# Distributed Network Scheduling

**Bradley J. Clement, Steven R. Schaffer**

Artificial Intelligence Group
Jet Propulsion Laboratory
4800 Oak Grove Drive, M/S 126-347
Pasadena, CA 91109
Bradley.J.Clement@jpl.nasa.gov, Steven.R.Schaffer@jpl.nasa.gov

**Abstract.** Distributed Network Scheduling is the scheduling of future communications of a network by nodes in the network. This report details software for doing this onboard spacecraft in a remote network. While prior work on distributed scheduling has been applied to remote spacecraft networks, the software reported here focuses on modeling communication activities in greater detail and including quality of service constraints. Our main results are based on a Mars network of spacecraft and include identifying a maximum opportunity of improving traverse exploration rate by a factor of three; a simulation showing reduction in one-way delivery times from a rover to Earth from as much as 5 to 1.5 hours; simulated response to unexpected events averaging under an hour onboard; and ground schedule generation ranging from seconds to 50 minutes for 15 to 100 communication goals.

## 1 Introduction

This paper focuses on issues of autonomously adapting communications for a remote network of spacecraft. If the spacecraft are designed to always guarantee available resources (e.g. transceivers, memory, power) to autonomously redirect communications on the fly, then in order to make such reactive communications autonomous, all that is needed onboard are control sequences for switching transceivers and slewing and routing algorithms for intermittent communication opportunities (Akyildiz *et al.* 2003). Such spacecraft designs can be expensive since providing resources for all possible scenarios can incur significant additional vehicle mass. We investigate missions where communications resources are limited, requiring autonomous planning and execution. Unlike typical networks, spacecraft networks are also suited to automated planning and scheduling because many communications can be planned in advance. Because the network of spacecraft can represent multiple missions, missions will be reluctant to give up control of the spacecraft. Because communication among spacecraft is often intermittent (due to orbital and resource constraints), a spacecraft that can make scheduling decisions autonomously will be more responsive to unexpected events. Thus, a centralized planning system will not be sufficient to enable reactive communications, so we propose a distributed network scheduling system.

The software automatically negotiates the rescheduling of these communications with other spacecraft while respecting constraints with communication resources (such as memory and transceiver availability). Each node (spacecraft) tracks only its own communication activities and makes its own scheduling decisions but can propose communications with others. It provides an interface for a user or automated process to request communication service and to receive a reservation with updates on the expected or resulting quality of service (QoS). The communication needed to coordinate planning ("meta-communications") are not scheduled by the system because the overhead is insignificant compared to science image transfers. However, simulations of the system limit this communication to available view periods.

Figure 1 shows the architecture of a single node in the distributed network scheduling system. The middleware component provides an application-level interface to the communications protocol stack. It passes requests from users (e.g. mission operations staff, scientists), autonomous control systems, or other spacecraft to a distributed planning interface that manages the negotiation process, and instantiates goals in the planner. The planner schedules communications to achieve the goal with help from adaptive communication algorithms. It does this by providing contextual information about the future network state and the communication goal in question. The adaptive algorithms then simulate and report how data will be transferred and with what quality of service (QoS). The distributed planning interface then returns the schedule and QoS information to the requestor as a reservation. Distributed planning also manages the negotiation of requests as the needs of the spacecraft change (as determined by the planner). Status of reservations are updated and reported as re-planning, negotiations, and execution of communication activities unfold.

In the remainder of this document, we describe the interfaces of the architectural components and their implemented capabilities. We then show that the exploration rate of a rover (similar to Mars Science Laboratory – MSL) on a long traverse can be improved by no more than a factor of three with adaptive communications. We also simulate communications between a rover and Earth with orbiter relays to

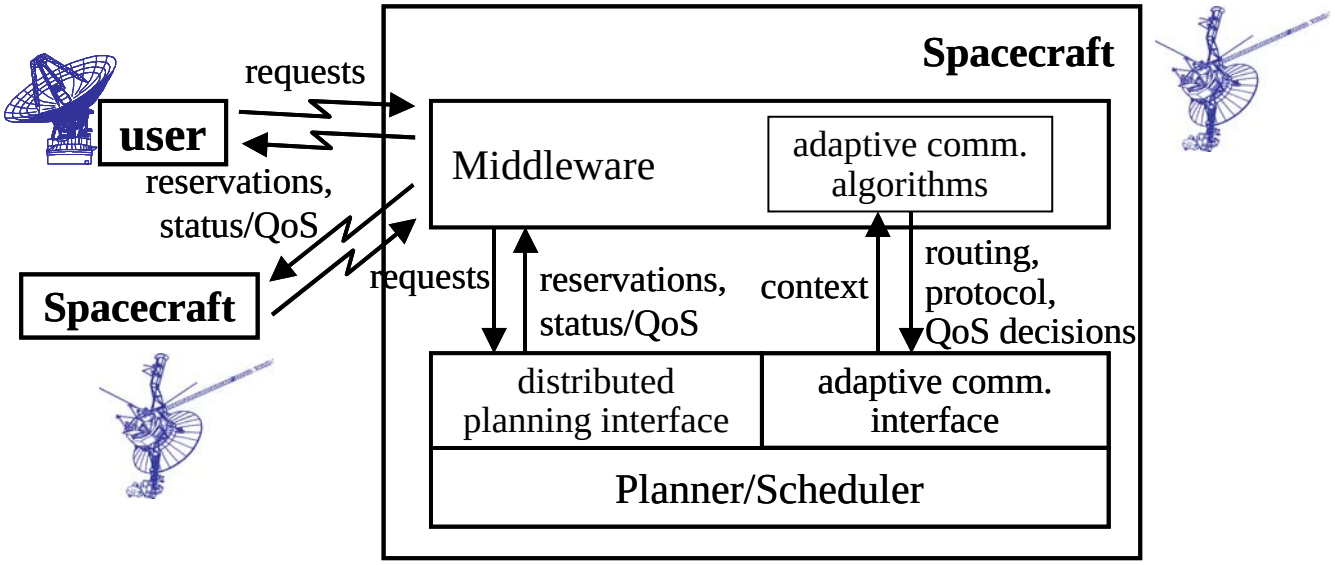


Figure 1. Distributed network scheduler architecture

demonstrate reduced latency by as much as 3.5 hours. We then report experiments on a simulated Mars network of five spacecraft/rovers to gauge the systems ability to reactively re-schedule communication activities in a distributed fashion.

## 2 Communication Requests, Reservations, and Status

An application or user requests future communication from the network by providing values for the following variables:

- `int` **`id`** – index for tracking
- `string` **`source`** – who is sending data
- `string` **`destination`** – who is receiving the data
- `int` **`size`** – estimate of size of data to be sent in Kbits
- `real` **`bandwidth_min`** – minimum required bandwidth Kbits/s
- `real` **`bandwidth_max`** – maximum usable bandwidth in Kbits/s
- `real` **`priority`** – importance of fulfilling request (larger numbers indicate greater importance)
- `int` **`start_time_min`** – minimum requested start time of communication
- `int` **`start_time_max`** – maximum requested start time of communication
- `int` **`duration_min`** – minimum needed time duration of initial data transmission
- `int` **`duration_max`** – maximum requested time duration of initial data transmission
- `int` **`delivery_time_min`** – minimum required delivery time
- `int` **`delivery_time_max`** – maximum requested delivery time
- `bool` **`progressive`** – whether data is recreated as it is received (= true) or transmission is only valuable when completed, i.e. all or nothing (= false)
- `real` **`loss_overall`** – maximum percentage loss tolerance of overall data
- `real` **`loss_per_block`** – maximum percentage loss tolerance for any block
- `real` **`loss_block_size`** – size of block for which the loss tolerance is specified
- `string` **`protocol`** – what protocol(s) should be used for transmission and with what options (e.g. CFDP - noack); this string has no generic structure and is to be generated and interpreted by adaptive communications software through an interface.

Upon receiving a request, the network will schedule ("reserve") the communication and reply with the expected quality of service for the same variables above and a real-valued `percent_delivered` variable, indicating the percentage of the data delivered or expected to be delivered. Status during and upon completion of execution is also reported through the same construct.

## 3 Local Scheduling

We use the ASPEN planning system (Chien *et al.*, 2000) to schedule communications according to constraints on memory, transceiver availability, and available windows of communication between scheduling nodes (spacecraft). The main activities scheduled are `send`, `receive`, and `relay`, for transmitting, receiving, and relaying data files. Segmentation and reassembly of files is supported for when files are too large to be sent in available

communication windows. In addition, scheduling supports cut-through switching, receiving and relaying a file simultaneously when multiple transceivers are available. The timing and duration of activities takes into account constraints on communication delay and bandwidth. While quality of service estimates/status is propagated through the network, the scheduler currently does not handle failures, such as over-tolerance data loss.

### 3.1 Scheduler Activities

The main activities we use to model data transfer are `send`, `transmit`, `receive`, and `relay`. The `send` activity recursively decomposes into a series of `transmit` activities for segmentation of the file transfer. It also includes a `free_memory` activity following each `transmit` where the amount of data sent is replenished to a memory resource at a time indicated by a `free_type` parameter, which has one of the following values: "`never`", "`on transmission`", "`on delivery`", "`on_custody_xfer`." If "`on transmission`," the memory is freed at the end of the `transmit` activity. If "`on delivery`," memory is freed at the end of the `receive` activity of the receiving node. The "`on_custody_xfer`" value is intended to support custody transfer protocols that are not yet implemented.

When one node is executing the `send` activity, the receiving node executes a `relay` activity. The `relay` activity decomposes into a `receive` activity. If the receiving node is not the intended destination of the file, then the `relay` activity also decomposes into a `send` activity, routing the data elsewhere.

The `transmit` and `receive` activities are constrained to be scheduled only during available communication windows, which are modeled as states having "`in_view`" and "`out_of_view`" values over time intervals provided by the system designer. These activities also must reserve a transceiver resource from a set provided by the system designer. The adaptive communication algorithms (shown in Figure 1) provides the assignment algorithm. The `receive` activity also consumes memory of the amount of the data received.

Delay between the start of the `send` and `receive` activities between pairs of nodes is specified through an adaptive communications function. Cut-through switching is implemented in the `relay` activity. This is where a file is received and transmitted simultaneously. The start of the `send` sub-activity (of `relay`) is computed according to the start of the `receive` sub-activity and the data rates of both `receive` and `send` such that data blocks are not sent before they received. These activities and resources are modeled in the ASPEN (Activity Scheduling and Planning Environment) modeling language (Sherwood *et al.* 2000).

The interface to adaptive communication algorithms (shown in Figure 1) is simply the provision of many of the dependency functions in the above activities. Again, these functions could be provided in a middleware communication layer.

### 3.2 Resources and States

- **memory** – Decisions about when to store and delete data are based on memory availability.
- **data** – It may be important to keep track of whether particular data files are stored or deleted in case one needs retransmission due to an unexpected failure.
- **antenna(s)** – Spacecraft can only communicate with one (or maybe two) others at a time.
- **communication windows** – Spacecraft can only communicate when in view of each other.

Obvious resources that are not considered are power and battery energy. We do not consider the network scheduler's role to handle these other resources and assume that their safe use is guaranteed by ground operations or an onboard planning and execution system.

### 3.3 Metrics

The network scheduler currently reschedules to resolve conflicts, but can be extended to optimize the schedule according to summed priority of scheduled activities over a horizon by using ASPEN's optimization framework. The network scheduler itself will be evaluated in simulation according to time to resolve new or changed requests. This will be compared to current techniques later in the Evaluation section.

### 3.4 Adaptive Communication API

The following functions (listed as dependencies in the model of the activities) should be implemented to decide how to adapt communication for a given context:

`string choose_antenna(my_name, protocol, requested_bandwidth)`
: determine which antenna should be used according to the protocol and requested bandwidth

`string route_to(sender, destination, protocol)`
: determine to whom data should be routed next

`real request_bandwidth(requested_bandwidth, sender, antenna, receiver, bandwidth)`
: determine the appropriate bandwidth based on the protocol and the requested bandwidth

`bool is_interruptible(protocol)`
: determine whether are not the transmission can be interrupted and continued later

`bool is_progressive(protocol)`
: determine whether or not this protocol allows the file to be created as it is received (otherwise it is sent all-or-nothing)

`string get_free_type(protocol)`
: determine when the data can be cleared from memory (i.e. when custody is transferred); valid values are "`never`", "`on_transmission`", "`on_delivery`", and "`on_custody_xfer`"

`real reply_percent_delivered(protocol, duration, bandwidth, size,percent_delivered)`
determine the percentage of the data expected to be delivered

`real reply_loss_overall(protocol, loss_tolerance_overall)`
determine the percentage loss of the overall image according to the protocol and the requested tolerance

`real reply_loss_per_block(protocol, loss_tolerance_per_block, loss_block_size)`
determine the maximum percentage loss of the image per block according to the protocol and the requested tolerance

`real calc_send_bandwidth(my_name, send_start_time, destination, receiver, antenna, requested_delivery_time, delivery_time_max, send_protocol)`
determine the expected bandwidth to be used by the sender's protocol base on the request

`string get_send_protocol(my_name, send_start_time, destination, receiver, antenna, requested_delivery_time, delivery_time_max, send_bandwidth)`
determine the protocol the sender should use for this request

## 4 Distributed Scheduling

Scheduling is distributed by propagating information through the network to nodes that are affected and by giving each node some level of decision-making authority with respect to local scheduling. We use Shared Activity Coordination (SHAC) (Clement and Barrett, 2003) to implement this.

SHAC is an interface between planning/scheduling systems, a general algorithm for coordinating distributed planning, and a framework for designing and implementing more specific distributed planning algorithms. Within SHAC, a shared activity is an activity that some set of planners must collectively schedule. It can be a coordinated measurement, a team plan in which planners have different roles, a use of shared resources, or simply an information sharing mechanism. Planners are coordinated when they reach consensus on the shared activity. Consensus is achieved when they agree on values for members of the shared activity structure:

- **Parameters**: Shared variables (e.g. start time, duration, bandwidth)
- **Constraints**: Each planner's constraints on parameter values
- **Roles**: Subset of planning agents assigned to roles
- **Permissions**: Variables that determine how each planner is allowed to add, remove, and modify a shared activity

Roles determine how an agent participates in the shared activity. For example, a transmit role in a shared communication activity has different resource constraints than the receive role. Roles specify which agents share the activity and can determine permissions and the protocol used to govern the agent's handling of the shared activity. Constraints can specify restrictions the agents have on the values of parameters. By propagating local constraints, agents can make scheduling choices that avoid conflicts with others without knowing the details of their plans. For example, an agent can send a constraint on the time windows of an activity as local scheduling constraints.

Protocols (distributed planning algorithms) specify how constraints, roles, and permissions of the shared activities change over time and are used to resolve conflicts among the planners. For example, a round-robin protocol rotates permission assignments among the planners, giving them each turns to replan the activity. A delegation protocol assigns and re-assigns agents to roles. Protocols are designed by sub-classing built-in and user-defined protocol classes.

By constructing protocols and modeling the attributes of shared activities, a system designer specifies the autonomy of each agent with respect to decision-making and computation. A completely centralized approach gives one agent a role in each activity with full permissions. Decentralization is introduced when agents propagate constraints, when agents fulfill different roles, or when more than one agent has planning/scheduling permissions. The SHAC coordination algorithm (stated simply) is a continual loop of refining/replanning, applying protocols to further modify shared activities, sending shared activity updates to the sharing planners, and integrating updates from others. The planner interface enables different existing planning tools to interact in this framework. More information about the algorithm and protocols can be found in (Clement and Barrett, 2003, Clement *et al.*, 2004).

SHAC is customized for the particular application domain. For a Mars network, we specified shared activities between pairs of spacecraft mapping transmit activities of one spacecraft to relay activities in another. Shared parameters include those of the request/reservation. The roles specify which local activity (`transmit` or `relay`) corresponds to each agent (spacecraft) potentially participating. The transmitter is assigned a delegation protocol for choosing a spacecraft to relay the data. Other agents are assigned a subordination role. The subordination protocol will remove the agent from the shared activity's roles with a specified probability if the agent is yet unable to successfully schedule the activity locally. This triggers the delegator to assign another subordinate.

## 5 Evaluation

The distributed network scheduler enables reactive communications within the context of scheduled operations by autonomously negotiating over communication changes. In addition, the scheduling system can generate schedules using the same negotiation mechanisms. This means that separate missions (such as the many studying Mars) can use this software to collaboratively schedule communications on the ground.

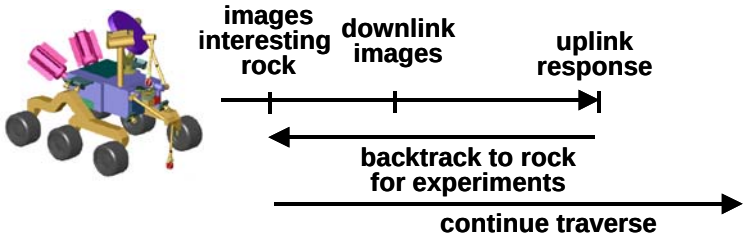


Figure 2. Rover backtracking to study a rock during a traverse

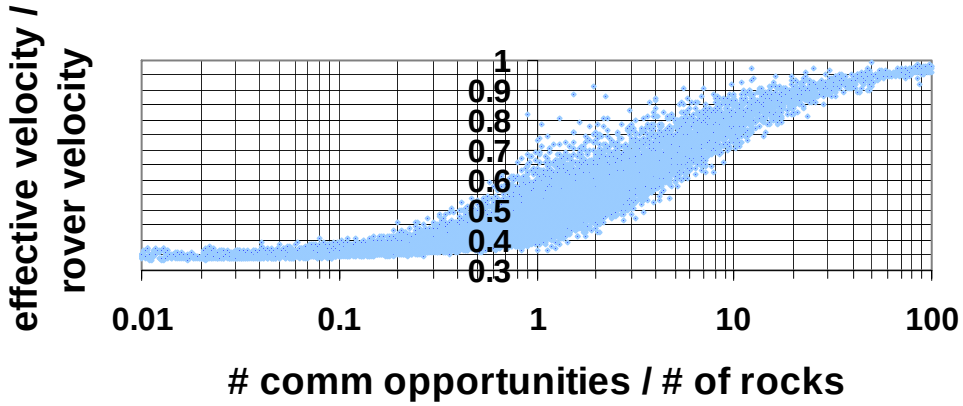


Figure 3. The effect of increasing communication opportunities on rover exploration speed

A prototype network scheduling system was implemented for communication models of MER-A, MER-B, MGS (Mars Global Surveyor), Odyssey, and Mars Express. We give experimental results for this application domain after giving more theoretical results illustrating the benefits of adaptive communication for rover exploration that this system enables.

### 5.1 Rover Exploration Performance

Here we examine the science return performance of a semi-autonomous rover investigating rocks during a long traverse between sites. We simulate a traverse based on early MSL scenarios where a rover has the ability to autonomously detect rocks of potential scientific interest, downlink images, and investigate based on commands returned after scientists have studied the images. The rover continues along its path and turns back to perform detailed measurements if commanded. Figure 2 illustrates how the rover must traverse the path three times in order to return to the rock. In the worst case, a target rock is identified just after each communication opportunity causing the rover to traverse the entire distance three times. By providing more communication opportunities through adaptive rescheduling this backtracking can be reduced, resulting in a theoretical opportunity of threefold exploration speedup.

We simulated traverses with communication opportunities at fixed time intervals and placed rocks along a straight-line path according to a Poisson distribution. The intervals between communication opportunities and number of total rocks were varied for each run. Figure 3 shows a plot of how the rover's exploration speed approaches the optimal as the number of communications opportunities increases with respect to the number of rocks. Adaptive communications allows the rover to take advantage of opportunities that were previously unscheduled. This does not mean that additional bandwidth to Earth is required since only the needed opportunities are taken. By giving more opportunities, the overall performance is increased. For example, if the communications opportunities are doubled (moving the x-axis ratio from 1.0 to 2.0 in the figure), there is a potential increase from 0.5 to 0.6 (y-axis) in the normalized exploration speed of the rover, resulting in a 20% increase in science return.

### 5.2 Rover Communication Performance

In some situations a rover must sit idle while waiting on a response from Earth. For example, using the Rock Abrasion Tool (RAT) can take as long as nine days because of uncertainties of executing a long sequence of grinding, drilling, sampling, and measuring and the delays of communication in getting new sequences from ground.

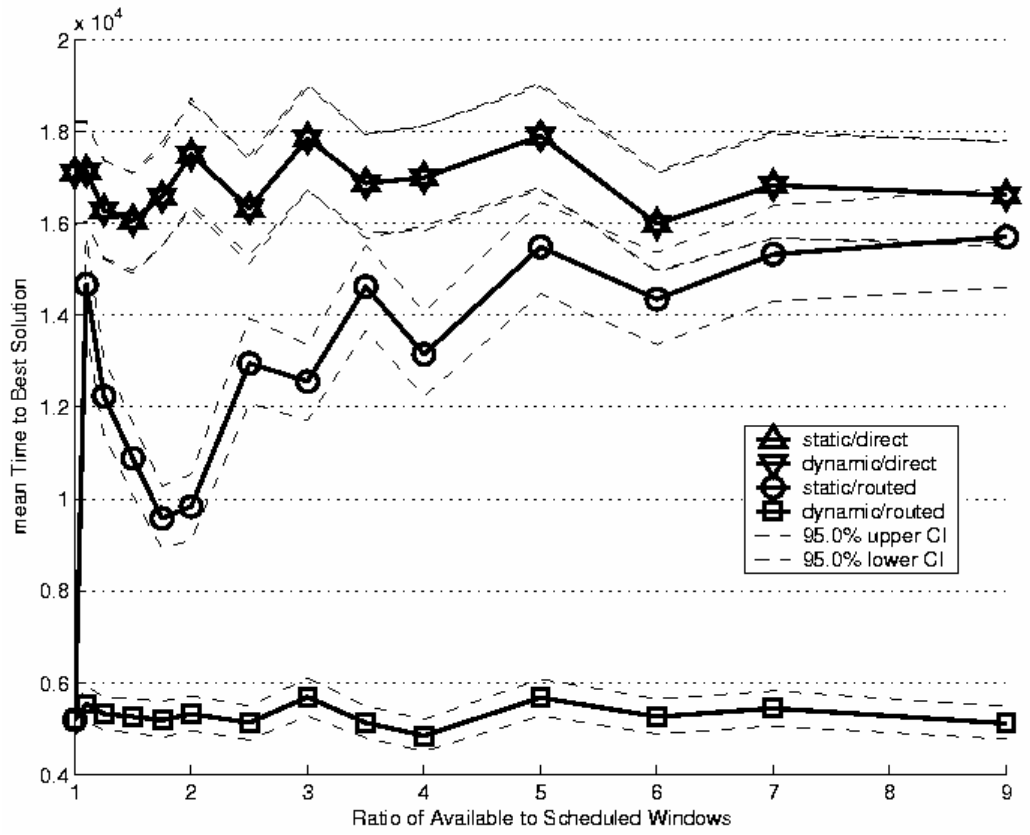


Figure 5. Improvement of adaptive rescheduling in delivery time from rover to Earth

Thus, reducing the delay between the need for communication and getting a response is of great interest to Mars missions.

Using information about the duration and frequency of orbiter passes as well as cross-link intervals between orbiters and Earth, we simulated the transfer of data from a rover to Earth with potential routing through orbiters or a direct-to-Earth (DTE) link. Passes of MGS and Odyssey with rovers is 10 minutes in duration, and Mars Express is 8 minutes with rovers. Communication delays are 20 minutes with Earth and 1 second otherwise. Figure 4 shows mean delivery times (in seconds) to Earth for 1000 simulations per point of four routing cases based on varying numbers of scheduled communications and possible windows of opportunity. "Static" means that only scheduled links can be used, and "dynamic" means that any window can be adaptively scheduled on the fly. "Direct" means that the rover can only use DTE communications, and "routed" means that it can route through orbiters. Dashed lines show the 95% confidence interval, indicating variance. Outliers skew the mean and cause the jaggedness of the plot. Here we assume that all DTE opportunities are scheduled, so the static/direct and dynamic/direct cases are the same.

The difference in the static/routed and dynamic/routed is the performance improvement for rescheduling communications on the fly. For a ratio of one (where all windows are scheduled), performance is the same (as expected), a little over 5000 seconds (about an hour and a half). For fewer scheduled windows, the performance quickly degrades and approaches the DTE plot (at the top) of ranging just below five hours. The difference in the static/routed and the DTE plot shows the performance improvement of routing through orbiters as the number of scheduled routing opportunities decreases.

### 5.3 Mars Network Experiments

A rover can take advantage of sending data to an over-passing orbiter with minimal negotiation. The orbiter can weigh the priority of the rover's data with the orbiter's current resource needs and decide whether it will agree to relay the data. We have assumed that the orbiters always agree to route the rover's data in the above experiments. However, once they have agreed to the new communications goal, the network of spacecraft may need to reschedule other communications. But, how long does it take for the network to reach consensus on a new schedule based on the addition of a goal?

We performed experiments for the Mars network given the same communication scheme for the previous simulation to see how long the distributed network scheduler could reach consensus upon the addition of a new communications goal. The passing of time was

simulated, and coordination messages were routed through SHAC according to the availability communication windows. We modeled SHAC similar to that at the end of the Distributed Scheduling section. To simulate the slowdown of flight processors, we need time to pass around 500-1000 times faster than real time. Since all five spacecraft were run on the same workstation in these experiments, we simulated time passing 100 times faster than system time for a slowdown factor of 500, roughly. Measurements were made in both cpu seconds and actual seconds while simulating time. Replanning cycles (1 for each SHAC cycle) were restricted to 20 ASPEN iterations, and when there were no remaining conflicts, SHAC slept for 0.1 seconds (10 simulated seconds) between cycles. Results for 5 problems are reported for each number of goals ranging between 3 and 20. These numbers are of communication goals for each spacecraft over a 3-day horizon. Goals were randomly generated with data size ranging between 2 and 100 Kbits, with delivery deadlines ranging between 40,000 and 100,000 seconds (~11 to 28 hours), and with random destinations to either Earth or a rover (but rovers always send to Earth). Once a schedule was generated, another goal was added for a random spacecraft at a random time in order to measure the effort in reaching consensus for an unexpected event.

Figures 6 and 7 show the results of rescheduling for one goal added to the system. Each point in the plots is a mean, min, or max value for the five spacecraft. Figure 6 shows that cpu time is limited to less than 200 seconds for a spacecraft. On a flight processor, this translates to 100,000 to 200,000 seconds, or 28 to 56 hours. The mean maximum time is between 4 to 30 seconds, or 1 to 14 hours. The system time in Figure 7 shows that on average, the system reaches consensus in less than an hour (according to the max time trend line), but it can take a little more than 200 actual seconds in some cases. We must multiply this by the simulation speedup factor of 100; thus, the approximate maximum time onboard for this problem set would be roughly 56 hours. This means that without careful coordination, activities scheduled within 56 hours of the unexpected event could be executed inconsistently with respect to other spacecraft. To resolve this problem, the spacecraft should switch to a simpler protocol with real-time consensus guarantees for activities soon to be executed. Although not employed in this application, an algorithm for determining when this switch should occur (based on the time of the activity, communication windows, and the routing required by the simpler protocol) is described in (Clement and Barrett, 2003). Communication overhead for reaching consensus ranged to at most 25 messages and 18 kilobytes. Many of the goals could be resolved locally, requiring no communication. Although not shown here, initial schedule generation (applicable to collaborative ground planning) averaged 25 cpu seconds, 102 actual seconds, 91 messages, and 55 kilobytes per spacecraft. The actual time (to reach consensus) ranges from seconds to 50 minutes. Plots of these results can be found in a lengthier version of this paper (Clement and Schaffer, 2004).

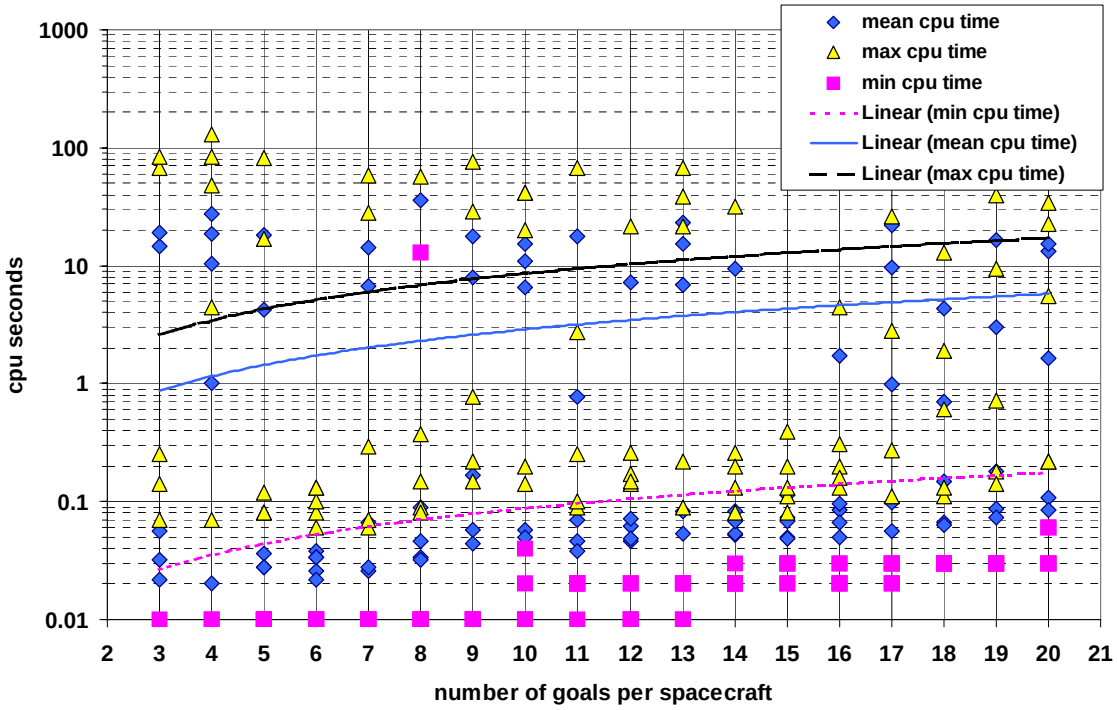


Figure 6. CPU time for scheduling a single goal

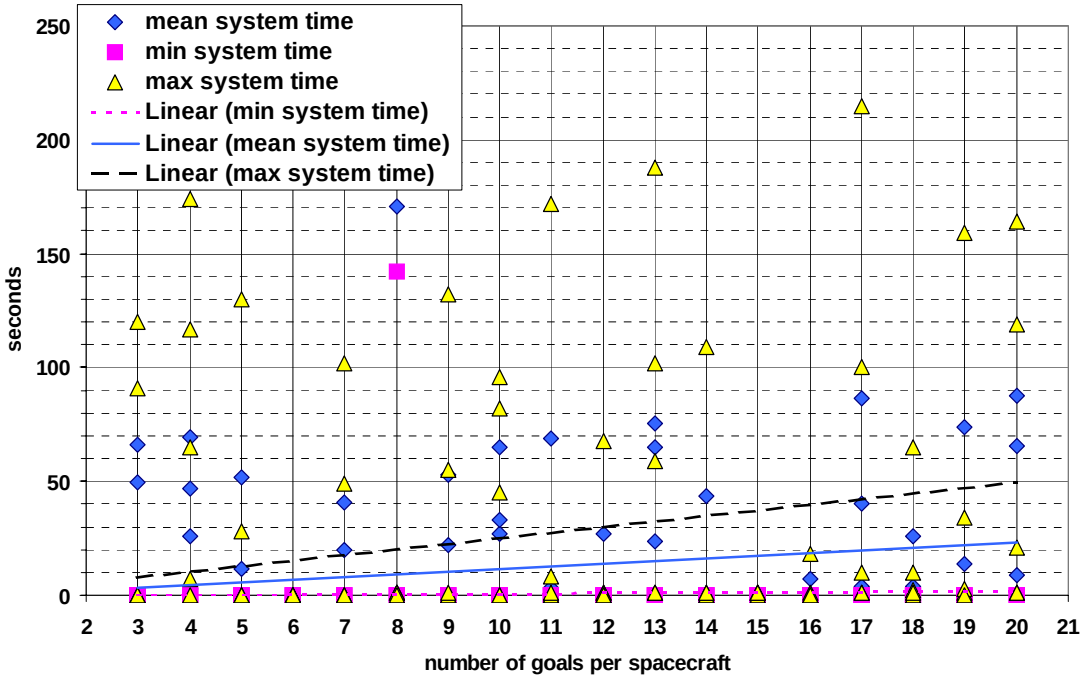


Figure 7. Actual time to schedule a single goal

## 6 Conclusion

This report described software for onboard distributed scheduling of communications among a network of spacecraft in a fashion. The distributed network scheduler enables reactive communications within the context of scheduled operations by autonomously negotiating over communication changes. In addition, the scheduling system can generate schedules using the same negotiation mechanisms to enable missions (such as the many for Mars) to collaboratively schedule communications on the ground. Evaluations were based on models of MER-A, MER-B, MGS, Odyssey, and Mars Express. Our main results include:

- simulation showing a maximum opportunity of improving traverse exploration rate by a factor of three;
- simulation showing reduction in one-way delivery times from a rover to Earth from as much as 5 to 1.5 hours;
- simulated onboard response to unexpected events averages under an hour; and
- ground schedule generation ranging from seconds to 50 minutes for 15 to 100 goals.

Future work includes extending the software to schedule during execution using varied protocols to give real-time consensus guarantees and integrating the software with a realistic communications simulator.

## References

I. Akyildiz, O. Akan , C Chen, J. Fang, W. Su. *InterPlaNetary* Internet: state-of-the-art and research challenges. *Computer Networks*, 43, 75–112, 2003.

S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, D. Tran , ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling, *Proc. SpaceOps 2000*, Toulouse, France, June 2000.

B. Clement, A. Barrett. Continual Coordination through Shared Activities. *2nd International Conference on Autonomous and Multi-Agent Systems (AAMAS 2003)*. Melbourne, Australia. July 2003.

B. Clement, A. Barrett, S. Schaffer. Argumentation for Coordinating Shared Activities. *Proc. IWPSS*, June 2004.

B. Clement, S. Schaffer. Distributed Network Scheduling. *The Interplanetary Progress Report*, 42(156), February 2004.

R. Sherwood, B. Engelhardt, G. Rabideau, S. Chien, R. Knight. *ASPEN User's Guide*. JPL Technical Document D-15482, http://www-aig.jpl.nasa.gov/public/planning/aspen/, February 2000.