Aardvark:
An Autonomous System for Scientific Experimentation on Mars

Dan Ports, Amanda Smith, Sarah Lieberman
May 6, 2004

6.033 Design Project 2

Recitation #9: Karger/Lesniewski-Laas

Contents

List of Figures

List of Tables

1 Design Overview

This report presents a design for an autonomously-controlled system for managing rovers on Mars. It addresses the problems of dealing with lossy communication using a acknowledgement-based protocol, deals with sensor faults by repeating experiments, and schedules experiments and assigns them to rovers using a greedy algorithm that allows for batching of multiple missions, cooperative work with multiple rovers on the same mission, and variable prioritizing of individual missions.

The Mars Rover project can most easily be split into three components: communication protocol, mission distribution, and mission execution. The problems associated with communication protocol are relatively straightforward and include message authentication, error detection, and the location of other rovers within broadcast range. Also, the section deals with authentication to avoid confusion from Poodles, and ways to make communication more reliable. Mission execution includes the handling of sensor failure and ways for multiple rovers to coordinate on missions. Also, we discuss an idea for how to make data more likely to make it back to central control. Mission distribution encompasses a wider range of issues. Most important, there is the basic algorithm for scheduling missions and distributing those missions among the rovers. Also, it describes design details that are intended to deal with rover failure, amendments, and the scheduling of data upload.

2 Communications System

The most important communications between the devices on Mars are the distribution of missions and the transmission of mission data. The network layer is already specified as a "black box" with the functions SENDMSG(*msg*, *length*) and DELIVER_MESSAGE(*packet*, *length*), so only packet segmenting must be dealt with. The link layer is also specified, because it is just a broadcasting radio receiver, although there is an issue of how to detect nearby devices which are available to receive messages. Thus, the majority of the communications design is in the end-to-end layer.

2.1 Protocol Design

The first task is to design a standard packet protocol. The Mars devices follow this protocol for the format of the packets (in the end-to-end layer; the network and link layers may, of course, add headers and trailers):

- Device name
- Message nonce
- Message type
- Message arguments (for message types one, five, six, seven, and ten)
- Signature
- Checksum

Device name The first part of the packet, the device name, is trivial. The rovers can be given numbers 1-50 as their names, and the command center can have the name 0, thus providing each device with a unique identifier.

Message nonce The nonce is a unique integer chosen by the rover or command center randomly. Since there are at most fifty-one messages being broadcast at the same time, a nonce can simply be an integer of several bytes.

ID	Message Type
01	Acknowledge
02	"Who's here?"
03	Begin communication
04	End communication
05	Mission assignment
06	Mission data
07	Data / assignment size
08	Not enough memory
09	Already have a mission
10	Resend message

Table 1: Message types

Message type and arguments The message type is a single integer specifying the type of message, such as an acknowledgement or error (see Table 1). Message type one, the acknowledgement, and message type ten, the request to resend, are followed by the nonces of the messages to which they refer. Message types five and six, which flag mission assignments and data respectively, will be followed by the expected sort of message in a standard format (specified in the section on mission distribution). Messages type seven, which flags a data size message, is followed by an integer specifying the amount data the device wishes to send in bytes. This is so the device receiving the data or assignment will know how much memory to allocate, or send a type eight message in reply if there is not enough memory. The rest of the message types do not require additional arguments.

Signature The next item, the signature, is to provide authentication so that the messages from Poodle rovers will not be mistaken for messages from NASA rovers. Since the Poodle rovers are not malicious, there is no need for message encryption or protection against attacks. The most logical solution is to have all devices sign their messages using a shared secret key. The only requirement for this key is that it is different from any key the Poodle rovers might be using; a 54-bit key would be more than enough protection. Finally, the packet is ended by a SHA1 hash of the message, used as a checksum. Whenever a rover receives a message, it checks the signature and recalculates the checksum of the message. If the signature is not the correct, expected signature of another NASA device, it disregards the message. If the checksum doesn't match, it sends the rover a type ten (resend message) message. The checksum is checked first, because there may be an error in the signature.

2.2 Message Delivery Procedure

In delivering a message from one device to another, the first step must be ensuring that another device is in range of the radio. This is accomplished by sending a type two ("who's here") message. Any device which receives this message will broadcast an acknowledgement (ack; signified by type one). If the device doesn't receive an answer and expects that it should (for instance, if a rover knows it is in range of the command center), it can re-broadcast the query after a reasonable period of time until it receives an ack. Repeated queries and acks cause little overhead, and a rover need not keep any state except for the knowledge that it is expecting a query reply. Note that a rover will not move while communicating with another device. This will prevent rovers from moving out of radio range while another device is attempting to communicate with them. When a suitable device has been located in radio range, mission data can be transmitted.

2.2.1 Rover – Control Center Communications

The command center will wait for queries. When a query is received, the command center acknowledges the query, and the rover will send a message of type three (begin communication). The rover will not move until the communication is finished. The rover will then send a type seven message along with the size of the data it plans to transmit, which the command center will acknowledge.

After receiving an acknowledgement, the rover will begin transmitting the data in 1500-byte packets, each flagged type six. The data will simply be segmented into pieces sized 1500 bytes - (size of packet headers and trailers). The rover will only transmit one packet at a time and wait for an acknowledgement or a request to resend a packet before sending a new one. If it does not receive an acknowledgement or resend request containing the nonce of the message it just sent after a timeout period (for instance, two or three times the average amount of time necessary to send, receive, and acknowledge a 1500-byte packet), it discards that nonce and resends the packet with a new nonce. Thus the devices only keep one piece of state (the current nonce) per communication.

The fact that devices only transmit one packet at a time, and wait for an acknowledgement before transmitting a new packet means that each device receives packets in order, and so no order information need be included with a packet of data. Since message transmission time is trivial compared to traveling and experimentation time, using more complicated protocols to transmit multiple packets would add complexity without much performance benefit.

A rover will continue transmitting its mission data until it has received an acknowledgement for the last packet. It will then send a type four (end communication) message. The command center can either acknowledge this, in which case the rover will sit idle until the command center contacts it (using the same query/begin system described above), or send a type seven (data / assignment size) message, containing the size of the mission assignment it wishes to send to the rover. This is also how the command center contacts rovers when they first reach Mars. The rover will acknowledge the assignment and allocate memory space, unless it already has a mission (an error condition which should not happen), in which case it will send a type nine message. It should have enough space free, since it just uploaded all of its data to the command center, so it can delete data starting with the oldest. The command center will transmit its mission assignment using the same protocol that the rover used to transmit its data. After it has finished transmitting the mission assignment, the command center will send a type four (end communication) message, and the rover will respond with an ack. This frees the rover to begin its travel to the mission site.

2.2.2 Rover – Rover Communications

Two rovers can share mission results with each other (described in Section 3.3). This is accomplished using exactly the same protocol as a rover sharing data with the command center, except that a rover can, after receiving the "data size" message, send a message stating it does not have enough memory for the data, at which point the other rover can decide to send only part of its data and try again, or terminate communication

2.2.3 Control Center – Earth Communcations

. The other special case is communication between the command center and Earth. This also follows the same protocol as described above, except that the command center does not wait for acknowledgements before sending the packets. Earth will know how many packets to expect upon receiving the "data size" packet, so the command center can simply number each packet it sends. It can even send packets multiple times, defending against packet loss, if it runs out of new data to send during a transmission time. The command center will not delete data it holds until it receives acknowledgements for every non-duplicate packet, thus the command center must keep extra state for communications with Earth. It will resend packets to Earth as necessary.

2.2.4 Failures

If a rover fails in the middle of a communication, then the device at the other end will notice after a suitably long period of time that it has not received any messages. It will then attempt several queries. If it times out again, then the device will assume that the rover has died. If the rover was communicating with the command center, it will be marked dead; if the rover was communicating with another rover, then that rover will simply move on to other tasks.

3 Mission Execution

Each rover is capable of autonomously executing missions according to parameters received from the control center. It does so by maintaining a queue of instructions, known as an *assignment*. There are three types of instructions:

- travel to a certain location
- perform an experiment (e.g. taking a photo)
- return to the control center

Field	Type
Instruction number	integer
Instruction type	2-bit instruction typecode (see above)
Arguments[a]	variable
Required repetitions[b]	integer
Associated mission ID	integer
Associated mission revision number	integer

[a]Destination to travel to, or type of experiment to perform and experiment parameters, depending on instruction type

[b]For perform-experiment instructions only

Table 2: Instruction data format

The rover also maintains a table of results. Each result contains metadata that identifies the experiment, the mission the experiment was performed for, and the rover performing the experiment, as in Table 3. This ensures that it is possible to identify which experimental results are unique, even when different rovers perform the same experiment or deliver the same results back to the command center.

Field	Type
Experiment	type of experiment and arguments
Experiment location	location
Experiment timestamp	time
Associated mission ID	integer
Associated mission revision number	integer
Performing rover ID	integer

Table 3: Result data format

The rover proceeds linearly through its queue of instructions, executing the first instruction and removing it from the queue once completed. To perform experiments that require multiple repetitions, the rover performs the experiment until it has gathered the correct number of copies of

the result in its result table. After each experiment execution is completed, it exchanges experiment results with any other rovers that are in range (i.e. other rovers performing the same experiment), and incorporates any of their results into its result table. This allows for multiple rovers to work cooperatively on the same mission (Section 3.2).

3.1 Sensor Failure

When a rover performs a experiment, there is a non-zero probability of sensor failure. We assumethat when a sensor fails, the faulty readings are uniformly distributed over the possible sensor range, and thus there will be no pattern in bad readings. Thus, when readings match, it is more likely that they are both correct than that they are both wrong. The number of readings that would have to agree in order to be confident in their value depends to some extent on the probability of failure. So, to ensure that a correct result is returned with 95% probability (or whatever certainty rate the scientists deem reasonable), the rover will repeat the experiment a certain number of times.

We will use the thermometer as an example for how this number can be computed.

Example The thermometer's fault rate is given as 7%. If we merely took 2 readings, then if one or more of the readings were bad there would be no way to tell which was correct. However, if we took three readings, this would give us a 98.60% chance of having at least 2 good results. Therefore, we could just send all 3 readings back in the data, and have a 98.60% chance of giving the scientists usable data. This number was obtained by finding the probability of having 2 good results, plus the probability that all 3 readings are good. With n experiments and a probability p of success, the probability of having k successes is given by the following formula:

$$\binom{n}{k} p^k (1-p)^{(n-k)}$$

And so the probability of having *at least* k successes is

$$\sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{(n-i)}$$

So, for each sensor, we just need to calculate beforehand a value of n that will give us a sufficient chance of having at least $k = \lceil \frac{n}{2} \rceil$ reliable readings. This approach requires only knowledge of the failure probability p. The rover itself does not have to analyze data in any way. The advantage of this approach is simple: the rover does not need to have any knowledge of how to interpret the mission results. Instead, the results can be evaluated by the scientists on Earth, who are presumably better equipped to identify and analyze correct results. If they deem the experiment results to be unsatisfactory, the mission can be re-inserted into the queue of missions to perform, forcing the mission to be repeated.

3.2 Rover Cooperation

In some cases where an experiment must be repeated, multiple rovers may be sent out together. In this case, when a rover finishes an experiment, it will send its results to all the other rovers within communications range, and receive any results from the other rovers. When a rover has all n required data points, it will move on to the next instruction, which is probably to return to central control. If one of the rovers fails during this time, then the other rovers will not be affected because they are not waiting for any responses from that rover.

3.3 Data Exchange

Finally, if two rovers should happen to pass within communication range in the course of their travels, each rover will transmit any experimental data it has collected to the other. If a rover does not have enough room for the additional data, then the data from the other rover will simply be discarded. The probability of rovers running into each other randomly is small, given the number of rovers, the size of the crater, and their relatively small communication radius, but the probability of meeting is non-zero. In particular, there may be points of interest in the crater which the rovers pass by disproportionately often: locations near which many experiments are performed, or on paths to and from the control center. Having duplicates of experimental data being carried by other rovers can only help the chance of that data being brought back to the control center, and does not have any negative side effects. If one rover fails before returning to the control center, the data may make its way back via one of the other rovers. This extra layer of redundancy might not have a very large effect on the overall performance of the system, but it will have some positive effect for very little cost, so therefore it is worthwhile to implement it.

The one confusion that might arise from this aspect of the design is the possibility of multiple copies of an experiment reaching control center. If the data for a given experiment is already in the result tables, and is listed with the metadata, then the new data should be interchangeable for the old, and thus unimportant. Therefore, it can be simply discarded.

4 Mission Distribution

The mission distribution subsystem constitutes the logic that runs on the control center. It receives missions and amendments from Earth, and efficiently allocates them to rovers. In addition to basic distribution, it accounts for possible rover failure, schedules experiment repetition to account for sensor failure, and allows for mission amendments.

The mission distribution subsystem allocates *assignments* to rovers. An assignment is a sequence of instructions for one rover that must be executed in order, i.e. “travel to location 37.33, -121.03; take a photo; travel to location 37.45, -120.82; take another photo; return to the control center”. The instruction format is described in Section 3 and Table 2. Each assignment is transmitted by the control center to a rover, which then executes the assignment and returns the results to the control center. The process by which rovers execute assignments is described in detail in Section 3

4.1 State Information

In order to manage the allocation of missions to rovers, the central controller must keep track of the current state of the system at all times. To accomplish this, it maintains two state tables: a table of rover states, and a table of mission states.

The table of mission states primarily associates a mission ID with the list of *tasks* required to complete the mission. We define a task to be a tuple consisting of an experiment, and the location at which it needs to be performed; in addition, we associate with each task record a list of the assignments that involve it. We also associate additional data with each mission, as in Table 4. To deal with amendments, as in Section 4.5, we give each mission record a revision number. We also track dependencies — missions that must be completed before this one – for each mission; since we split scattered and scattered-cluster missions into their individual components (see Section 4.5), this ensures that the components are executed in order. Finally, we introduce the notion of a *priority value* for the mission, which captures the relative importance of each mission. This is used in the scheduling algorithm, and is described extensively in Section 4.2.1.

The table of rover states associates each rover’s ID number with its current status. There are four possible states each rover can be in:

Field	Type
Mission ID	integer
Revision number	integer
Tasks	list of task records (experiment/position/assignments)
Dependencies	list of mission IDs
Priority value	integer

Table 4: Mission state table fields

- *available*, meaning it is present at the control center and waiting to be given an assignment
- *busy*, meaning it is currently executing an assignment
- *lame*, meaning it is currently executing an assignment that has become moot due to a mission amendment
- *dead*, meaning that the control center has not heard from the rover and has presumed it to have failed

For rovers that are not in the available state, we associate with them their current assignment (an assignment record), and the time at which it was assigned. We also compute the expected return time, which is the sum of the start time, the required travel time for the assignment, and the sum of the predicted times required to complete each experiment.

Field	Type
Rover ID	integer
Rover state	2-bit state type (see above)
Current assignment	assignment record
Start time	time
Expected return time	time

Table 5: Rover state table fields

4.2 Mission Scheduling and Assignment Algorithm

The control center assigns missions to available rovers. It does so by periodically checking which rovers are in the available state: at regular intervals, it sends out a "who's-here?" message. It gathers responses from the nearby rovers, and once a short timeout period has elapsed, begins assigning missions to any nearby rovers that are in the available state, according to the rover state table.

Mission scheduling is performed by a greedy algorithm. Our algorithm is designed to meet the principal design objective of *maximizing useful work*. We assume that rovers will fail using a memoryless failure model having a mean time to failure of 100 days, and this failure is independent of whether the rovers are active or idle. Hence, in order to complete the maximum amount of work before all rovers fail, it is optimal to have all the rovers doing as much work as often as possible; idle rovers are "wasted resources."

The basic greedy algorithm is augmented by three additional concepts. Priority values are used to express the relative priority of missions; this scheme allows the most valuable missions to be executed first. If beneficial, multiple missions may be *batched* and executed sequentially by the same rover. Finally, if additional rovers remain idle with no missions to execute, they may be assigned to cooperate with some other rover on an assignment that can be executed in parallel.

4.2.1 Priority Values

In particular, we would like to accomplish the most useful work as quickly as possible. This concept of relative utility is expressed by the priority value associated with each mission. This is an extension to the design specifications, but it has several important properties that make it a natural addition. The first is the very likely possibility that all missions are not of equal value. Second, even in the original non-prioritized case, it neatly resolves the ambiguity of whether each mission has the same value, or whether longer missions are inherently more valuable than shorter ones: in the former case, each mission can be assigned the same priority value, and in the latter case each mission can be given a priority value proportional to its length. It also allows NASA to provide low-priority experiments that will be executed only if rovers are available, which can help ensure that rovers are never idle. Finally, though we do not explicitly address starvation in this design (we presume that higher-valued missions really are valued higher, and hence should be prioritized over others), it is easy to imagine a solution that would address starvation by incrementing the priority values for missions that have been in the queue longer.

Accordingly, we rank available assignments in terms of the quotient $\left(\frac{\text{priority value}}{\text{required time}}\right)$, and assign them greedily. We define a procedure Greedy-Select that takes a list of assignments, ranks them, and selects the highest-ranked. We will use this to iteratively assign the mission with highest quotient to the first available rover, and eliminate it from the available mission pool.

4.2.2 Batching

We also allow for *batching* of multiple missions when it is advantageous. That is, we may consider having a rover execute one mission, then proceed to execute a different mission without returning to the control center first. The advantage of this scheme is obvious: if the two missions take place close together, more work can be accomplished in the same time, since we avoid the overhead of having to travel along the possibly-circuitous route from one mission site back to the control center then to the second mission site. The drawback is that the rover will be away from the control center longer, and the probability that it will fail while executing its assignment increases accordingly. Moreover, a rover failure has the potential to be more catastrophic because it can cause the data for *all* the missions it is executing to be lost, not just a single mission.

We attempt to quantify the benefit or cost of a particular batching assignment of two missions a and b using this reasoning, in the following equation:

$$\begin{aligned}&\textsc{Travel-Time}(\textsc{End}(a) \rightarrow \textsc{Start}(b)) - \quad \textsc{Travel-Time}(\textsc{End}(a) \rightarrow \textit{control-center} \rightarrow \textsc{Start}(b))\\ &> \textsc{Work-Required}(a) * \textit{failure-probability} *\\ &(\textsc{Travel-Time}(\textsc{End}(a) \rightarrow \textsc{Start}(b)) + \textsc{Work-Required}(b)\\ &+ \textsc{Travel-Time}(\textsc{End}(b) \rightarrow \textit{control-center}) - \textsc{Travel-Time}(\textsc{End}(a) \rightarrow \textit{control-center}))\end{aligned}$$

This equation takes into account the benefit of the time saved by traveling from a's endpoint directly to b, rather than through the control center, and weighs it against the product of the probability that the rover will fail in the extra time it spends performing mission b, with the work from mission a that would be lost in such a failure.

Before performing mission assignments, the mission distributor on the control center first examines all pairs of available missions, and evaluates whether it would be beneficial to batch them using this metric. If so, it joins the two, and then repeats the process to see if there are any other missions it would be beneficial to batch with the pair. When we join two assignments together, we sum their priority values; this reflects that the new assignment will be doing more useful work. This process is repeated until no further beneficial batchings remain.

4.2.3 Cooperative Execution

Our scheduling approach is optimized for the case in which there are more unassigned missions than available rovers. In this case, it can assign one rover per mission (and sometimes one rover per multiple missions, when batching can be performed), selecting the missions that provide the greatest benefit the most rapidly. This process ensures that the most limited resource, rover time, is allocated effectively.

It is reasonable to assume that more missions will be available than rovers. First, it is always possible for NASA to upload more missions to ensure an ample supply is available. Since we divide scattered and scattered-cluster missions into their component parts (Section 4.5), the mission distributor actually has more missions in its queue than were transmitted by NASA. Finally, as rovers inevitably fail, the number of remaining rovers will dwindle, so that eventually there will be more missions than rovers, even if originally more rovers than missions were available.

However, a recent amendment to the specifications proclaimed that at any given time there would be approximately 25 missions either being executed or waiting to be distributed. Hence, when we have a mission available that requires some experiment to be performed repeatedly in order to achieve the desired failure probability, we allow for multiple rovers to be cooperatively assigned to the same mission.

Though this is not optimal in terms of maximizing useful work done, since both rovers will incur the overhead of traveling to the appropriate location, it allows missions to be performed more rapidly. Additionally, the multiple rovers provide redundancy: if one rover fails while performing the experiment, its previous results will have been replicated onto the other rovers, minimizing the amount of result data lost. It is certainly appropriate to perform cooperative execution if idle rovers are available at the control center, since their time would otherwise go to waste.

The algorithm in Figure 1 is used to determine whether to assign rovers cooperatively to a mission assignment. First, there must be rovers available and no outstanding unassigned missions. Next, each assignment that has been previously assigned to another rover is considered. The assignment must have fewer rovers executing it than the maximum number of repetitions required for any experiment in the assignment. The algorithm also takes into account how long it has been since the previous rovers were assigned to that mission: if the rovers had been assigned a long time in the past, they may be almost complete executing the mission, and there is no point to assigning more rovers to it. Hence, we take the sum of the times since all previous rovers had been given that assignment, divide it by the duration of the experiment requiring the most repetitions, and subtract that from the maximum number of repetitions required. If the resulting number of repetitions is greater than 1, the assignments will be considered for cooperative execution. If multiple assignments satisfy this criterion, the allocation will be performed greedily using the same metric as for normal assignments.

```
ASSIGN-COOPERATIVE-ASSIGNMENTS()
1  while rovers are available and no missions are unassigned
2        do available-assignments ← {}
3           for assignment in busy-rover-assignments
4                 do x ← MAXIMUM-NUMBER-OF-REPETITIONS-IN(assignment)
5                    x ← x − NUMBER-OF-ROVERS-CURRENTLY-EXECUTING(assignment)
6                    if x > 1
7                       then available-assignments ← available-assignments ∪ {assignment}
8           selected-assignment ← GREEDY-SELECT(assignments)
9           ASSIGN(selected-assignment, first available rover)
```

Figure 1: Cooperative assignment pseudocode

4.2.4 The Algorithm

The mission distributing algorithm is presented in Figure 2. The mission distributor first takes the list of available mission assignments, and batches them together whenever appropriate. It then ranks the assignments in terms of the quotient of their priority value and the required time-to-execute, and greedily assigns them to available rovers. If all missions are assigned and additional rovers remain, they will be considered for cooperative assignment execution (as described above).

```
DISTRIBUTE-MISSIONS()
 1  ▷ Initialize the list of possible assignments from the available missions list
 2  assignments ← {}
 3  for mission in unassigned-missions
 4       do assignments ← assignments ∪ {mission.tasks}
 5
 6  ▷ Batch assignments together when beneficial
 7  ▷ Beneficial batchings are identified using the equation in Section 4.2.2
 8  while IDENTIFY-BENEFICIAL-BATCHINGS(assignments) ≠ ∅
 9       do {a, b} ← FIRST(IDENTIFY-BENEFICIAL-BATCHINGS(assignments))
10          batched-assignment ← JOIN-ASSIGNMENTS(a, b)
11          assignments ← assignments ∪ {batched-assignment} \ {a, b}
12
13  ▷ Greedily assign rovers to missions
14  while rovers are available and assignments ≠ ∅
15       do selected-assignment ← GREEDY-SELECT(assignments)
16          ASSIGN(selected-assignment, first available rover)
17          assignments ← assignments \ {selected-assignment}
18
19  if rovers are still available
20       do ASSIGN-COOPERATIVE-ASSIGNMENTS()
```

Figure 2: Mission distribution algorithm pseudocode

4.3 Result Collection

The control center handles the task of retrieving mission results from rovers that have completed their assignments, and transmitting this data to NASA on Earth. When a rover returns to and locates the control center, it begins a transmission to the control center. It then transmits the contents of its mission results buffer to the control center. This is a sequence of mission results, each tagged with appropriate metadata: the location and type of experiment performed, the mission ID and revision number of the mission that the experiment was performed for, the ID number of the rover that performed the experiment, and the time at which the experiment was performed.

When a mission result is received, the control center adds it to a queue of results awaiting transmission back to Earth. The only exception is if the same result — with identical metadata, including the performing-rover and timestamp — is already in the queue. In this case, the duplicate result is simply discarded. In all other cases, the result is queued for transmission. We do so even if the result is presumed to be invalid due to a sensor fault, since the scientists at NASA may wish to attempt to recover meaningful data or diagnose the faulty sensor for it. Similarly, we also return results even if they may no longer be meaningful due to an amendment, since the results may still be useful and it seems wasteful to simply discard them. In doing so, we make the assumption that

there is adequate bandwidth (albeit high latency and limited availability) to transmit all the data; if this is not the case, it may be necessary to discard presumed-invalid results.

The control center is periodically queried by NASA on Earth to see if there are any mission results available. If so, it transmits all available mission results. Once the results are acknowledged, they are deleted from the control center's queue. If all results pertaining to a mission have been completed and transmitted to Earth, and no rovers are currently assigned to that mission, the mission may be considered complete and removed from the mission status table.

4.4 Handling Rover Failures

Rovers can fail unpredictably. The control center can only detect failures when it fails to receive any response from the rover. There must therefore be a time after which the control center assumes the rover has failed if no response has been received. We compute this time by summing the required travel time and the time required to perform and repeat the experiments. We then apply a "slack" scaling factor that accounts for inaccuracies in travel and experiment times estimate. The value of this scaling factor depends, of course, on the quality of the estimates being used, and should be chosen accordingly.

The control center periodically iterates through the list of rovers, and checks whether the difference between the current time and the time that rover's mission was assigned is greater than the maximum time previously mentioned. If the mission has not been completed by this time, it is marked unassigned and the rover assigned to it is marked dead. If it turns out that the rover was in fact still operating, and it returns after being previously marked dead, the experiment results will be downloaded, the mission marked completed, and any new rover assigned to that mission treated as lame, as though a mission amendment had been received. The second set of mission results will still be returned to Earth, as described in Section 4.3.

We originally considered the possibility of sending a "interceptor" rover to travel to the location of the lame rover and inform it that its mission has been made moot by an amendment. The two rovers could then either return to the control center for a new assignment, or proceed to execute the next mission. Though this could conceivably eliminate some of the wasted time used by lame rovers that continue to execute missions that are no longer relevant, it is fraught with complexities and we concluded that it would be impractical. First, it is not always beneficial to send an interceptor rover; if the lame rover has almost completed executing its mission, the minimal time savings will be nullified by the time required to send the interceptor. More troublesome, it is non-trivial to correctly deliver a message. The location of the target rover must be ascertained, and this becomes impractical because of the number of variables involved. The travel time and experiment execution times are only estimates, so it is impossible to compute with complete certainty where the rover will be found at a certain time. If the interceptor does not find its target rover where it was expected to be, it cannot determine whether the rover has failed, or whether it has simply finished its earlier tasks more quickly or slowly than expected. In light of these complexities, we opted not to use this approach.

4.5 New Missions and Amendments

When a mission is received, it is added to the mission table in the unassigned state. Scattered and scattered-cluster missions are broken up into their individual or cluster components, and these are treated as separate missions. Dependency lists, as described in Section 4.1 are used to ensure that the individual components of each scattered mission are executed in order. This allows rovers to return to the control center to offload their data between parts of the scattered mission, minimizing the amount of data that will be lost if the rover fails. The tradeoff here is between the time that will be lost traveling back to the control center instead of directly between the two mission locations, and the expected increase in the amount of work that will need to be redone if the rover fails. Fortunately,

this tradeoff is accounted for by the batching algorithm (Section 4.2.2). If it is ultimately beneficial for a rover to travel directly between two mission locations, it will do so; otherwise, it will travel first to the control center. The priority value of a scattered or scattered-cluster mission is divided among the new component missions, proportionally to the amount of work in each mission.

When an amendment is received from Earth, the mission distributor updates the mission table with the new parameters and increments the mission's revision number. If the mission is still unassigned, no further processing is necessary; the newly-revised mission will be distributed by the assignment algorithm when it is appropriate to do so. If the mission has already been assigned or completed, then the modified portions of the mission will be rescheduled as though it were a new mission. Any rover that is currently executing the old revision of the mission will be marked "lame". If the lame rover completes the old mission, the revision number information will indicate to the control center that it is an outdated revision and is not satisfactory for the purpose of marking the mission as completed. The results, however, will still be sent back to Earth, since they may still be of some value.

If any rovers are currently in the available state when new missions or amendments are received, the mission distributor will re-run the assignment algorithm to appropriately assign the new missions to the available rovers.

5 Conclusion

This design should be able to ensure that resources are efficiently allocated in order to ensure that as much valid scientific data as possible gets sent back to the scientists at NASA. In order to achieve this goal, we took into consider the various complications that could hinder the process. These complications include mission amendments, the failure of sensors and rovers, and poor communications. The mission distribution algorithm is made flexible by the variable amount of rovers that can get sent out to perform a mission. This flexibility is an asset because it allows the system to efficiently allocate resources under a wide range of work loads. Also, the priority values that can be assigned to missions make the overall system fairly customizable. With simple changes in the procedure for value assignment, the overall system can be customized to suit a variety of needs. Some of the design choices we have made depend on assumptions that would be valid in the real world, whereas other assumptions might only be valid given this design projects model of the world. However, we believe that we have kept these assumptions to a reasonable level and that within the world of this design project the design would soundly deal with the issues with which it is presented.