May 8, 2020

Macrocosmos: a NASA ExtraNet Storage Solution

By Anisha Agarwal, Weitung Chen, Nate Fletcher, and Xiao Mao

(anisha24@mit.edu, weitung@mit.edu, nfletch@mit.edu, xiao_mao@mit.edu)
Recitation Instructor: Adam Belay

1 Introduction

With the number of missions to other planets within our solar system increasing, NASA's ExtraNet system aims to transmit data between Earth, Mars, the Moon, and all of the satellites in between. However, ExtraNet faces a problem: storing data and choosing which data to send in a timely manner, maximizing the amount of data that reaches its destination. Currently, ExtraNet's storage system is extremely simple and does not support communication between the storage systems of different nodes. This may result in nodes not always knowing the exact order in which to send the bundles of stored data, which could slow down the entire system. Furthermore, if ExtraNet's storage system is not improved before future expansions to other planets, sending data between nodes could take even longer than expected, especially for more distant planets. Therefore NASA has tasked us with redesigning the storage system to be used on the current nodes in place as well as any future nodes, while providing increased throughput and improving the amount of data that reaches its destination.

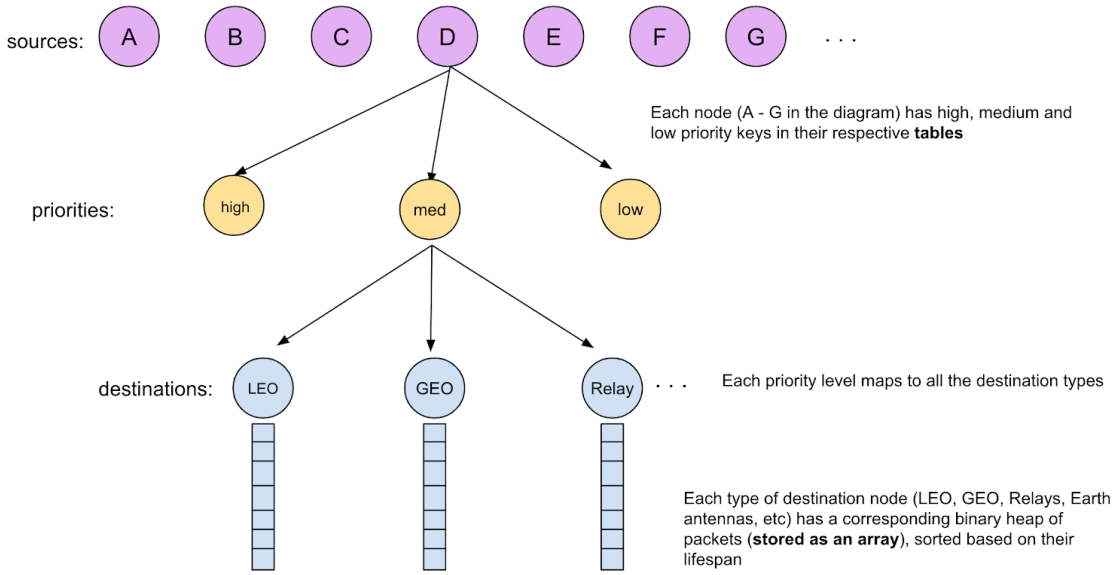
The system that we are introducing, *Macrocosmos*, is able to accomplish the goals mentioned above while keeping at the center of its design our goals of efficiency, reliability, and

scalability. *Macrocosmos* efficiently provides for each node in the system an absolute order of the bundles that must be sent to each other node. This feature is integrated within the storage system of each node, which allows data to be saved and accessed without having to search through the entirety of the node's stored data. The system aims to distribute the data as evenly as possible between nodes so that no single node is overburdened with data while others have a surplus of available storage. By making use of the sliding window protocol and five priority levels, each for specific types of data, *Macrocosmos* is able to reliably send bundles between nodes and ensure with a high degree of certainty that the bundles were received. These features, along with the design of the storage systems of the nodes, do not constrain *Macrocosmos* to the current satellites and relays — in fact, they allow our system to be implemented on a network with any number of nodes. Below we discuss how *Macrocosmos* is implemented as well as its main features.

2 Node Storage Design

The storage system for bundles at each node optimizes based on three main considerations: first, the specification that all bundles from the same source node A must be sent in decreasing priority order. Second, prioritizing packets that are closer to timeout within the same priority level. Third, efficiency of finding packets destined for specific locations. In order to satisfy these constraints, *Macrocosmos* utilizes a three-tiered data structure. The first tier is a hash map that maps source node names to a table with 3 values: high, medium and low. Each of these 3 priority levels, in turn, maps to a hash map with destination types as keys (for example, LEO, GEO, Earth antenna, Relay, etc). Each of these destinations corresponds to a binary heap of packets, sorted based on their lifespan (such that the packet that is closest to timing out is at the top of the heap).

Figure 1: System Overview



Consider a hypothetical packet p. Packet p is from source A, with its next destination being a GEO satellite, and is high priority. It arrives at an arbitrary node N. This will trigger an insertion: p will be hashed to its source node in the high level map, and then sorted to its priority level (high) within the table. Then, it will be hashed to GEO (since that is its target next hop), at which point it will be inserted into the binary heap that corresponds to GEO based on its lifespan. In python code, if we wanted to access the heap into which we just inserted packet p, it would be: node_map[A][high][GEO].

When another node S approaches within communication-distance to node N, N loops through all the keys in the highest level dictionary, searching for any packets destined for node S. The pseudocode is as follows:

```
send_bundle(dest):
     for source in node_map:
          if high is not empty:
               priority = high
          else if medium is not empty:
               priority = medium
          else if low is not empty:
               priority = low
          else:
               return
          bundle_heap = source[priority][dest]
          if bundle_heap is not empty:
               return bundle_heap.root
```

By iterating through each source and only accessing a priority level if there are no packets from any higher level, we ensure that bundles from the same source A are sent in strictly

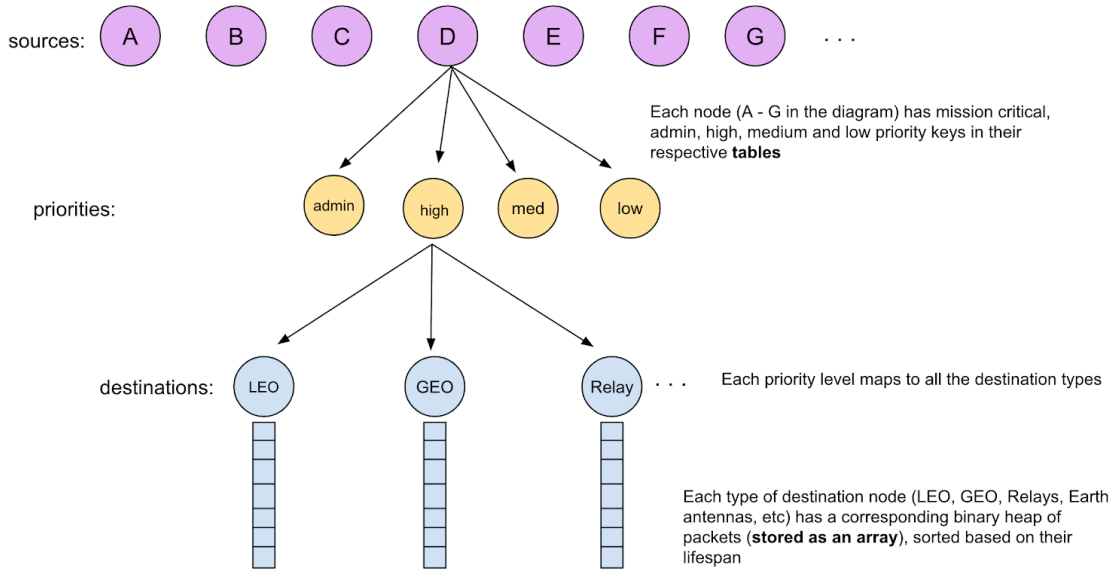
decreasing order. In other words, if packet *j* is low priority and packet *k* is high priority, and both came from source A, *k* will always be sent before *j*.

For storage optimization purposes, the binary bundle heaps are represented as arrays, with the root of the heap being the value at index 0 in the list.

2.1 Custody Nodes

The above storage structure is in reference to regular nodes. Custody nodes have the additional constraint that they must ensure their packets are successfully sent. In order to accommodate this requirement, *Macrocosmos* adds 2 features to all custody nodes: first, we add a separate hash table. This hash table maps a bundle's unique id to the bundle itself. Then, when bundles are sent, rather than being deleted from the heap entirely, they are moved from the heap to the secondary storage hash table. They are deleted from this storage space once the acknowledgment bundle is received. If it is not received within the allotted time for the bundle, then the packet is resent. The second feature of custody nodes is an additional priority level, the "admin" level. This level contains all administrative packets. As the administrative bundles are small and important for communicating acks and storage state information, we assign it the top priority. The packets in "admin" are sent first, but they are also discarded first should the node begin to run out of storage space. This allows the entire custody node pipeline to run more efficiently: information regarding the success of attempts to send packets is disseminated in the most efficient manner, which means packets can be deleted from the secondary storage space as quickly as possible.

Figure 2: Custody node design



This storage structure allows *Macrocosmos* to function well for our main design considerations: time and space efficiency and scalability.

2.2 Efficiency

The hash map with heap structure allows bundles to be inserted in O(1), deleted in O(s*logn), where s is the number of source nodes (which we can assume to be significantly smaller than the number of packets, and therefore essentially a constant). With the use of lists to represent the binary heaps, it takes up O(n) space. Our tiered table/map takes up very little of the 0.5 or 10 TB available at each node, allowing for as many packets to be successfully sent as possible.

2.3 Scalability

Macrocosmos functions very well under the assumption that new nodes can be added at any time. All it must do is add keys to the map and slot into a table, which can be done in O(1) time (constant time to create the source key, constant time to create the 3 different priority keys, and constant time to allocate space in memory for its binary heaps).

3 Protocols to Decrease Lost Packages Due to Storage Limits

3.1 Warning System

To better prevent over saturating the storage of a single bundle node, we use a protocol that tells a bundle node to inform its neighbor when a certain percentage of its storage is used up.

Let's assume an easier model: each bundle node has a fixed set of neighbors. Every time a bundle node uses up 55% of its storage, it tries to issue a "half-full" warning to all its neighboring bundle nodes, and updates its state to "half full." Every time a half-full bundle node's percentage of used storage decreases to below 45%, it lifts its warning and updates its state back to normal. It tries to send a new packet to each neighboring bundle node, telling them it's no longer half-full. Every time a bundle node wants to send a packet, it checks all the possible next hops. If one of the next hops is not half-full, it sends the packet there. Otherwise, if the node itself is half-full, send packets as normal, and if the node itself is not half-full, only send the top 50% of the packets with the highest priority and earliest expiry time. If there is a change of status during the time packets are sent, switch the sending method immediately after the current whole packet is sent.

Our model is more complicated than this since the neighbors of each bundle node is constantly changing, but since the routing protocol always gives the bundle nodes updates on the information about their neighbors(i.e. bundle nodes that are reachable), we can adapt our method to this model with some modifications:

1. Every time two bundle nodes that haven't been reachable to each other become reachable to each other, both bundle nodes will report their current status to each other. This will happen before any packet exchange. Both bundle nodes will assume that the other side is half-full before the other side sends its report.

2. When a change of status occurs, send the update to all bundle nodes that are reachable. Those bundle nodes may need to change the way they are dealing with their packets after receiving the update.

3.2 Gradual Increase of Drop Rate

When a bundle node uses up more than 75% of its storage. It will start dropping the packets it contains. packets with lower priorities and later expiry dates will be dropped first. The drop rate is computed using the following formula:

$$R = (x - 75\%) / (25\%),$$

where x is the current percentage of storage used. For example, if 80% of the storage is used, then R = (80% - 75%) / 25% = 20%. If 100% of the storage is used, then R = 100%.

The way we drop packets is as follows: with a probability of 1 - R, nothing happens, and with a probability of R, a random packet with the lowest priority and the latest expiry date will be dropped.

3.3 Sharing Storage Between LEO and GEO Satellites

An important feature of our storage system is the sharing of storage across different LEO and GEO satellites. We designed a protocol that aims to use the storage of these satellites more thoroughly without causing too much overhead in delivery time or sacrificing any scalability, security.

3.3.1 Intuition

Although each GEO satellite and each LEO satellite only has .5 terabytes of storage, there are 100 LEO satellites and 100 GEO satellites. There's little difference between them other than the positioning, but since those satellites orbit the earth in a small period of time, the same type of packets stored in different satellites of the same kind will get sent in a similar period of time. Therefore, it makes sense to share the storage of those satellites by constantly transferring data between them.

3.3.2 Method

Every 5 minutes a GEO satellite or a LEO satellite will send information about its percentage of storage used to all reachable satellites of the same kind. Upon knowing this information, each satellite will go through all the information it knows and see if any satellite uses more than 1% less storage than it does, if so, it will randomly choose a packet, deliver the packet to that satellite and request for a "packet received" acknowledgement. It deletes the packet when such acknowledgement is received. The sender will continue to send more packets while waiting for acknowledgements.

Moreover, satellites will not take "risky" moves and send a packet to another satellite that is about to become unreachable. This is because doing so may cause the packet to be partially delivered or the acknowledgement to be lost, resulting in duplicate packets and a waste of storage.

4 Evaluation

4.1 System Analysis

4.1.2 Calculations

4.2 Use Case Analysis

To evaluate *Macrocosmos*, we analyzed how our system handles various use cases with different requirements and properties. The properties of these use cases are listed in Table 1. It is shown in the table that most of these applications have a reliability requirement to a certain extent. Therefore, we want our system to be reliable and can recovery from bundle loss due to full storage capacity and transmission failure. The general strategy is that our system incorporates the storage warning system and prevent bundles from being sent to nodes with high capacity. For use cases that don't have a latency requirement and with low priority, it is suggested that the application should request for an ack from the destination. If an ack is not received, then a resend is initiated. On the other hand, *Macrocosmos* handles mission-critical messages with extra care. The system sends the bundles through custody transfer and delivers an ack bundle at each level of transmission to ensure extremely high reliability.

Table 1. Requirements and properties of each use cases

Message Types	Size	Priority Level	Reliability	Throughput	Latency
Routine Personal Msg	1KB - 10MB	Low	Yes	Yes, maintain **1KB/minute**	-
Routine Task Msg	1KB - 10MB	Medium	Yes	Yes, maintain **1KB/minute**	within **10 min of expected time**
Solar Flare Msg	1KB - 10MB	**Mission Critical**	Extremely Yes	-	Lowest
Normal Telemetry Measurements	~20GB for a set of solar data	Low. (High for X-ray data)	Yes	Yes, at least **4GB/minute**	-

Solar Flare Measurements	~20GB	**Mission Critical**	Extremely Yes	Yes, **Highest** throughput	Lowest
Software Updates	1GB - 100GB	Low	Yes	-	-
Streaming Data	0.5MB (4 Megabits) per second	**Mission Critical**	Tolerate 10% data loss	**30MB/minute**	Need low latency sometimes

4.2.1 Routine Communications

Routine communications require the network system to be able to handle small messages that are regularly sent. There are roughly three categories of this type of communication: regular personal messages (such as email) without delivery time constraints, regular task-related messages with 10 min delivery time constraint, and mission-critical solar flare messages that require fast and accurate delivery. *Macrocosmos* is designed to handle all these scenarios, and the following describes how it deals with each message type in detail.

It is stated that regular personal messages should be sent and arrive in a reasonable time frame. In the explanation below, we will assume that 30 minutes is a reasonable time frame for the message to arrive.

Regular personal messages have the lowest priority and long lifetime. It is possible that these bundles are accumulated in a single node and couldn't be sent out quickly, resulting in a resend and extra delay. *Macrocosmos*, on the other hand, provides a more efficient and reliable way to send regular personal messages in parallel with other bundles. If it is detected that a single node is loaded with a lot of messages and the neighbors aren't, the load balancing mechanism will come into action. With the sharing of the LEO and GEO storages, the nodes of our network system can send messages out to another satellite with a lighter load and avoid blocking by higher priority messages. This can increase the possibility of sending out these low priority messages and the reliability of the system. In this case, the application, such as an email app, should request an acknowledgment from the destination to know whether the bundle has arrived. If the sender gets an acknowledgment from the destination within an hour, which is two times the estimated bundle lifetime, we don't need to initiate a resend. Otherwise, we will resend the bundle again.

Task-related messages are prioritized above the regular personal messages. They have only a bundle lifetime of 10 minutes and should be sent out from a node before any personal message. *Macrocosmos* handles this type of message similar to that of the personal messages, but with higher priority. The application should also request for an ack and initiate a resend when the bundle didn't arrive at the destination.

On rare occasions, extremely critical solar flare messages will need to be sent. In this scenario, these messages should be sent under custody transfer with the highest priority, mission-critical. To ensure the highest reliability, custody acknowledgment should be sent to the previous custody node for each level of transmission. Furthermore, due to the unique mechanism that *Macrocosmos* is continuously balancing the load among LEO and GEO satellites, it is almost 100% certain that there will be space on the custody path as long as the entire network is fully loaded. It is this advantage of our system that ensures an available transmission path during emergency scenarios.

4.2.2 Telemetry Transmissions

Telemetry data transmissions are special in the way that it has a throughput requirement, and the data size is relatively large. It is required that the telemetry data be sent from the GEO satellites to Earth for further analysis and logging. This type of communication can be categorized into two scenarios: Normal measurements that are not critical and Mission Critical measurements in the case of the solar flare.

Normal measurement data should be specified as low priority as they are not critical. On average, we need to send solar telemetry data every minute and earth images every 5 minutes. As the measurement is not urgent, the bundle lifetime can be infinite. Also, the application should send an ack from Earth if the bundle has arrived to prevent dropping a specific measurement data. Because the measurement bundles are large and need to be sent frequently, they may be accumulated quickly in a single LEO satellite. The benefit of *Macrocosmos* is that it can share all the storages among LEO satellites and make use of all 50 TB of storages (100 LEO satellites times 0.5TB of storage each). Therefore, the bundles won't be dropped easily due to the full storage capacity.

It is said that the soft x-ray data within the normal measurements is vital to determining the occurrence of the solar flare. Therefore, we would define it as a medium priority. The soft x-ray data will be sent out by GEO satellites before the other normal measurements have been sent out in normal times. This helps people on Earth to recognize solar flare more quickly.

In the case of the solar flare, the measurement data should be sent out as fast and as reliable as it can. Solar flare measurements should be sent through custody transfer and specify the "mission-critical" priority. However, the measurement data may pack a single custody path with the data. Macrocosmos will periodically check the storage in its neighboring nodes and transfer some of the bundles to the adjacent nodes. Then, it will try to initiate another custody transfer starting from that node in the hope of relieving the load of a single custody path. Noteworthy is that we will also need to send the custody acceptance report to the previous node to ensure that the bundle was received correctly.

4.2.3 Telemetry Transmissions

In the case of software updates, we need a guaranteed delivery to the target node, but there isn't a delivery time requirement. Therefore, we will set the lowest priority for software updates and request status reports at each node. Then we can track the report-to node to see whether a specific node has received the update file. After a node has received the update file,

we will try to send out bundles from the queue to the neighboring nodes before initiating the update process. Our load balancing mechanism will ensure that the nearby nodes won't overload. Also, we will need to send out a state change message to the adjacent node before going into the update process, telling others that it will be out of order for an amount of time.

4.2.4 Streaming

Streams are a high-priority, low-bandwidth system of transmitting data between nodes. Applications may elect to send their bundles via a stream for a variety of reasons, such as a satellite wanting to livestream video of a planet it is currently orbiting back to Earth. In this case, as well as in all cases of streaming, this is not necessarily interactive in that the satellite is not expecting a response back from Earth because of the amount of time that the bundles take to travel between nodes as distant as in this case. However, the satellite is still trying to emulate a livestream by streaming the bundles so that the people on Earth can watch the video as it gets received.

A stream first begins at any one of the nodes. This node, using the routing protocol, will then choose the next node in the path, prioritizing nodes that will be within range for a long period of time so as to reduce the probability that it will need to identify a different node to which it can transmit during the stream. While the node waits for a response from the next node informing it that it has enough storage capacity for a transmission, the node stores the stream to its memory. Since streams are transmitted at 4 megabits per second, and with one of the longest round trip time being between Earth and Mars at 40 minutes in the worst case, a failure would mean that the node has to store 1.2 GB of the stream, or about 0.24% of the total storage of a LEO, GEO, or relay satellite, which have the least storage of all nodes in the system. Once the node can begin to transmit, however, this data can be deleted. Each node along the path does this until the stream reaches its destination, at which point all stream bundles are sent along this path.

We have put safeguards in place in order to prevent or reduce the number of interruptions to these streams. First, all of the bundles that make up a stream are sent using the High priority level. While High priority is the third highest priority level that exists in our system, the two that are higher (Mission Critical and Administrative) are not expected to be use so often that they would pose a serious risk to interrupting streams. There is, however, the chance that a higher priority bundle will need to be sent out from a node before it can begin to send out stream bundles. There is not much that can be done in this situation, but the risk to the continuity of the stream is worth ensuring that all nodes receive Mission Critical information that could save a node from being irreparably damaged or an Administrative bundle so that a node could begin to free up its storage. These events do not occur with such regularity that an application should expect its stream to be interrupted.

The second safeguard only comes into effect in the case that the stream lasts for a longer period of time than the amount of time that two nodes in the path are in communication range. We first isolate three nodes in a path that are streaming data such that the first two are about to lose connection with each other. The routing protocol will then notify the first node of other available nodes nearby. Shortly before the two nodes move out of range, the first node will send a bundle to the second node containing the information of the next node to which the first node

will continue to transmit the stream, and then the first node will cease communication with the second node. The second node will then forward that last bundle to the third node in the path, which will inform it that it should prepare to continue receiving the stream from the node specified in the bundle.

To further explain this, we will examine a case where an Earth antenna is currently streaming data which is passing through a LEO satellite (which will be denoted as LEO_1), and then to a GEO satellite (denoted as GEO_1), before continuing on to its final destination. LEO_1 is about to travel over the horizon and out of range of the Earth antenna, which could cause an interruption to the stream. To circumvent this, the Earth antenna, which receives a list of all of the available LEO satellites in range every minute courtesy of the routing protocol, will choose another available LEO satellite (LEO_2) from generally the same area of the sky that LEO_1 was at the beginning of the transmission of the stream in order to maximize the chance that LEO_2 will be in range of GEO_1. The Earth satellite will briefly interrupt the transmission of the stream to send a small bundle containing the unique address of LEO_2 and the destination of which will be GEO_1. Next, the Earth antenna will end its stream to LEO_1 and continue the stream where it left off, now transmitting to LEO_2. LEO_2 in turn will begin transmitting to GEO_1, which will have received the bundle from LEO_1 and will have been waiting for the transmission from LEO_2.

A slightly altered version of this has the Earth antenna streaming to a LEO satellite (LEO_1), which is itself streaming to a LEO satellite (LEO_2). If LEO_1 is about to move out of range of the Earth antenna, the antenna will identify a LEO satellite (LEO_3) that is near the path of LEO_1 and insert LEO_3 into the path of the stream. In both of these cases, using either a replacement or an insertion, *Macrocosmos* attempts to preserve the path that the stream will take so that a different path does not have to be identified in one of these cases. Finding a completely new path for a stream can be costly, since it could take minutes for a node to receive confirmation that the next node in the path is ready for a transmission, and this has to happen for each node in the path. Therefore, *Macrocosmos* tries to limit this so that it only occurs once per stream.

4.2.5 Interoperability

The above use case analysis shows that the majority of traffic in our system will be small administrative bundles, for communicating acknowledgments, load balancing bundles for transferring packets to neighboring nodes, and application-specific message bundles. Also, a significant portion of the storage will be occupied by low priority messages, such as normal telemetry data and software update fragments. On some rare occasions, we will need to prioritize solar flare related traffics in our network. With *Macrocosmos*, the storage spaces of LEO and GEO coordinate with each other and balance the load. It is not until the total storage capacity reaches 75% that our storage system starts dropping packets. Therefore, we are sure that most of the time, our system can reliably transmit and receive the bundles. Given the highest load case when:

- Two people on the Moon and the two people on Mars are sending 10 MB videos simultaneously.

- In the case of the solar flare, we need to send a set of solar telemetry data (20 GB) every minute at the highest priority.
- There's an ongoing software update file transmission to both Mars and Moon devices.
- There is an application that requires the transmission of video streams from Earth to the Moon or Mars.

According to the calculations, if there's no load balancing mechanism and all traffic goes through a single node in the very worst case, that node will be out of storage in 14.9 minutes. However, if we make all the LEO and GEO satellites work together at their level and share all the storages, it will take 30.9 hours to reach 75% total capacity if the network remains in this high load traffic. This proves the reliability of *Macrocosmos*.

Macrocosmos is also very flexible and can quickly accommodate large changes in the use cases. To explain this, we will provide an example of a very different application. For instance, if the application requires a bundle to arrive at a station at a particular time, we can first estimate the transmission time and send a high priority or mission-critical bundle through custody transfer. This guarantees the message to arrive at the destination at the time that we estimated. The primary reason that our system can be compatible with various applications, such as the example given above, is that we maximize the available satellites through balancing the bundles throughout LEO and GEO satellites.

5 Conclusion

Macrocosmos allows each node to store its data in a fast and organized manner. Nodes choose the next bundle to send by searching through its data, which is organized by source, priority, destination, then time until the death of the bundle (if specified) and uses nested hash tables until the final section, which uses a binary heap [2]. These hash tables allow for a constant lookup time, and the heap allows for a constant lookup time, allowing for a node to store and access data in constant time [2.2]. This is significantly faster than the linear store and access times of the naïve approach of searching through all of the node's data. If the node that is chosen to be sent next contains very important information, such as mission critical bundles, our system will send it through custody nodes so that the information does not get lost or dropped [2.1]. Similarly, if a bundle has specified to do so, nodes will attempt to send reception and delivery status reports to report-to nodes [4.2.3]. In order to reduce the amount of data that is dropped due to space restrictions, LEO and GEO satellites attempt to share data as evenly as possible [3.3]. Streaming data is also supported, and our system works to deliver as much of the streamed data as possible, but there are still situations in which a stream may have to be completely dropped due to frequent or extended gaps, perhaps due to higher priority bundles that take precedence [4.2.4]. Finally, *Macrocosmos* was designed not only to support the infrastructure already in place, but also to be able to be extended to future bodies within and, when the time comes, beyond our solar system.

6 Author Contributions

All four team members designed *Macrocosmos* together, and they wrote and edited this report. Introduction, Conclusion, and Streaming by Nate Fletcher. Node storage design and

evaluation by Anisha Agarwal. Organized the use case analysis table in the Evaluation section, Analyze three use cases, Interoperability analysis by Weitung Chen.