Optimizing Bandwidth Utilization in Packet Based Telemetry Systems

Jeffrey R. Kalibjian
Lawrence Livermore National Laboratory

Keywords
bandwidth utilization, intelligent telemetry processing

Abstract

A consistent theme in spacecraft telemetry system design is the desire to obtain maximum bandwidth utilization given a fixed transmission capability (usually due to cost/weight criteria). Extensions to basic packetization telemetry architectures are discussed which can facilitate a reduction in the amount of actual data telemetered, without loss of data quality. Central to the extensions are the establishment of an "intelligent" telemetry process, which can evaluate pending data to be telemetered, and act to compress, discard, or re-formulate data before actual transmission to ground stations.

Introduction

In its Brilliant Pebbles Flight Experiment series, Lawrence Livermore National Laboratory found packet based telemetry architectures to be an adequate match for the transmitter hardware utilized in those missions [1], [2]. However, as with any spacecraft experiment in which phenomenology is a primary objective, there is always a desire to telemeter more image data. An analysis of our architecture, and packet based telemetry systems in general, revealed that they may be extended to more efficiently process data to be telemetered. The primary enhancement involves establishing a telemetry "monitor" process which can evaluate data queued and packetized for telemetry. The "monitor", with knowledge concerning specific mission goals, can act to manipulate data in any fashion (e.g. compression, reformulation); thereby, reducing bandwidth utilization and perhaps allowing for yet more significant data to be telemetered. It is proposed the modified architecture be utilized on spacecraft engaged in real time data collection with bandwidth limitations.

After briefly reviewing packet based telemetry architectures, this paper discusses the extension of such architectures to include a monitor, the definition of an interface to communicate mission goals to the monitor, and data manipulation tools the monitor can utilize.

Packet Based Telemetry Software Architectures

A typical packet based flight telemetry software system will consist of two processes. The first process acts as the interface through which the flight software may request telemetry services. This first process validates the request and creates internal structures that conveniently package the request for the second process, the packetizer. The interface process and the packetizer may communicate in a number of ways; for instance, by a queue, by message, etc. The packetizer breaks telemetry data into packets. While this is being done, the data is placed at a memory location which is accessible to the hardware that accomplishes the encoding of the digital data on the transponder carrier (this can be considered done in the transmitter). See Figure 1.

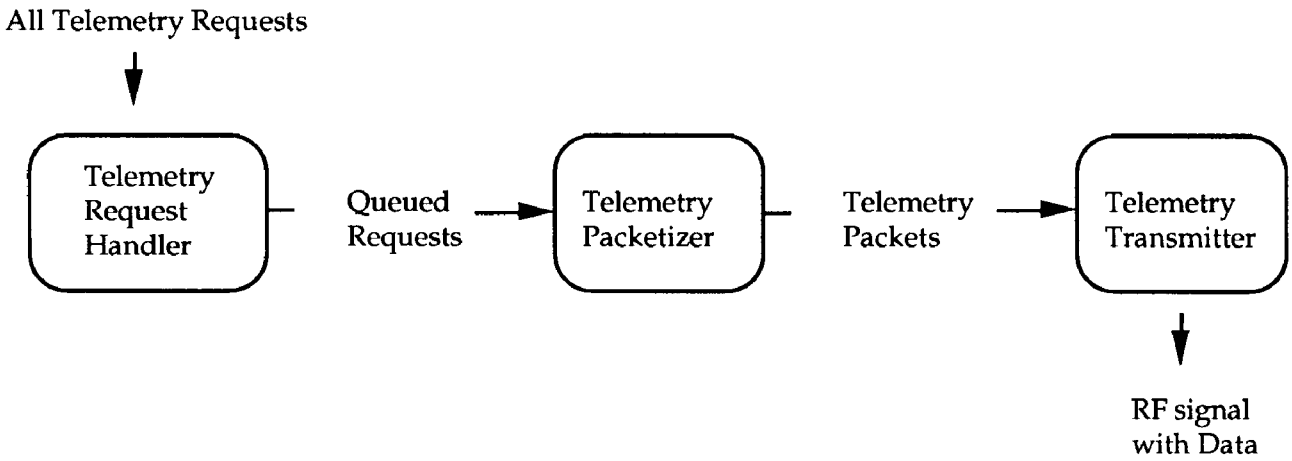

Figure 1. Software architecture for packet based telemetry system.

The telemetered packets will contain a packet header and a packet trailer. The packet is of variable length up to some maximum size. In order for the telemetry decoder to successfully extract data from the packets, an input file (sometimes known as the decoder types file) defines the structures telemetered in each packet type.

Intelligent Telemetry Processing

Intelligent Telemetry Processing (ITP), carried out by a Telemetry Monitor (TM), may not be appropriate for every mission application. ITP is oriented toward missions in which real time data acquisition is a priority, but 1) telemetry bandwidth is limited and 2) on board data storage (beyond CPU RAM) does not exist.

Figure 1 depicted a typical software architecture implementing packetization. Intelligent Telemetry Processing could intervene after requests for telemetry of data are queued (A), or after initial packetization has been accomplished (B), see Figure 2. At first, it might appear that the choice for ITP interfacing would be after request queuing. This would be because of the overhead associated with decoding the packets in flight, then re-packetizing. However, the one advantage of choosing B

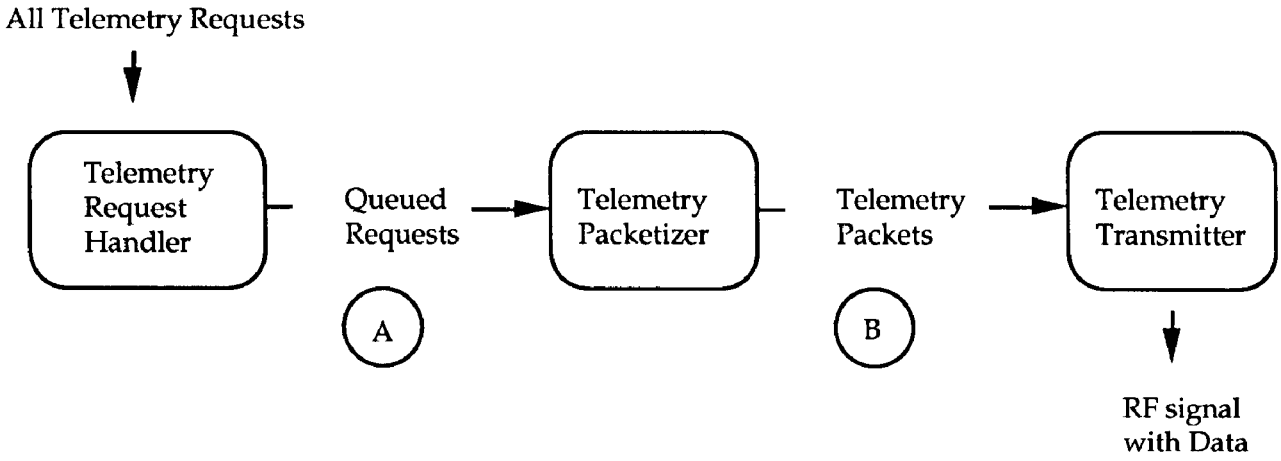

Figure 2. ITP access points in typical packetization architecture.

as the access point is that the interface specification already exists, i.e. the telemetry format. The Telemetry Monitor could be given access to a telemetry decoder and appropriate decoder types file; thusly, enabling it to gain access to the data it has been requested to filter.

The telemetry format could also be used to aid in the establishment of a communication protocol between the Mission Sequencer and the TM. Since the Mission Sequencer needs to communicate the telemetry items that are candidates for filtering to the TM, what better way then through the packet definitions. Consider Figure 3a. Here an example packet for gyroscope data is defined, along with a corresponding C language type definition the decoder might use to extract such data from a packet.

Nominally one expects to find a single value for each datum at a time t. However, to communicate expectations to the TM, each datum must have a minimum and maximum value associated with a relative time into an event, Figure 3b. In addition a TM control structure is introduced for communication of such items as filtering algorithm, constraints codes, etc. to the Telemetry Monitor.

Time	<time data>
Gyro x	<rate data>
Gyro y	<rate data>
Gyro z	<rate data>

```
typedef struct rate {
 float Time, Gyro_x,Gyro_y, Gyro_z;
} Rate;
```

Figure 3a. Example gyroscope packet definition.

Time min	<time data>
Time max	<time data>
Gyro x min	<rate data>
Gyro x max	<rate data>
Gyro y min	<rate data>
Gyro y max	<rate data>
Gyro z min	<rate data>
Gyro z max	<rate data>
TM Func 1	<TM control data>
TM Att 1	<TM control data>
TM Func 2	<TM control data>
TM Att 2	<TM control data>

```
typedef struct filterRate {
float Time[2], Gyro_x[2], Gyro_y[2], Gyro_z[2];
int TM[4];
} FilterRate;
```

Figure 3b. The definition needed to convey expected gyroscope data to a Telemetry Monitor.

Observe that the TM can use a slightly modified telemetry decoder types file, along with the same telemetry decoder to read the expectation information. Once read by the TM, the expectation data for each packet can be placed in an Action Linked List (see Figure 4), where it may be referenced by the Telemetry Monitor.

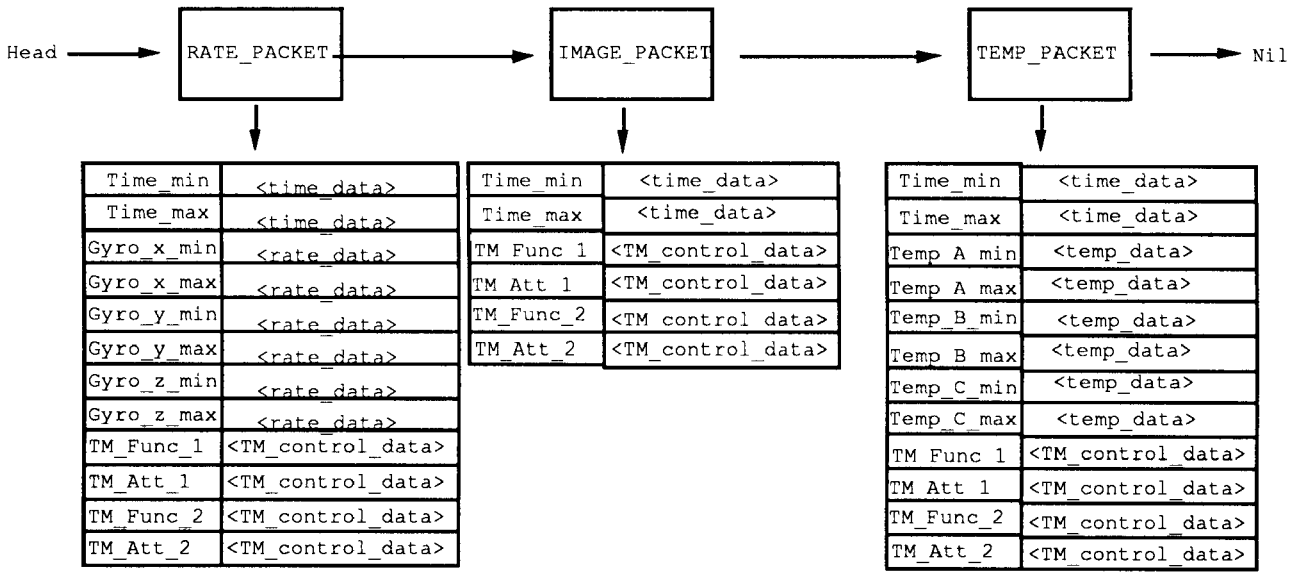

Figure 4. An example of a TM Action Linked List. In this case, operations for three packet types have been specified.

Telemetry Monitor Operation

The elements which comprise the Telemetry Monitor are depicted in Figure 5. A Sniffer process monitors packets being produced by the telemetry packetization software. The Sniffer process has access to the Action Linked List, which describes the packets to be filtered. When a packet of interest is detected, it is not permitted to proceed to the transmitter; instead, it is placed on a filtering queue (one queue exists per type of packet being processed). Queuing will always occur except when the packet encountered is an image packet. If the first packet of an image is detected; while another image is currently being operated on, it will be allowed to pass to the transmitter. This helps balance the computational load. Depending on processor speed, it may be possible for the TM to operate on more than one image; however, computational evaluations must be made before enabling this mode of operation (to insure CPU hogging will not occur).

A filtering engine will perform the appropriate operations on the queued data. Packets will be read off the queue, decoded, data operated on, and intermediate result sets placed on an output queue (one output queue per packet type being processed). When processing has been completed on a packet type (signaled by either end of event time, or a milestone completed), the re-formulated data is posted to the Telemetry Request Handler.

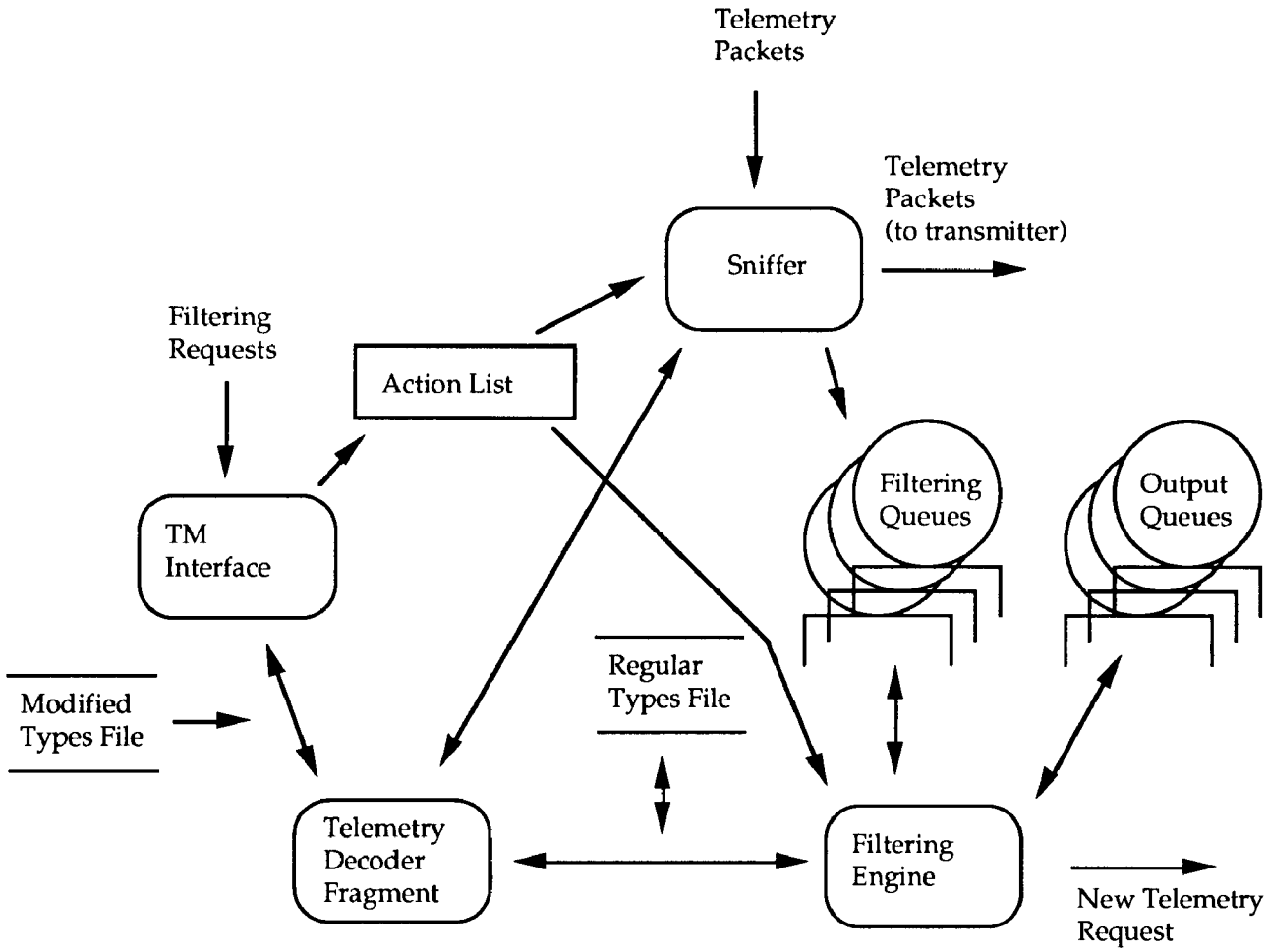

Figure 5. Elements of a Telemetry Monitor.

Monitor Application Functions

Data filtering/removal is only one application the TM might exercise. Another is data compaction. The compaction algorithm may operate on any type of data, although it will most often be applied to images. Many compaction algorithms can be employed e.g. run length encoding, etc. The selection of the appropriate algorithm usually depends on the data type (e.g. image vs. measured/predicted data) and available CPU.

Data aggregation is another possible TM function. In this application, packet data is not only filtered but aggregated over a specified interval. The results of the aggregation are telemetered on every interval boundary. Note that special aggregation data types might need to be utilized to telemeter the results of the aggregation. These are usually the union of the base data type and a statistical data structure.

Expectation Failure

When the TM detects an anomalous scenario (i.e. the Mission Sequencer specified expectations are not met during an event), emergency actions must be taken to insure all necessary data is transmitted to ground stations for evaluation. As previously mentioned, when a packet is selected for filtering, the TM establishes a queue of processed packets for that packet type. The queue size may vary depending on the packet supported. When an expectation is not met on the packet, the TM disables packet filtering and immediately sends the queued packets (containing the original data over the specified interval) to the transmitter for transmission.

Example Scenario Using ITP

A sample scenario is now illustrated. A spacecraft with ITP software will photograph a thrusting satellite. In order to maximize image throughput the Mission Sequencer instructs the Telemetry Monitor to compress as many telemetered images as possible during the event. First, the Mission Sequencer will pass a structure to the Telemetry Monitor indicating that an attempt should be made to compress image data over a specified interval. The Monitor will use a modified portion of the telemetry decoder software to process the request and place the request on the Action Linked List. When the interval for compression arrives, the Sniffer begins searching for image packets. When the first image packet is found, the Sniffer places the packet on the appropriate Filtering Queue. Next, the Filtering Engine will invoke the decoder to gain access to the image data and place the intermediate results onto the appropriate Output Queue. As subsequent packets (making up the image) are encountered they are also stored on the Filtering Queue, decoded and integrated with the image being stored on the Output Queue. If a new image is detected by the Sniffer, it will allow the packets to pass to the transmitter, since the current image it is working on has not been completed. After the last packet of the image that is being filtered is encountered, the Filter Engine will operate on the now complete image. The Filter Engine determines the algorithm to be used by examining the appropriate entry in the Action Linked List. Once processing is complete, the modified image will be posted to the Telemetry Request Handler for re-packetization.

Conclusions

A Telemetry Monitor application can substantially increase throughput of key telemetry data during important spacecraft events. While the TM concept is not applicable to all missions; it is particularly useful for satellite systems engaged in real time data acquisition with limited bandwidth and data storage capabilities.

Acknowledgments

The author gratefully acknowledges Tim J. Voss for his review of this paper.

References

[1] J.R. Kalibjian, A Packet Based, Data Driven Telemetry System for Autonomous Experimental Sub-Orbital Spacecraft, 1993 International Telemetering Conference (ITC) Proceedings.

[2] J.E. Hoag, J.R. Kalibjian, D. Shih, E.J. Toy, Recovery of Telemetered Data by Vertical Merging Algorithms, 1994 International Telemetering Conference (ITC) Proceedings.