A Data-Rate Aware Telemetry Scheduler

CMU-RI-TR-01-12

Punitha Manavalan
punitha@andrew.cmu.edu
Information Networking Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Phone: (412) 268-6334

Michael D. Wagner
mwagner@cmu.edu
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Phone: (412) 268-5354

May 2001

Abstract

Tele-operation of robots involves transmission of numerous telemetry data across often low-quality communication links. The available data rate of these links may vary over the course of a mission; however a human tele-operator must always receive at least the most critical data to safeguard the robot. This paper presents the *Telemetry Manager*, a data rate-aware telemetry scheduler that attempts to guarantee delivery of high-priority data from robot to operator even in the face of a low-performance communication link. In our scheme, data are prioritized by their "time to live" (TTL). Data with small TTLs are sent before those with large TTLs. We compare how capably the Telemetry Manager can achieve this performance to a control scenario where communication over a network link is not scheduled. The initial results show marked improvement in the number of high-priority messages received within their TTL as well as in mean transmission times of high-priority messages.

1. Introduction

During operation of a mobile robot, various data are generated such as robot pose, location information, power levels and scientific data. These data must somehow be transmitted off the robot for tele-operation, scientific experimentation or evaluating performance. However, some of the most interesting robotic exploration scenarios provide only a minimal communications link between robotic device and human operator. Terrain occlusions, competing networks and weather can all make the available data rate to the robot unpredictable and noisy. When the link is strong, high data rates can be achieved and all available telemetry and command data should be transmitted to and from the robot, enabling precise tele-operation by a human operator in the face of hazards, and archiving all robot telemetry for post-processing and data analysis. However, in times of low data rate, only the highest priority telemetry and commands should be transmitted, allowing the robot to still be rescued if necessary. No design currently exists for sensing what data rate is available to make intelligent decisions about what data to send and receive. This paper presents an approach to intelligently scheduling telemetry data transmission based on the currently available network data rate.

The remainder of this paper is organized as follows: sections 2 and 3 detail the problem we are attempting to solve and the state of the art in telemetry scheduling. Section 4 summarizes our approach towards solving the problem at hand, and section 5 explains the software architecture of the implementation we call the *Telemetry Manager*. Section 6 details how the performance of the Telemetry Manager was evaluated using a general-purpose testbed along with the results of this evaluation. Section 7 summarizes future directions of research and section 8 concludes the paper.

2. Problem Definition

Communications links between robots exploring other planets or remote parts of the Earth require manual scheduling of data transmission to a remote operator. In some cases, proprietary networking protocols are used that are tuned to low bandwidth and noisy links. In other situations, common protocols such as TCP or UDP are used. These offer advantages such as ease of development and compatibility with existing network applications and messaging APIs [Simmons01, RTI01]. However, these protocols do not work as well over unpredictable links due to transmission control inappropriate for low bandwidth links (such as TCP's slow start mechanism). Therefore developers are forced to choose between very conservative estimates of available data rates to and from the robot or to risk link congestion that can cause firm real-time tele-operation deadlines to be missed.

One factor involved in manual scheduling of telemetry is prioritization of messages. Scheduling faster production of high-priority data and slower production of lower-priority data is a typical strategy. For instance, a message stating that the robot has detected a hazard such as a boulder or cliff is sent at a higher frequency than data about local weather conditions. While this approach is simple it suffers from two drawbacks. First, it does not allow for asynchronous production of data; each message must be scheduled. This places unnecessary constraints on the value of each type of data. Second, it often suffers from congestion when available network data rates degrade unexpectedly. With protocols such as TCP, this can cause the link to effectively provide a step function of performance: either the link is working normally or it is congested and nearly no data are being received. If this congestion

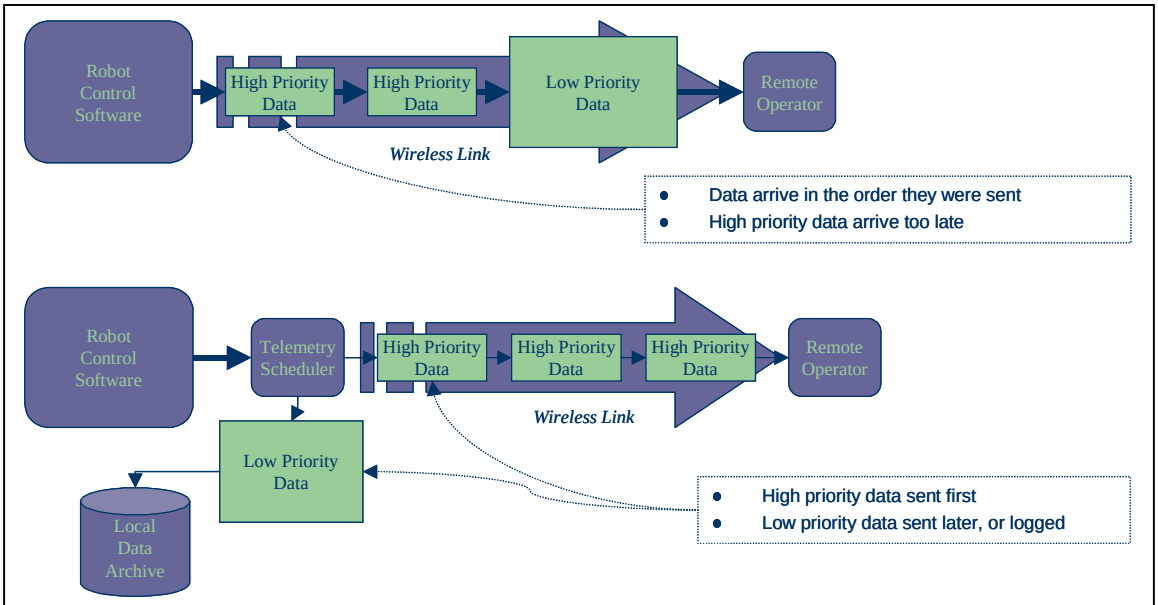

Figure 1. Telemetry scheduling allows high-priority data to arrive to a remote operator first, avoiding priority inversion even if low priority data is generated first.

comes about while sending a large, low priority message, the system is suffering from a type of priority inversion, where high-priority data cannot be received because the link is busy sending low-priority data (see Figure 1).

3. Related Work

Space missions such as Mars Pathfinder [Mishkin98] have very low data rate (6 kbps) connections to ground control and but generally manually plan all data uplink and downlink communication. Direct tele-operation is infeasible on these missions due to round trip delays caused by the speed of light, so communications that took slightly longer than expected were considered expensive rather than dangerous. During time-critical space operations such as planetary fly-bys, communications were carefully and manually scheduled.

Some experimental terrestrial robots such as Nomad [Apostolopoulos00, Bapna98a] have focused on maximizing available data rate to and from a robot by developing technologies such as antenna pointing systems [Bapna98b]. Furthermore, while advanced navigational and science autonomy reduce the need for high speed links to a remote robot [Singh00, Roush99] little research has focused on how to optimize whatever link data rate is available.

4. Approach

Our approach is to avoid data priority inversion by automatically scheduling data for transmission over the network. Data are no longer transmitted in the order in which they were generated by the robot. Instead they are placed in a priority queue so that high-priority data are transmitted before low-priority data. The scheduling approach utilized has been targeted towards the design of a specific application, the Hyperion robot.

4.1 Target Application

The Hyperion solar powered field robot is being developed by the Sun-Synchronous Robotics Project at the Robotics Institute of CMU [Wettergreen01]. In July 2001, this four-wheeled autonomous field robot will demonstrate long-distance (24 km) sun-synchronous circuits in the Canadian Arctic (see Figure 2). On- and off-board intelligence will attempt to optimize solar energy gain while avoiding damaging obstacles. The prototype robot will operate continuously for 24 hours, demanding highly reliable mechanical, electrical and software systems.

Communication with Hyperion will be performed by a radio Ethernet network using the 802.11 protocol. Messages are passed between processes on and off the robot using the IPC messaging API described in [Simmons01]. IPC message types are represented by standard C data

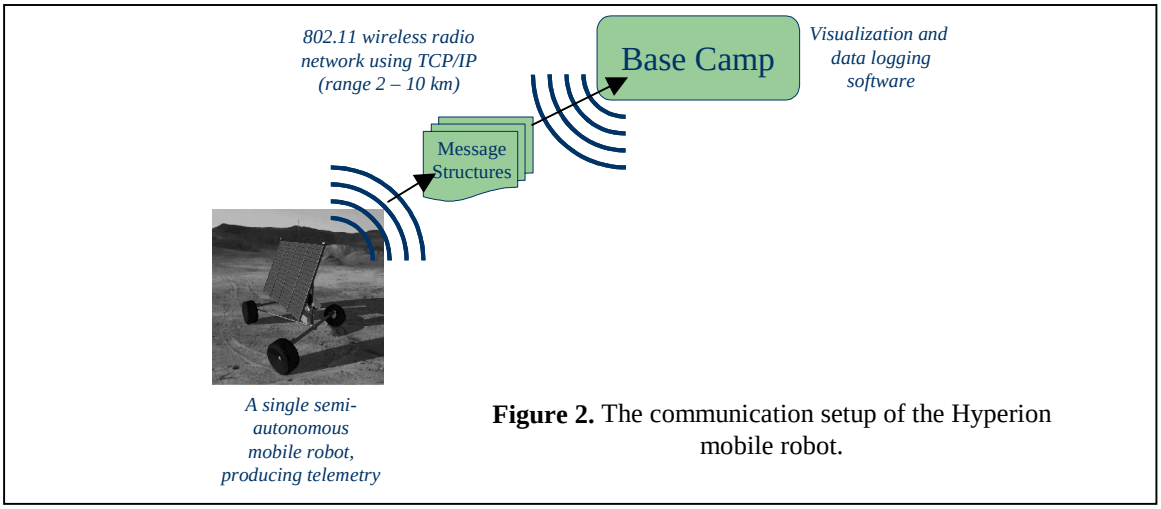

Figure 2. The communication setup of the Hyperion mobile robot.

structures. Messages are transmitted in an anonymous publish / subscribe architecture.

Hyperion's uses IPC for all its command and telemetry data. Robot pose, GPS location, battery and solar power levels, navigation sensor data and panoramic imagery are all various types of messages that Hyperion will generate. These messages will be transmitted to a base camp several kilometers away, where a graphical operator's interface will use them to visualize the robot's status.

Additionally, all messages will be logged to enable performance analyses of the robot. While logging all messages on Hyperion would seem to be the most efficient option, on-board data storage resources are limited to a 1 GB hard drive so all telemetry cannot be logged onboard – some messages must be logged at the base camp. Furthermore, a data log on Hyperion is inherently susceptible to damage due to vibration or harsh thermal conditions, so a data log in the base camp is preferable even if the robot did not have disk space limitations.

4.2 Message Priorities

We initially attempted to assign a priority to each IPC message generated by Hyperion. Optimizing these priorities is difficult because their relative importance is based on too many factors that have yet to be quantified. Therefore our prioritization approach involved assigning a "time to live" (TTL) for each message. If the remote operator receives a message within that message's TTL, its contents are still useful for tele-operation or performance monitoring. After a message's TTL, its contents are no longer useful. A message's TTL can be determined by analyzing sensitivities of tele-operational control loops, such as robot stopping distances and processing times.

4.3 Design Assumptions

Several design assumptions arise from our target application:

1. A reliable transmission protocol is utilized (in the case of Hyperion, TCP/IP is used). The scheduler itself does not attempt to guarantee delivery of messages.
2. Delivery of messages within their TTL is a firm real-time task. Hard real-time deadlines are not imposed on the delivery of messages.
3. The network between robot and remote operator contains only two nodes: the robot itself and a computer at the base camp.
4. Telemetry data generated by the robot is the primary source of network packets. Command sequences to the robot are considered to be an insignificant source of packets.

These assumptions allow us to characterize the network only by its link quality and not by its degree of congestion. In fact, all congestion can be avoided by proper scheduling of message transmission.

5. System Architecture

The telemetry manager exhibits an architecture shown in Figure 3.

While targeted towards operation on Hyperion, the architecture displays enough generality to apply to other applications with minimal modifications. Each module has a well-defined role and interface, which results in a simple

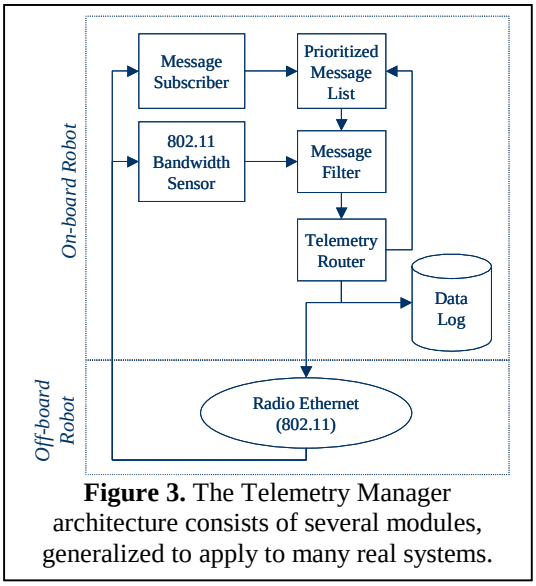

Figure 3. The Telemetry Manager architecture consists of several modules, generalized to apply to many real systems.

system that is easier to debug and analyze. Each module is defined below.

5.1.1 Message Subscriber

This module is responsible for receiving all telemetry messages generated by the robot. As such, it is the module that is most dependent on the messaging API used. In the case of IPC, the message subscriber notifies the IPC central process of its interest in receiving all message types it has defined. When any other process on the robot generates a robot, IPC central passes the data to the message subscriber, which subsequently inserts the new message into the prioritized message list.

5.1.2 Data Rate Sensor

This subsystem is responsible for reporting the currently available data rate (bits per second) between the robot and the operator's base station. Most data rate sensing involves testing a network connection by measuring the time to transmit a number of bits. This intrusive method adds unwanted overhead to a connection, thereby decreasing data rate for telemetry messages themselves. We took advantage of our specific application and unobtrusively measured data rate by querying the strength of the radio link (measured in dB) between base camp and the robot. Link strength is proportional to effective radio network data rate [Raylink01], [Tourrilhes01].

Using iwspy on a linux system one can test the Mobile IP support. It allows the user to set a list of network addresses in the driver. The driver will gather quality information for each address. The tool allows the user to display the information associated with each address in the list. Iwspy accept IP address as well as MAC address. This was useful for us. One the command line, hardware address should be prefixed by the keyword hw.

iwspy eth2 15.144.104.4. hw 08:00:0E:21:3A:1F

frc>iwspy eth2 08:00:0E:21:D7:4E : Quality 15; Signal 29 ; Noise 0

The code was implemented using the wireless extensions available on Linux.

The ioctl data, from the wireless interface is obtained from a range of parameters that have been defined in wireless.h, as struct iw_range. We then test the availability using /proc/net/wireless, and can get the current data rate using the macro SIOCGIWRATE.

5.1.3 Prioritized Message List

Messages are inserted into the prioritized message list by the message subscriber and removed by the telemetry router. The purpose of the list is to maintain messages until they can be transmitted or until their TTLs expire and they are logged. In times of low data rate, the message list grows large since the telemetry router cannot empty it across the network. The list is emptied later when the network link quality improves. This allows the system to perform well if the available data rate is unpredictable. However, the list size must be monitored to avoid using excessive amounts of memory if data rate remains low for long periods of time.

5.1.4 Message Filter

This subsystem uses the output of the data rate sensor and examines the messages stored to decide which messages should be shipped over the wireless link.

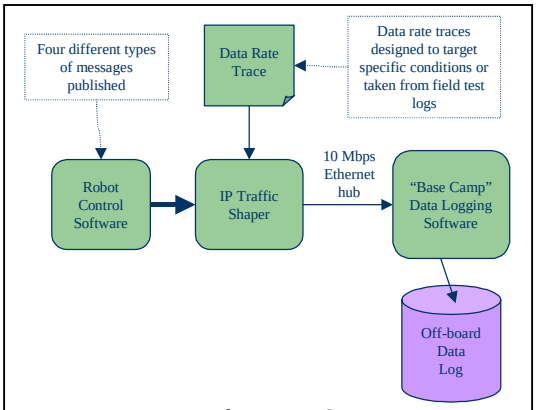

Figure 4. Setup for control scenario tests, without Telemetry Manager.

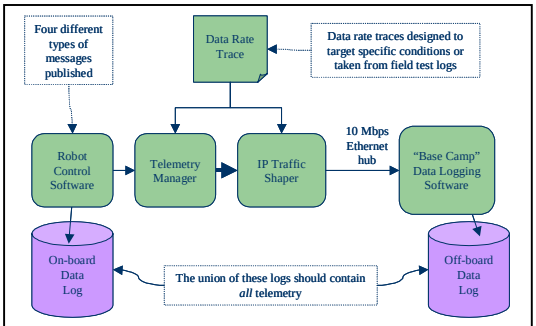

Figure 5. Setup for experimental scenario tests, with Telemetry Manager.

The filter is invoked at some specified frequency. Starting at messages with short TTLs, it traverses the list and marks a message as "to be transmitted" if its size is small enough to fit within the cumulative available data rate. If a message is marked "to be transmitted", its size is subtracted from the cumulative available data rate. A message is marked as "do not transmit" if it requires more than the cumulative available data rate.

5.1.5 Telemetry Router

The telemetry router carries out the decisions of the message filter. It traverses the message list. If a message is marked "to be transmitted", the telemetry router transmits the message to the base camp. As such, it is dependent on the messaging API used. Since the current implementation uses IPC, the telemetry router must overcome limitations imposed by this anonymous publish / subscribe system. It must temporarily disconnect from the IPC central process running on the robot and connect to an IPC central process running at the base camp. This is called a "context switch", and is required because of the anonymous publish / subscribe model used in IPC. Once a message is transmitted it is removed from the prioritized message list.

Messages that are not transmitted are logged if their TTL has expired. At this point the current time is recorded and the message is removed from the list

6. Performance Evaluation

We sought to quantify the benefits of a Telemetry Manager using two metrics:

1. The number of messages received by the base camp within TTL.
2. The mean time elapsed between message generation on-board the robot and message reception at the base camp.

We evaluated these two metrics in control and experimental scenarios. A control scenario involved transmitting all messages from a simulated robot host to a simulated remote operator station host without scheduling from the Telemetry Manager (see Figure 4). An experimental scenario involved the Telemetry Manager controlling message flow between the robot and operator (see Figure 5).

6.1 Experimental Setup

6.1.1 Message Types

These two metrics were calculated with a setup consisting of four messages, shown in Table 1. Each message as comprised of a fixed-length array of bytes and a UNIX time stamp, returned by the `gettimeofday` system call.

Message Name	Size [bits]	TTL [sec]
A	328	1
B	528	2
C	2128	5
D	8128	20
Total size:	11112	

Table 1. Messages used in experimental setup.

While a real robotic system would consist of many more messages, this relatively simple set allows performance to be analyzed more accurately.

We believe the sizes and TTLs of these messages reflect some common characteristics of robot telemetry. For instance, high-priority messages are generally designed to be small, since they

must not take much time to transfer. Larger messages, such as science instrument data, generally do not need to be sent to the operator immediately. There are, of course, exceptions to these rules such as imagery, which are important data for tele-operation. We intend to investigate these scenarios in the near future.

6.1.2 Simulated Robot Host

A "robot simulator" process using IPC simulates the various processes running on Hyperion that generate telemetry. Each message type was published at 5 Hz. This also reflects many realistic situations where the robot can generate much more data than it can transmit to a remote operator.

The simulated robot software ran on a 400 MHz Pentium II computer running Red Hat Linux 7.0 (kernel version 2.2.16-22). Version 3.2.4 of IPC central was running on this machine.

6.1.3 Simulated Operator Station Host

An IPC process that logged all messages received ran on a computer simulating an operator's station. This computer was connected to the simulated robot via a 10 Mb/s Ethernet hub. This 700 MHz Pentium III computer also ran Red Hat Linux 7.0 (kernel version 2.2.16-22) and IPC central version 3.2.4.

6.1.4 Simulated Network Conditions

The Hyperion wireless network hardware was not fully available in time to be evaluated by this study. Therefore instead we developed a customizable testbed consisting of the hosts listed above. We used the IP traffic shaper module included in the Linux 2.2.16 kernel distribution. This loadable module (`shaper.o`) is connected to a network adapter and controls the flow of packets sent out through that adapter. The shaper module, enforcing a specified one-way link performance between two hosts, drops additional packets transmitted by an application beyond a user-specified maximum data rate.

The total data rate required to transmit all message data at 5 Hz is 55,560 bps. We sought to test the Telemetry Manager's performance at both low and high data rates, so we varied the shaped data rate between 10,000 and 70,000 bps. Because 802.11-compatible hardware was not being used, the data rate sensor in the Telemetry Manager did not actually measure available data rate across the network. Instead, it returned values read from a data rate trace file written by hand before the test.

There is an additional overhead imposed by control data sent between the processes and their corresponding IPC central processes, overhead due to extra bits required for TCP headers and uncertainty due to the lack of a completely controlled network environment. These effects were cumulatively estimated with initial experiments to be approximately 20%, but the dependency on message size and number of message publications has yet to be rigorously determined.

6.1.5 Testing Procedure

The traffic shaper is a driver that creates new interface devices, those devices are traffic-limited in a user-defined way, they rely on physical network devices for actual transmission and can be used as outgoing routed for network traffic. The shaper was introduced in Linux – 2.1.15 and was backported to Linux-2.0.36. This shaper is compiled as a module.

Before all tests, the shaper loadable module was configured on the simulated robot host with the following commands:

```
insmod shaper

shapecfg attach shaper0 eth0

shapecfg speed shaper0 <speed, bps>

ifconfig shaper0 robot_host netmask
255.255.255.240 up

route add –host base_camp_host dev
shaper0
```

The shaper device can only control the bandwith of outgoing traffic, as packets are transmitted via the shaper only according to the routing tables.

For each given data rate between 10,000 and 70,000 bps, process were started in the following order:

1. The `shapecfg` program was run to set the shaped data rate to the base camp.
2. *For experimental scenarios only*: an IPC central process was started on the robot host.
3. An IPC central process was started on the operator station host.
4. A message logging process was started on the operator station host.

5. *For experimental scenarios only*: a Telemetry Manager process was started on the robot host.
6. The robot telemetry simulator program was started on the robot host (Note: for control scenarios, this process connected directly to the IPC central running on the operator station host; for experimental scenarios, this process connected to the IPC central running on the robot host).

The robot telemetry simulator ran for 60 seconds, transmitting telemetry. When a message is generated, its time stamp is set using the `gettimeofday` system call. This time stamp is transmitted with the message. When the message is logged at the operator station, a second time stamp is stored along with the message, noting when the message was received. The `rdate` program uses the Time Protocol described in RFC868 [Postel83] to correct for clock skew between the two host computers.

With these measurements, the two performance metrics could be calculated for all scenarios. In the experimental scenarios, the number of messages transmitted by the Telemetry Manager could be found by counting the number of messages logged at the operator station.

6.2 Results

One set of control scenario results and three sets of experimental scenario results were generated. We generated three different sets of experimental results because of the Telemetry Manager's observed sensitivity to network congestion caused by uncharacterized overhead (see section 5.1.4).

These results are also sensitive to clock skew between the two network hosts. While we observed that `rdate` synchronized the hosts' clocks, we were not able to verify their accuracy to less than one second. Because the TTL of message A is one second, a clock skew of even 0.5 seconds could noticeably affect results.

6.2.1 Metric 1: Number of Messages Received within TTL

The number of each message type received within its TTL is shown for control scenarios in Figure 6 and for experimental scenarios assuming 0%, 25% and 50% overhead in Figures 7, 8 and 9, respectively.

In control scenarios, it is clear that no inherent message prioritization is done. In fact the lowest priority message, D, was received most often within its TTL, probably because its TTL is the longest. Similarly, the highest priority message, A, was least often received with its TTL. These results exhibit a clear case of priority inversion.

The Telemetry Manager scheduling results of experimental scenarios seems to effectively avoid priority inversion. Messages of lower priority are initially received less often when data rates are low. In tests with higher data rates these messages are more commonly sent within their TTL, as desired. In fact, by 30,000 bps nearly all A, B and C type messages are received within their TTL.

Another effect to notice is that message types B and C are generally received within their TTL more often than type A. While this has yet to be proven, we believe this effect is due to some combination of extra processing time imposed by passing through the Telemetry Manager and sub-second clock skew between the hosts. Eliminating this effect will be an important future task because many messages on Hyperion could be reasonably assigned TTLs of one second.

6.2.2 Metric 2: Mean Transmission Time

Figures 10 through 13 depict the mean transmission times of each message type for control scenarios and experimental scenarios with 0%, 25% and 50% overhead, respectively. Transmission time is calculated as the difference between the time a message is logged at the operator station and the time the message was generated by the robot simulator software. Note that in experimental scenarios, the transmission time includes time in which the message waits in the Telemetry Manager's prioritized message list. Error bars in the figures designate the standard deviation of message transmission time.

It is clear from these results that the Telemetry Manager greatly improves mean transmission time at all data rates tested. Without a Telemetry Manager, network congestion occurs when the available data rate is less than 60,000 bps (the approximate total data rate requirement of all robot telemetry), and occurs universally for all message types. This causes IPC communication to become bursty, with long periods lacking any message transmission and short periods of where many messages are transmitted. In experimental scenarios we see less network congestion,

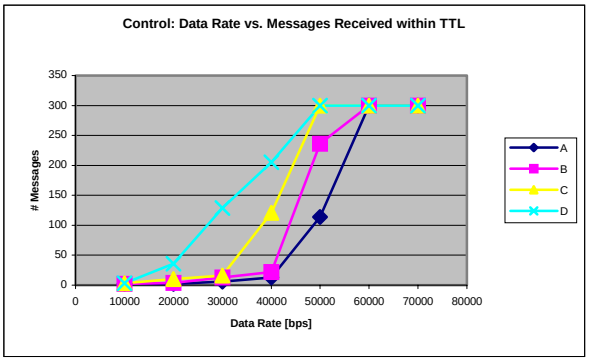

Figure 6. Messages received within TTL in control scenario

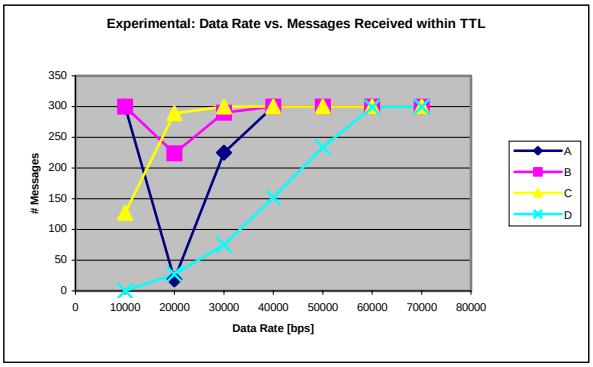

Figure 7. Messages received within TTL with 0% overhead

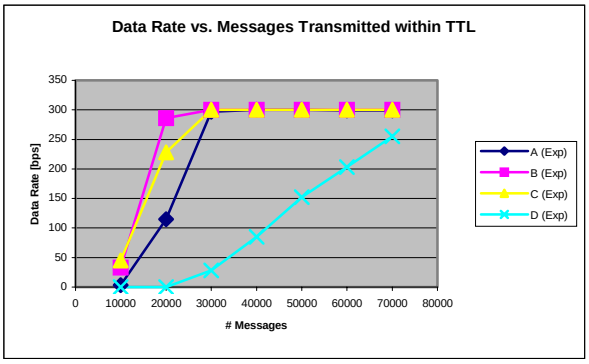

Figure 8. Messages received within TTL with 25% overhead

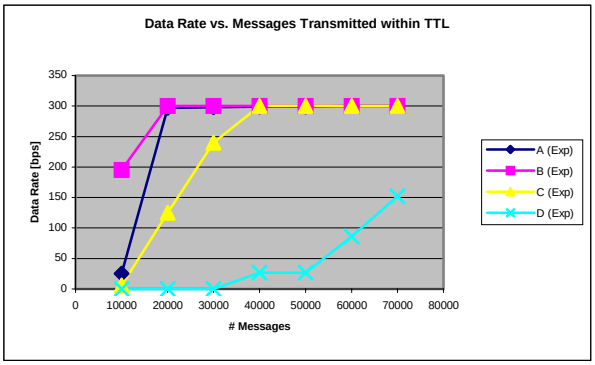

Figure 9. Messages received within TTL with 50% overhead

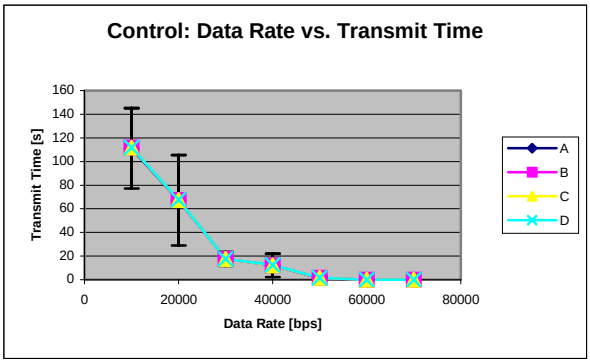

Figure 10. Mean transmission times in control scenario

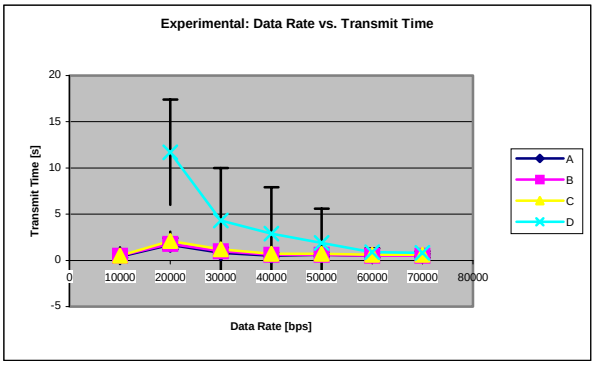

Figure 11. Mean transmission times with 0% overhead

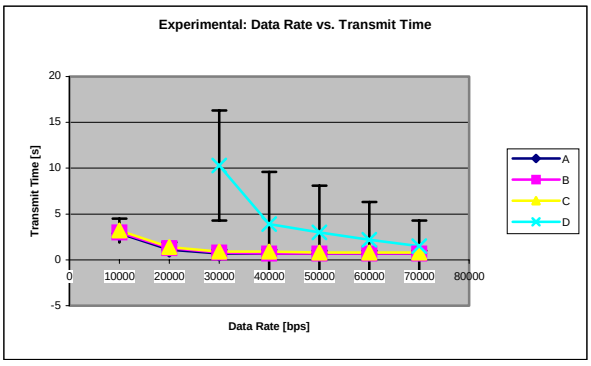

Figure 12. Mean transmission times with 25% overhead

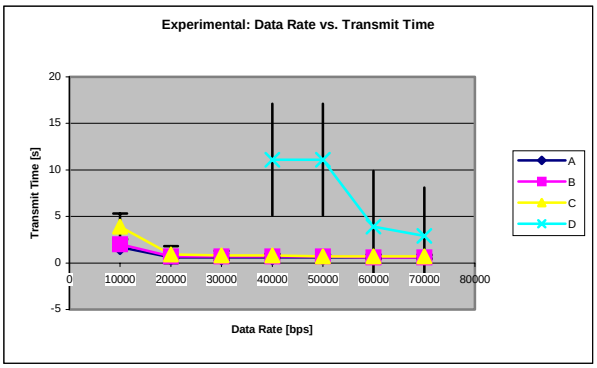

Figure 13. Mean transmission times with 50% overhead

especially when a larger overhead factor is assumed. This is verified by observing the behavior of the IPC central processes, which showed no messages backing up during the tests. With congestion reasonably under control, high-priority messages can be reliably sent with their TTL. At all data rates, the mean A, B and C message transmission times are well within their TTLs. At 10,000 bps no D messages are transmitted because all available data rate is consumed sending higher priority messages.

Another important result from this examination is a measurement of the extra processing time required by the Telemetry Manager itself. In control scenarios, a message passes from the robot simulator software to the operator station host without any additional processing. With the Telemetry Manager in place, however, a message waits for at least some small period of time in the prioritized message list. We can see that even in periods of high data rate that the mean transmission times for all message types are about 0.6 to 0.8 seconds. We assume that this latency is due to processing time in the Telemetry Manager, which could be optimized with further development time. However, clock skew between hosts could mean this measurement is either more or less than the actual Telemetry Manager processing time. The best way to accurately measure this value is a micro-benchmarking of the Telemetry Manager itself.

7. Future Work

7.1 Characterization of Effects Causing Congestion

It is clear from the results of this study that network congestion between the robot and operator station must be avoided. This should certainly be possible with two capabilities. First, the available data rate must be accurately sensed. This capability will be put to the test when the Telemetry Manager is tested using actual wireless Ethernet bridge hardware that implements the 802.11 protocol. An important factor that can be understood using this study's experimental testbed is that of overhead caused by IPC control messages, TCP header data and other as of yet unknown effects. Some of these, such as the effects of TCP, can be estimated with further review of literature. However, an in-depth study of IPC control message data rate requirements has yet to be documented.

7.2 Micro-benchmarking of Telemetry Manager

This study did not attempt to optimize the performance of the Telemetry Manager software. If messages spend an inordinate amount of time waiting in the Telemetry Manager, they are less likely to arrive to the operator within their TTL. As stated in section 6.2.2, clock skew between the two hosts in the testbed could have obscured exactly how much time messages spend in the Telemetry Manager during these tests. Therefore a micro-benchmark of the Telemetry Manager should be performed to see which optimizations, if any, would be beneficial.

7.3 Performance with Different Message Types

The four message used in this study are among the simplest types to be used on the real Hyperion robot. Some smaller messages could have long TTLs, while some rather large messages such as compressed images could have small TTLs when Hyperion is being tele-operated. Finally, IPC messages may be of variable length. The transmission characteristics of a variable length message type with a constant TTL are currently unknown.

7.4 Performance under Varying Data Rate

Each test performed in this study involved a constant available data rate. Our testbed does not yet fully support data rates that vary over time. The testbed would need to be updated with the capability to simultaneously update the shaped link data rate while notifying the Telemetry Manager's data rate sensor of the new rate.

7.5 Mapping Regions by Communication Link Quality

Telemetry scheduling has many practical benefits such as safer tele-operation and more efficient data logging. In addition, sensing available network data rate in real time as a robot explores a region enables generation of a map linking robot location to communication link quality. While Hyperion's research is currently focused on planning paths to optimize solar power, exploration robotics could benefit from planning based on finding paths with maximum communication availability. A data rate sensor on board a mobile robot could map out terrain occlusions or positions that suffer from

multipath. This is especially beneficial in planetary operations where an orbiting communication satellite is usually involved.

8. Conclusions

This paper addresses the issue of data rate sensitive telemetry scheduling and how it benefits tele-operation of mobile robots. Telemetry scheduling ensures that messages reach their destination within their time to live (TTL), while avoiding priority inversion that can occur over an unmitigated network link.

The architecture of the Telemetry Manager is presented with a focus on how this architecture is being implemented on a real mobile robot. An initial evaluation of the Telemetry Manager shows that it is much more capable of transmitting high-priority messages within specified TTLs than control scenarios without a Telemetry Manager. Low priority messages are transmitted only when available data rate is high enough to permit transmission of all messages.

9. Acknowledgements

We thank our mentor Professor Dr. David O' Halloran, for his support, approval and for helping us understand and build an evaluation scheme for this implementation.

References

[Apostolopoulos00] D. Apostolopoulos, M.D. Wagner, B. Shamah, L. Pedersen, K. Shillcutt, and W.L. Whittaker, "Technology and Field Demonstration of Robotic Search for Antarctic Meteorites", International Journal of Robotics Research, December, 2000.

[Bapna98a] D. Bapna, E. Rollins, J. Murphy, M. Maimone, W.L. Whittaker and D. Wettergreen, "The Atacama Desert Trek: Outcomes", Proceedings of ICRA 1998, Vol. 1, May, 1998, pp. 597 - 604.

[Bapna98b] D. Bapna, E. Rollins, A. Foessel and W.L. Whittaker, "Antenna Pointing for High Bandwidth Communications from Mobile Robots", Proceedings of ICRA 1998, Vol. 4, 1998, pp. 3468 - 3473.

[Mishkin98] A. Mishkin, J. Morrison, T. Nguyen, H. Stone, B. Cooper, B. Wilcox, "Experiences with Operations and Autonomy of the Mars Pathfinder Microrover," proceedings of the 1998 IEEE Aerospace Conference, March 21-28 1998, Snowmass at Aspen, Colorado.

[Postel83] J. Postel and K. Harrenstein, "Network Working Group Request for Comments: 868, Time Protocol", available via FTP at: ftp://ftp.isi.edu/in-notes/rfc868.txt

[Raylink01] IEEE 802.11b Wireless LANs www.3com.com/technology/tech_net/white_papers/503072a.html

[Roush99] T.L. Roush, V. Gulick, R. Morris, P. Gazis, G. Benedix, C. Glymour, J. Ramsey, L. Pedersen, M. Ruzon, W. Buntine and J. Oliver, "Autonomous Science Decisions for Mars Sample Return", Lunar and Planetary Science Conference XXX, 1999.

[RTI01] RTI / NDDS Web Page: http://www.rti.com/products/ndds/ndds.html

[Simmons01] IPC Distribution Web Page: www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html

[Singh00] S. Singh, R. Simmons, M.F. Smith, III, A. Stentz, V. Verma, A. Yahja, and K. Schwehr, "Recent Progress in Local and Global Traversability for Planetary Rovers", Proceedings of the IEEE International Conference on Robotics and Automation, 2000, IEEE, April, 2000.

[Tourrilhes01] Wireless Extensions for Linux - Jean Tourrilhes, web page: http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Extensions.html

[Wettergreen01] D. Wettergreen, B. Shamah, D. Apostolopoulos and W. Whittaker, "Robotic Planetary Exploration by Sun-Synchronous Circumnavigation", to appear in the proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space, June 18-22, 2001, Montreal, Canada.