A. Awaiting Acknowledgement (ACK): If a bundle requires an ACK, it remains stored so that it can be retransmitted if the ACK is not received before *ACK Retransmission Time*.
B. Awaiting Custody Transfer and/or Delivery Confirmation: If the node has custody over the corresponding bundle and is awaiting a custody transfer report from the next custodian or is the source of the bundle and is awaiting a delivery confirmation report from the destination, it uses *Next Critical Retransmission Time*. If both cases apply, this field is first set for custody transfer and afterwards updated for end-to-end reliability.

If either value is set to 0, this means the bundle does not require the corresponding report.

2. Expected Bundles: They have been allocated space in a node but are not yet received. In this case, the bundle has a valid metadata block address but an invalid storage address. The *Expected Size* metadata field represents the reserved space and has been subtracted from *Remaining Storage* to prevent over-allocation.

3. Recently Delivered Bundles: They have an invalid metadata block address and storage address, indicating they have reached their final destination and no longer require storage or forwarding. These entries are retained for 24 hours to support duplicate detection at the destination and upstream nodes (3.2.3).

This classification system supports forwarding decisions, discussed in the following section.

3.1.6 Eviction Protocol

This process occurs when a node is queried for storage space it does not currently have. Within the query, a node receives the expiration time of the incoming bundle and compares it against those of the bundles it currently stores.

The node only considers evicting expired bundles, and only if they have a later expiration time than the incoming bundle. This strategy favors retention of fresher data while dropping only stale bundles, reducing the risk of meaningful data loss. If there would still not be enough space after eviction, the node declines the reservation and notifies the querying neighbor.

3.1.7 Memory Failure Protocol

Memory failures fall into two categories: memory exhaustion and unexpected memory loss. Exhaustion is highly unlikely, as each device has 64GB of memory, far exceeding the required capacity to store memory tables even under maximum bundle load. For example, at full 10TB storage utilization, a node could store up to 40,000 *small bundles*. The largest memory components, the metadata blocks, use only about 6 MB at 156B per block – negligible compared to 64GB. Even with 1KB bundles, data is transmitted quickly, making it unlikely to saturate storage with small bundles alone. We also must consider *Recently Delivered Bundles* that aren't in long-term storage. Since they are tracked as lightweight 4-byte entries (no metadata block) and retained for only 24 hours, their impact on memory usage is minimal. Worst case, the system follows the eviction protocol to clear memory. In the case of memory loss, tables can be fully reconstructed by scanning long-term storage.

3.2 Forwarding Protocol

The SNORT Forwarding Protocol enhances the standard Bundle Protocol to ensure reliable and timely delivery across SolarNet's dynamic and delay-prone environment. It increases delivery success via redundancy and state-aware retransmission, while reducing network load through duplicate suppression and adaptive routing based on custody, storage, and expiration pressure. Key features include query-based neighbor probing, reservation-backed transmission, and metadata-tagged duplication control.

3.2.1 Core Features

Query Handling

Before transmission, a node invokes `query_neighbor()` to check if the neighbor can accept the bundle. The neighbor follows step 1 of Section 3.1.1 which considers storage and bandwidth constraints as well as fragmentation feasibility to determine 1) how much space it will reserve for the bundle and 2) the reservation period. The neighbor notifies the querying node of these values via `storage_available()`. There are special response values for how much space is reserved:

- -1 if the bundle is *Recently Delivered* (i.e., a duplicate bundle; see Section 3.2.3),
- 0 if no space is available, or
- a positive value if partial or full transmission is possible.

Upon receiving a valid response, the sender re-invokes `get_amount_transmissible()` to re-check the validity of the reservation, ensuring that both capacity and timing are still valid before transmission proceeds.

Confirmation Reports

To ensure reliable and timely delivery, SNORT employs three types of confirmation reports. System-generated bundles may not require such reports to prevent transmission loops.

1. **Acknowledgement (ACK)**: Sent to the previous hop to confirm successful handoff, preventing further backtracking. Used only for priority 3 bundles due to potential congestion.
2. **Custody Transfer Confirmation**: Notifies the previous custodian that the full custody bundle is held by the next custody node.
3. **Delivery Confirmation**: Sent by the destination to confirm final receipt. Source nodes retain bundles until this confirmation to guarantee end-to-end reliability. Given their large storage (10TB), sources can afford to store bundles longer than devices with limited capacity (e.g., 0.5TB).

If a node does not receive the expected confirmation report within a designated timeout period, it must retransmit the bundle. Timeouts represent the expected round-trip times (to the reporting node and back) and are estimated differently depending on the type of confirmation:

- For ACKs: Because ACKs are exchanged only between direct neighbors, the node uses `estimate_expiration_time()`—the same precise calculation used for queries—to estimate the round-trip time, including a built-in safety margin for potential delays.
- For Custody Transfers and Delivery Confirmations: Since these confirmations can involve multi-hop paths and much longer distances (e.g., Earth–Mars), SNORT uses discrete retransmission time values: if the source and destination are not between the Earth and Mars region, the round-trip time is set to 30 seconds—significantly larger than the typical round-trip

time of a couple seconds; otherwise, the round-trip timeout is set to 80 minutes, double the maximum expected delay of 40 minutes.

The transmission reports are created and sent via `send_receipt()`, where the reporting bundle inherits the original bundle's priority and is assigned an expiration time equal to the corresponding estimated round-trip time. This ensures confirmation bundles remain active long enough to complete delivery without being prematurely evicted.

When a confirmation report reaches its destination, the corresponding flags and retransmission times in the bundle's metadata are cleared. For custody transfer reports, however, if the node is also the source of the bundle, it sets *Next Critical Retransmission Time* to the expected round-trip time to the final destination. If the report is the last outstanding confirmation required for the bundle, the node removes it from long-term storage and memory.

3.2.2 Protocol Workflows

The forwarding and receiving workflows define how bundles are selected, routed, probed, and transmitted in a way that maintains reliability, supports confirmation reports, and respects storage constraints. The forwarding workflow ensures the bundles progress efficiently through the network, while the receiving workflow ensures that incoming bundles are properly handled.

Forwarding Workflow

1. **Select bundle**: The node retrieves the next bundle from the queue using `get_next_bundle_to_send()`. A bundle awaiting confirmation is re-queued until its metadata block address becomes invalid, which means the node has received the required confirmation report and cleared the metadata since the bundle was last dequeued. In that case, the node simply moves on. Otherwise, the node checks whether either retransmission time (*ACK* or *Next Critical*) is nonzero. If either retransmission time is exceeded by the current time, the node calls `set_retransmission_times()` to update them based on the expected round-trip times, the bundle is transmitted as normal, while a copy remains in storage. In both cases—whether retransmission occurs or not—the confirmation report has not yet arrived, so the bundle is requeued. If the retransmission time has not yet passed, no transmission is triggered. Otherwise, the bundle is ready for forwarding in the following steps.
2. **Determine next hop**: If the node holds custody of the bundle, it consults the routing table to identify the next custody node. Otherwise, it selects the next best forwarding candidate in the following order:
    - Nodes returned by `next_hop()`.
    - Node of next type along the path (e.g., LEOCom to GEOCom if the bundle is enroute to the Moon) with valid remaining capacity.
    - The two closest neighbors of the same type, considered only if the queue is full, *Remaining Storage* drops below 5%, or the bundle is Priority 3.

    This ensures bundles move forward under pressure while limiting unnecessary transfers, reducing congestion and maintaining reliability.
3. **Probe next destination**: The node queries neighbors in order of the above priority. It uses `estimate_round_trip_time()` to determine the maximum time to wait for a response before

moving on to the next best hop. If no valid responses are received from any neighbors after two rounds of queries, the node requeues the bundle using `return_bundle_to_queue()`.

4. **Transmit bundle**: If only part of the bundle can be sent, the node fragments it with `fragment_bundle(bundle, amount)` and sends the fragment. The remainder is re-queued via `push_fragment_onto_head_of_queue()`. Fully sent bundles with no retained copy have their metadata entry deleted.

Receiving Workflow

System-generated bundles are immediately processed and discarded. For other bundles, the node checks metadata to determine if it's an *Expected Bundle*—meaning space is still allocated for it. If not, the bundle must have arrived after the reservation period expired, so the node must check whether sufficient storage is available (i.e., *bundle_size* < *Remaining Storage*). If so, the bundle is stored, and *Remaining Storage* is adjusted accordingly—this scenario motivates the use of a *Free Storage Threshold*. Otherwise, the bundle is dropped and will be retransmitted due to end-to-end reliability. If it was *Expected*, it is promoted to a *Stored Bundle* and any difference between the actual and *Expected Size* is used to correct *Remaining Storage*.

Once stored, the bundle is further processed to determine its next action. In the simplest case, the bundle does not require any confirmation reports and has not reached its destination; it is added to the appropriate queue. If the bundle is high priority and requires an acknowledgment, one is sent to the previous hop. If the node is a custody node, it must check that the bundle is either not fragmented or that it is the final fragment, by considering the bundle's fragment block in memory. Once either condition is satisfied, the bundle may be added to the appropriate queue, and a custody transfer report is sent. Finally, if the node is the bundle's destination, the bundle does not need to be queued; it is marked as *Recently Delivered* in metadata, and a delivery confirmation is sent back to the source.

3.2.3 Special Behaviors

Redundant Forwarding

To ensure timely delivery, SNORT uses redundant forwarding for bundles with short expiration periods and high priority. Since *small bundles* tend to have short expiration periods (e.g., routine communications) and are never fragmented, per the *valid remaining capacity* requirements, they are forwarded to all available neighbors to maximize delivery success. Large high-priority bundles follow a more selective strategy, forwarding to the top two best neighbors to limit fragmentation and traffic overhead. On account of duplication management and short expiration times allowing quick eviction, congestion is quickly minimized.

Duplication Management

Duplicate bundles may arise due to redundant forwarding or retransmissions when confirmation reports are lost or delayed, potentially leading to network congestion or perpetual retransmissions. To mitigate this, SNORT's duplication management leverages end-to-end delivery confirmations. When a delivery confirmation report is sent from the destination back to the source, each intermediate node along the path marks the bundle as *Recently Delivered* in metadata. Upon doing so, it also calls `schedule_remove_entry(expiration_time, bundle_id)` to remove bundle_id from memory

after expiration_time. The destination node that creates the report determines this expiration_time as 24 hours from the current time. This value is included in the report, allowing each intermediate node to use it when calling `schedule_remove_entry(expiration_time, bundle_id)`.

The 24-hour window is chosen because the maximum expected end-to-end delivery delay is approximately 40 minutes, so one day provides more than sufficient time for all duplicates to be eradicated across the network. SNORT expects that bundle IDs are assigned sequentially, allowing the safe assumption that reuse of a bundle ID will not occur during this window. Since each bundle ID is 4 bytes (supporting approximately 4.3 billion unique IDs), even if a node were producing bundles at an extremely high rate of 1,000 bundles per second, reuse would only occur after about 49 days — far longer than the 24-hour expiration period. Typical expected traffic patterns are much lower, with routine communications generating approximately 1KB of data per minute, making the risk of ID reuse negligible.

During a query, if a neighbor identifies the bundle as a duplicate – either *Recently Delivered* or actively stored on the node – it returns -1 via `storage_available()`, signaling that transmission is unnecessary. In the case where the bundle is *Recently Delivered*, the neighbor sends a nonzero expiration_time, allowing the querying node to also record the *Recently Delivered* entry and call `schedule_remove_entry()`. If the duplicate is merely a *Stored Bundle*, the neighbor sets expiration_time to zero, and no additional memory entry is needed. This check is also performed upon bundle arrival in case confirmation occurred after transmission. While not all duplicates are caught immediately, most are intercepted along the confirmation path, and the destination node drops any remaining ones. If a delivery confirmation report is lost, the query approach ensures that the knowledge of recent delivery still propagates back to the source, allowing it to stop unnecessary retransmissions.

Software Updates

Due to their large size and infrequency, software updates are handled with special logic. We recommend staggering updates across components to ensure continuous network operability. The `query_neighbor(bundle_size…)` function is extended to support software update alerts: if called with bundle_size 0, nodes interpret it as a software update query and refuse other bundles until the update is complete.

For LEOCom, relay, and Moon/Mars equipment updates, Earth antennas send individually addressed update bundles to each node.

For GEOCom updates, 12 pre-designated LEOComs act as intermediaries for a pre-defined software update duration. Earth antennas send the software updates to these LEOComs, which forward copies of the updates to GEOComs with the best available bandwidth from the routing table. During the update window, these LEOComs transmit only GEOCom software bundles and, once the update period ends, report back to Earth with a list of successfully updated GEOComs.

4. Use Cases

4.1 Routine Communications

SNORT supports reliable, timely routine communication between parties in space (e.g., astronauts on the Moon connecting with family on Earth, space station to Ground Control updates). These bundles are typically small and non-urgent, but still require correctness. To maximize delivery success, redundant forwarding (3.2.3) is applied specifically to small bundles, like routine messages, ensuring they reach their destination quickly and reliably. In parallel, protocol-wide end-to-end confirmation guarantees all data is delivered accurately. This enriches quality of life for individuals in space, enhances crew safety, and supports Earth-based operations.

In accordance with NASA policy, routine communication bundles are not usually forwarded to GEOComs, as the forwarding policy limits unnecessary transfers (3.2.2).

4.2 Land Satellite Telemetry

SNORT supports rapid transmission of land satellite data for real-time analysis and disaster response, directly supporting predictive warning systems. Large-scale redundant forwarding (3.2.3) improves reliability and ensures urgent data – like landslide alerts in remote regions – reaches Earth promptly. SNORT prohibits forwarding within the same layer of devices unless urgent conditions apply (3.2.2), guaranteeing that only high-priority, time-sensitive data like landslide alerts are forwarded to LEOComs, while observational data remains on the land satellite until it can reach Earth.

4.3 Solar Telemetry

SNORT transmits critical solar telemetry data using redundant forwarding for small bundles (3.2.3), enabling accurate solar storm predictions and timely alerts about incoming solar flares. Note that the *small bundle* threshold is specifically tuned to accommodate the size of all solar flare warnings, guaranteeing these time-sensitive messages are always prioritized. Reliable solar flare alerts help safeguard both crew health and satellite infrastructure.

4.4 Software Updates

To meet tight update windows without impacting overall performance, software updates are staggered so that only a small portion of the network is updating at any time. Nodes undergoing updates do not respond to neighbor queries during this period, allowing the rest of the network to operate normally and avoiding unnecessary transmissions.

5. Evaluation

To evaluate SNORT, we consider both its expected behavior under typical conditions and its robustness in worst-case scenarios. Our evaluation is structured around three key lenses:

1. Metric Calculations
2. Worst-Case Scenario Analysis
3. Design Priority Reflection

5.1 Metric Calculations

5.1.1 Latency

The time between bundle creation and delivery depends on the path it takes and includes transmission, storage, and fragmentation-related delays. We model total latency as the sum of four key components:

- Transmission Delay, d_t: The time for data to physically traverse each hop, assuming a link is available. Because only one device can transmit at a time, delays may arise from link contention or intermittent visibility. Under ideal conditions, bundles travel at the speed of light, but we conservatively estimate one second per hop (e.g., 10MB bundle from Earth to LEOComm to GEOComm to Moon

$$\frac{10\text{MB}}{2\text{GB/s}} + \frac{10\text{MB}}{1.2\text{GB/s}} + \frac{10\text{MB}}{622\text{MB/s}} \approx 0.03\text{ s} \leq 3\text{ s} = (1\text{ s}) \times (3\text{ nodes})$$

  assuming maximum bandwidth)

$$d_t = \text{hops} \times 1\text{ s}$$

- Queueing Delay, d_q: Bundles may be delayed in the queue, especially if higher-priority traffic arrives. This delay varies with network load. We assume a worst-case estimate of 1 second per hop.

$$d_q = \text{hops} \times 1\text{ s}$$

- Custody Delay, d_c: Custody nodes hold onto a bundle until all fragments are received and a confirmation is received from the next custody node. This delay includes time to reassemble fragments and receive a custody ACK. Since SNORT immediately transmits control-plane bundles and conservatively estimates ACK round-trip-time at 30 seconds (using the transmission delay calculation from earlier, this assumes the bundle travels through 30 nodes, an overestimate), we assign a one minute delay per custody hop.

$$d_c = \text{custody hops} \times 60\text{ s}$$

- Fragmentation Delay, d_f: Large bundles are split into fragments, which may be transmitted over slightly different paths. Assuming the smallest fragment size of 250MB, and that fragments may take two additional hops, we estimate the worst-case delay, where s is the size of the bundle.

$$d_f = (\frac{s}{250\text{MB}}) \times (1\text{ s}) + (2\text{ s})$$

We argue that this is sufficiently fast for any of the use cases. The most time constrained use case is the 100GB software update to Mars, which needs to reach Mars within 30 minutes. There are a total of five hops in the most direct path from Earth to Mars (see Appendix 9.2). It follows that in the most extreme case, there are four custody hops, as the destination node is not a custody node. We are also given that the transmission time between Earth and Mars varies between 4 and 20 minutes: $d_t = 20$ minutes.

$$d_q = 5 \times 1 \text{ second} = 5 \text{ seconds} = 0.0833 \text{ minutes}$$
$$d_c = 4 \times 60 \text{ seconds} = 240 \text{ seconds} = 4 \text{ minutes}$$
$$d_f = (\frac{100\text{GB}}{250\text{MB}}) \times (1 \text{ s}) + (2 \text{ s}) = 402 \text{ seconds} = 6.7 \text{ minutes}$$
$$\text{latency} = 20 + 0.0833 + 4 + 6.7 = 30.7833 \text{ minutes}$$

Although this estimate exceeds 30 minutes, it represents a highly conservative upper bound, assuming both maximum transmission time and full fragmentation overhead. In practice, larger fragments and better link alignment would significantly reduce fragmentation delay. Moreover, SNORT's forwarding logic ensures software updates are still delivered reliably even under high network load or partial node failures by leveraging custody forwarding, update isolation, and priority-aware queuing

5.1.2 Network Overhead

Since SNORT employs ACK-based confirmations, redundant forwarding, and custody, it floods the network with more bundles than the original BP did. Thus, the network overhead can be estimated as the ratio of additional bandwidth usage to total bandwidth consumed:

$$\text{Overhead} = \frac{\text{Administrative} + \text{Redundant} + \text{Extra Data}}{\text{total transmission}}$$

- Administrative: Control plane data, including custody acknowledgements, status reports, and end-to-end confirmations.
- Redundant: Bundle copies from redundant forwarding.
- Extra Data: Anything additional the protocol requires.

For low-priority traffic, there is no redundancy, custody, or status reports, meaning that the only overhead would be from fragmentation metadata. Each fragment incurs an additional 10KB of metadata, combined with the fact that each fragment is at least 250MB, incurs a negligible overhead.

For high-priority traffic, worst case, the bundle will require both custody and hop acknowledgements. On one extreme, if the bundle is extremely large, the administrative and extra data overhead will be negligible (i.e., several megabytes compared to a 10 gigabyte software update), so the overhead will mainly be from however many redundant copies are sent. Let n be the number of redundant copies sent per node and h be the average number of hops. Then, the total number of bundles sent will be n^h, making the overhead $\frac{n^h-1}{n^h} = 1 - \frac{1}{n^h}$. We can calculate the overhead, using $h = 2$, since most bundles are between Earth and LEOComms/GEOComms.

n	**Overhead**
1	0%
2	75%
3	89%

SNORT's forwarding protocol states that redundant forwarding happens to two neighbor nodes for large, high-priority bundles to avoid having too much overhead.

On the other extreme, if the bundle is small, the administrative and extra data overhead will not be negligible. Worst case, the bundle is a solar flare alert: a high-priority, 10MB bundle. Assuming control plane bundles are 0.1MB, there are two bundles at each hop (custody and hop-by-hop ACK), and each copy requires a status report, worst case, the overhead from control plane bundles will be $n^h \times h \times 3 \times 0.1$ MB. Accordingly, the overhead will be

$$\frac{((n^h - 1) \times 10) + (n^h \times h \times 3 \times 0.1)}{n^h \times (10 + h \times 3 \times 0.1)}$$

While small bundles experience a relatively high proportional overhead due to control-plane traffic, this overhead is not a cause for concern in practice. Because these bundles are small in absolute size and infrequent compared to bulk data, their total contribution to network load remains minimal. SNORT prioritizes these bundles precisely because their timeliness outweighs efficiency, and the network is designed to absorb this class of traffic without compromising the delivery of larger, lower-priority data.

5.1.3 Unnecessary Retransmissions

SNORT will occasionally unnecessarily retransmit a bundle if the original transmission succeeds with delayed confirmation. There are two classes of retransmission events.

First, neighbor-to-neighbor communications (e.g., custody ACKs, query responses) are timed using `estimate_round_trip_time()`, multiplied by a factor of two to account for delays or jitter. Because neighbor-to-neighbor communication is typically fast and predictable, the padding is generous, and we predict that unnecessary retransmissions in this category are rare. We can conservatively estimate that fewer than 1% of such retransmissions are unnecessary.

The second category includes custody transfer and delivery confirmations, which often span multiple hops and regions, making them more variable. For these, SNORT uses fixed timeout values instead of estimated RTTs: 30 seconds for transfers within the Earth-Moon system, and 80 minutes for Mars-related deliveries. These values are set based on the wide variability in round-trip times, ranging from a few seconds in local links to 40 minutes between Earth and Mars. Because of this conservative design, custody and delivery confirmations usually arrive before the retransmission timer expires.

For example, in the worst case, even if confirmation reports are delayed as much as expected, the likelihood of SNORT retransmitting a bundle prematurely remains very low. For local custody transfer,

assuming a normally distributed RTT of 20 seconds with a standard deviation of 5 seconds, only about 2.3% of retransmissions would be unnecessary. For Mars-bound delivery confirmations, assuming RTTs average 40 minutes with a 10-minute standard deviation, fewer than 0.01% of retransmissions would be early. These estimates reflect upper bounds, assuming the most conservative plausible distributions of delay, and demonstrate that SNORT's retransmission strategy is robust against timing uncertainty.

5.2 Worst-Case Scenarios

5.2.1 Overlapping Urgent Use Cases

To demonstrate how SNORT performs under simultaneous high-priority demands and partial system failure, we consider the following real-time scenario. This example illustrates SNORT's ability to support multiple use cases at once while maintaining its design goals of reliable transmission and timely delivery.

Scenario Overview
A solar probe detects an anomalous solar loop pattern, indicating a likely flare. It immediately generates a 200MB P3 bundle with a 30-minute expiration, flagged for end-to-end delivery confirmation. At the same time, the network is in the middle of completing a scheduled 16GB software update to all GEOComs. Midway through this process, one of the GEOComs experiences a partial storage failure, losing access to half of its available long-term storage.

System Response
Step 1: Redundant Forwarding and Priority Enforcement
The solar probe's alert, due to its small size and high urgency, is redundantly forwarded to multiple GEOComs. SNORT prioritizes this bundle through the data-plane queue and initiates multiple simultaneous transmissions to maximize delivery success. Because all bundles are marked for end-to-end confirmation, SNORT tracks its status and prepares for retransmission if no confirmation is received before timeout.

Step 2: Maintaining Control-Plane Coordination
Despite the ongoing solar alert and software update, SNORT continues processing routing table updates and confirmation messages through its control-plane queue, which is prioritized above all data traffic. This ensures that the network remains synchronized, avoiding stalling due to routing mismatches or outdated neighbor information.

Step 3: Storage Virtualization and Failure Recovery
When the affected GEOCom loses half of its storage capacity, SNORT's storage virtualization layer absorbs the failure. Because bundles are preferentially stored on the more stable device, the remaining capacity remains usable. If the node is unable to reserve space for the alert, SNORT automatically reroutes the bundle to alternate neighbors or retransmits it later, preventing data loss without compromising delivery speed.

Step 4: Isolation of Update Impact

The software update does not globally stall the network. The GEOCom performing the update temporarily pauses only its own bundle processing, while other LEOComs, GEOComs, and ground antennas continue routing, reserving storage, and forwarding data as usual. This isolated update model ensures that system maintenance does not delay time-sensitive traffic such as the solar alert.

Step 5: Confirmation and Cleanup
Once the alert is successfully delivered to Earth and confirmed, SNORT sends a delivery confirmation report back to the probe. All intermediate nodes along the confirmation path mark the bundle as *Recently Delivered*, dropping any stored duplicates. If other nodes holding duplicates query them for space, they are notified to drop the bundle as well, allowing delivery knowledge to propagate beyond the confirmation path. Once the delivery report reaches the source, it drops the bundle, preventing further retransmissions. This design prioritizes reliability and thorough cleanup without risking premature deletion.

Outcome and Implications
This example demonstrates SNORT's ability to handle multiple critical use cases concurrently, including solar flare alerts, software updates, and unexpected hardware failures, without sacrificing reliability or timeliness. SNORT achieves this through its layered approach: priority-aware queueing, redundant forwarding, storage virtualization, and confirmation-based cleanup mechanisms. Even in worst-case conditions, SNORT ensured successful delivery and avoided system-wide stall, fulfilling the mission requirements of SolarNet.

While SNORT is designed to handle concurrent high-priority use cases through prioritization, redundancy, and storage management, the system may become overloaded and fail to deliver all critical bundles if too many high-priority transmissions occur simultaneously, exceeding available bandwidth and storage across the network. Additionally, if a SolarNet outage severs all paths to a node's neighbors or both storage devices in a node suffer hardware failures, SNORT will be unable to forward or store bundles or confirmations.

5.3 Design Priorities

5.3.1 Reliability

SNORT's primary design priority is reliability. While not all data that SNORT routes is urgent, even low-priority transmissions like emails from astronauts' loved ones are important in their own ways. We'll recap the tradeoffs SNORT's modules make to achieve reliability. Starting in the storage system, we keep all bundles in long-term storage, ensuring data integrity in the face of unforeseen outages. The tradeoff here is slow bundle lookups, but we've added the Bundle Lookup Table to mitigate this (3.1.2). In SNORT's forwarding protocol, we query a node's neighbors and transmit only if they can store some portion of the bundle we want to send (3.2.1). This confirmation process allows us to avoid any unnecessary transmissions. The querying process is an essential factor in SNORT's reliability, but slows transmission time. Additionally, SNORT implements end-to-end confirmation in addition to ACKs and custody transfer (3.2.1) to ensure that bundles reach their destination even if data is lost at some point during transmission. We temporarily increase network overhead until bundles have been confirmed as

delivered, but end-to-end confirmations for all bundles, as well as our other confirmation methods, are essential to delivering on SNORT's reliability. We trade off a lot of speed for SNORT to achieve reliability. While this would mean trouble for the urgent data that needs to be forwarded quickly, we'll next recap SNORT's functionality to ensure timely delivery for these important bundles.

5.3.2 Timely Delivery

SNORT's secondary design priority is timely delivery. We want to ensure that urgent data gets forwarded to its destination before expiration. We've added special cases to show how SNORT routes urgent data without compromising its reliability. SNORT's key functionality to route urgent data is redundant forwarding (3.2.3). During the period of redundant forwarding, network overhead will spike, but this tradeoff is justified by both the urgency and infrequency of this event. After urgent data reaches its destination, SNORT employs its duplication management (3.2.3) to remove the duplicate data that is no longer needed.

6. Conclusion

SNORT enhances interplanetary communication with a system designed for reliability and timely delivery, resilient to harsh, unpredictable space environments. We've discussed the tradeoffs of each of the features we've added, as well as how they help SNORT achieve its design goals. Together, these features ensure dependable and timely data delivery across space.

SNORT is designed to operate efficiently within the current scale of SolarNet. While SNORT's modular protocols (e.g., metadata deduplication, ACK suppression, and storage-aware forwarding) scale well across moderate network sizes, several constraints emerge as the network expands. In particular, the per-node memory overhead for tracking custody, duplicate suppression, and reservation state grows linearly with bundle traffic and node count. Additionally, redundant forwarding may lead to congestion or storage saturation if link capacity is not correspondingly scaled.

7. Author Contributions

The team collaboratively discussed the entire design project and made all major decisions through group discussion before dividing the writing: Cameron wrote the system overview, evaluation, and diagrams, Jennifer wrote the forwarding protocol and metrics, and Kayli wrote the storage system and API functions.

8. Acknowledgments and References

We would like to thank our recitation instructor and lecturer Katrina LaCurts for her engaging lectures and insightful feedback throughout the term, as well as our WRAP instructor Jessie Stickgold-Sarah for her guidance in clearly communicating our ideas. We appreciate the dedication of the entire 6.1800 staff for making this course a challenging and rewarding experience.

9. Appendix

9.1 New and Changed API Functions

Function Call	Description	Relevant Sections
`query_neighbor(bundle_size, neighbor, bundle_expiration, bundle_id, fragment_id=None)`	Sends a request for space to be reserved on the neighbor for the incoming bundle. The input now includes the bundle's ID and expiration time. If the value `0` is queried, then nodes recognize that a software update is about to be transmitted.	3.2.1 Query Handling 3.2.3 Software Updates
`storage_available(storage_available, expiration_time, neighbor)`	Sends a response to the previous `query_neighbor()` request, returning the amount of space reserved (`storage_available`) and time at which the reservation will expire (`expiration_time`). - A `storage_available` value of -1 indicates the bundle is a duplicate and should be dropped. There are 2 subcases: 1) If the duplicate is *Recently Delivered*, `expiration_time` is set to the amount of time the receiving node should retain the *Recently Delivered* entry in memory. 2) If the duplicate is an already *Stored Bundle*, `expiration_time` is set to 0 - A `storage_available` value of 0 indicates that the neighbor cannot store or forward the bundle due to insufficient storage or bandwidth.	3.2.1 Query Handling, Duplication Management
`get_storage_available()`	Returns the amount of space available in long term storage calculated as follows: *Storage Available - Free Storage Threshold*	3.2.1 Query Handling
`get_amount_transmissible(sender, current_size, remaining_capacity)`	This function is called by both the sender and receiver of a user bundle during the querying process. It is first invoked by node A (the queried node), and later by node B (the sender), after A's response but before B transmits the bundle. It returns the most up-to-date maximum size of the original bundle that can be transmitted between the sender and receiver. `sender`: `True` if the caller is node B (sent query), `False` if the caller is node A (was queried). `current_size`: The size of the bundle being forwarded (if `sender` = `False`), or the	3.2.1 Query Handling

	`storage_available` value returned by node A (if `sender` = `True`). `remaining_capacity`: The number of bytes that can be reliably sent to the target neighbor, based on the routing system. The amount transmissible is calculated as follows: - `max_size` = min(`current_size`, `remaining_capacity`, **`get_storage_available`**() iff `sender` = `False` else ∞) - If `max_size` < `current_size`, fragmentation is required. If the bundle can be split into two parts each at least the size of a *small bundle*, the function returns the size of the largest such piece that fits. - Otherwise, it returns 0.	
`estimate_round_trip_time`(`neighbor`, `send_size`, `receive_size`)	Returns the estimated total time required to send a bundle of size `send_size` to `neighbor` and receive a response bundle of size `receive_size` from `neighbor`. The expiration time is calculated as follows: - $T_{RTT} = k \times (T_{A \to B} + T_{B \to A})$ $= 2 \times (\frac{send\_size}{B} + \frac{receive\_size}{B})$ - B = minimum bandwidth of self & neighbor - Returns current time + T_{RTT}	3.1.1/3.2.1 Query Handling & ACK Retransmission
`deallocate_space`(`expiration_time`, `bundle_id`, `fragment_id=None`)	Schedules a timeout at `expiration_time` to release reserved storage if the bundle has not arrived by then. When triggered, the node increments *Remaining Storage* by the bunde's *Expected Size* in metadata and removes the corresponding metadata blocks.	3.2.1 Query Handling
`set_retransmission_times`(`bundle_id`, `fragment_id=None`)	Updates the relevant retransmission metadata values for the corresponding bundle by checking its metadata. If the current time has passed the bundle's *ACK Retransmission Time*, both this value and the *Next Critical Retransmission Time* are updated. *Next Critical Retransmission Time* is also updated if the current time has passed it. The *ACK Retransmission Time* is reset using the result of **estimate_round_trip_time**(), based on the	3.2.2 Forwarding Workflow

	neighbor link and bundle size. The *Next Critical Retransmission Time* is updated based on if the source and destination are between Earth and Mars: - Between Earth and Mars: 60 seconds from the current time. - Not between Earth and Mars: 80 minutes from the current time.	
send_receipt(type, expiration_time bundle_id, fragment_id=None)	Creates bundle report of `type` ACK, Custody Transfer, or Delivery Confirmation with matching priority of the bundle, the provided `expiration_time`, and destination equal to its source. Adds the bundle to the control-plane queue.	3.2.1 Transmission Confirmation
schedule_remove_entry(expiration_time, bundle_id, fragment_id=None)	Removes metadata entry corresponding to `bundle_id` at `expiration_time` to allow for bundle ID reuse.	3.2.1 Transmission Confirmation

9.2 Estimated Hop Counts

Source → Destination	**Estimated # Average Hops**	**Justification**
Earth Satellite → LEOCom	1	Direct uplink
LEOCom → GEOCom	1	Direct link
GEOCom → Relay	1	GEO → Relay
Relay → Mars/Moon Surface	1	Direct link
Earth Satellite → GEOCom	2	Earth → LEO → GEO
Earth Satellite → Mars/Moon	5	Earth → LEO → GEO → Relay → Relay → Mars/Moon