Recovery of Telemetered Data by Vertical Merging Algorithms

Joseph E. Hoag Jeffrey R. Kalibjian, Dwight Shih
Lawrence Livermore National Laboratory

Edward J. Toy
EL and Associates, Inc.

Keywords

data recovery, telemetry post processing

Abstract

A long standing problem in telemetry applications is the recovery of data which has been damaged during downlink. Data recovery can be significantly improved by telemetering information in a packet format which employs redundant mechanisms for data encapsulation. A simple statistical algorithm (known as a "merge" algorithm) can be run on the captured data to derive a "least damaged" data set.

Introduction

One of the requirements for Brilliant Pebbles Flight Experiment Three (FE-3) was to develop a telemetry architecture providing robust data recovery if downlink problems were encountered. A checksumed packet based system with circular buffers was implemented [REF001]. This architecture easily facilitated the transmission of redundant data to ground stations. A class of post processing algorithms were developed which made use of the redundantly transmitted data to correct errors in packet transmission. These "merge" algorithms could also make use of application specific data to aid in the error detection and correction process.

After briefly reviewing the FE-3 telemetry architecture, this paper discusses the vertical merge algorithm developed to recover FE-3 data, and presents actual data recovery results. Finally, a horizontal merge algorithm that can also make use of additional ground station captures of a telemetry product is briefly described.

The FE3 Telemetry Stream Structure

FE-3 data was telemetered in packets which could either contain raw data or a data circle. A circle is a ring whose elements comprise a certain data type. Each circle has an "in pointer" (referred to as inPtr) associated with it. The inPtr represents the point in the circle at which the next new piece of data can be inserted. When the ring is full, new data always overwrites the oldest data present.

Telemetry redundancy is achieved by assigning each data type in a circle a redundancy count c and a period p. A circle is telemetered (c + 1) times every p milliseconds. Thus, a c of 0 implies "no extra redundancy,", a c of 1 implies"l extra packet of redundancy,"etc.

As an example of how circles work, assume a hypothetical circle with 7 records. Observe Figure 1; it shows how a circle changes through time by looking at consecutive transmissions of the circle in the telemetry stream. The arrows represent the circle's inPtr, the square at the bottom of each circle contains the shading of the elements added to the circle for that transmission of the circle.

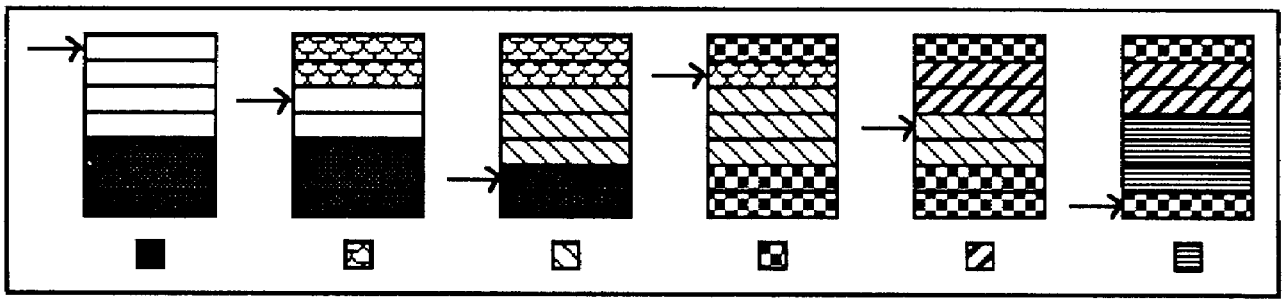

Figure 1: Circle progression, c = 0.

Figure 2 shows the same hypothetical circle, with a redundancy count of 1. Each circle is repeated once every time it is output.

As shown by Figures 1 and 2, any given record in a circle is also present in at least one more transmission of that circle[1] . This has some important ramifications:

1. If any particular packet of circle data is lost, the new information in that packet can probably be recovered by looking at the next similar packet.

[1] Assuming packet overwrite does not occur. See SectionImplications of Time stamped Circles.

2. If several packets in a row (of a particular type) have bad checksums, it is conceivable that individual records can be "patched" by looking at surrounding similar packets.

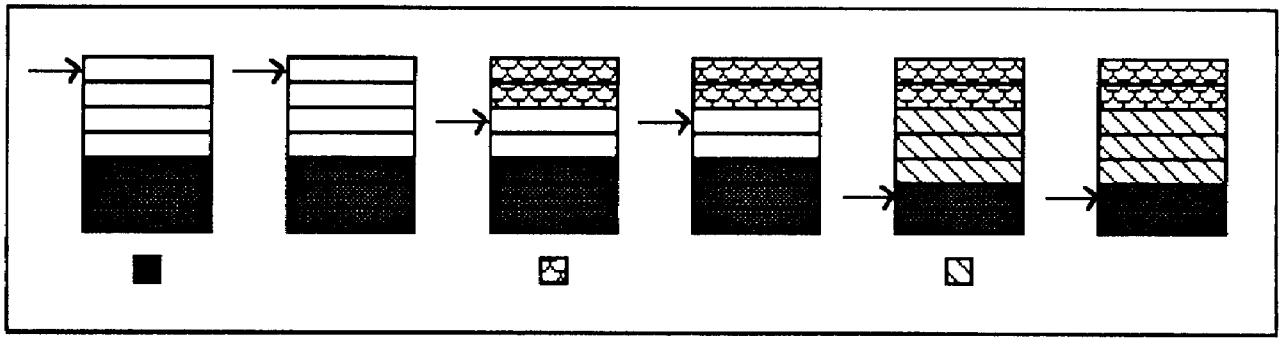

Figure 2: Circle progression, c = 1.

The FE3 telemetry decoder, tdfe3, takes advantage of the first point mentioned above. It can easily recover from one-time hits in circle data because of the redundancy built into the system.

In order to recover data from streams with multiple packet errors, one needs to exploit the second point mentioned above; that is, patch up individual records (or words or bytes) in a packet by examining surrounding similar packets. The vertical merge algorithm provides this capability.

Vertical Merge Operation

The vertical merge algorithm (known as vmerge) organizes the entire telemetry Stream into one long doubly-linked list of packets. This is called the packet list, and includes every packet in the stream. Similar packets (i.e. packets containing data for the same circle) are also linked together in similar packet chains. In Figure 3, these concepts are illustrated with 3 packet types: A, B, and C. The arrows represent the packet list, and the dashed lines represent similar packet chains.

The algorithm examines the entire list of packets, one packet at a time. It performs a "vote" on certain information in the head packet, outputs that packet, and increments the "head" pointer. This process goes on until the telemetry stream is exhausted.

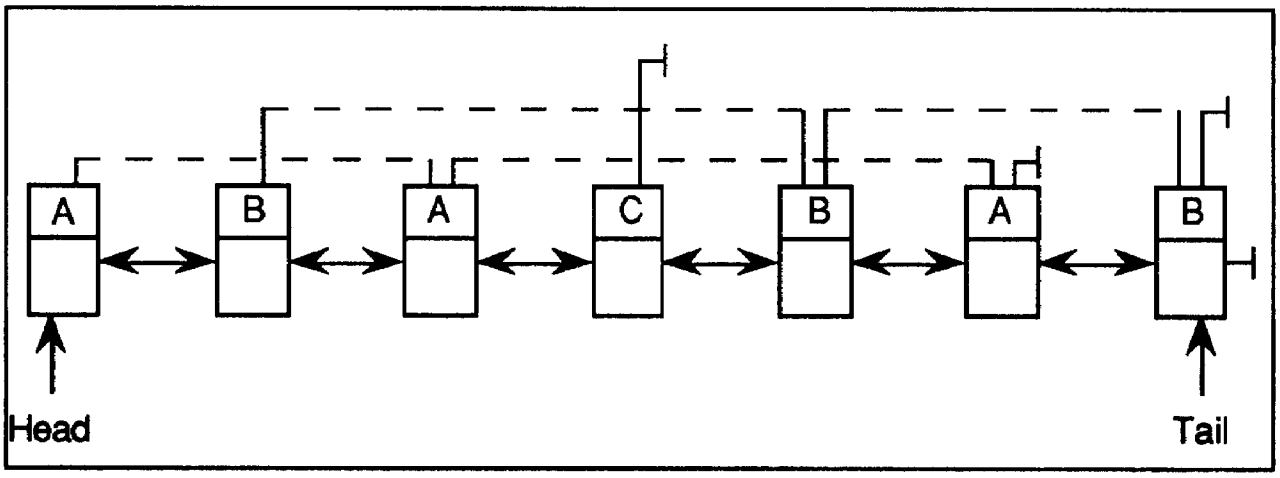

Figure 3: Sample packet list in a vertical merge.

Let packet p' represent the packet at the head of the list at some particular instant. Assume packet p' is of data type t. Vmerge will perform the following sequence to "vote" on p':

1. Ascertain the new circle elements in p'. For example, if the last inPtr for circle type t was 2, and the inPtr for p' was 5, then 3 new elements (elements 2-4) would be added to p'. If there were no new elements in p', then p' is discarded, the "head" pointer is advanced, and this step is repeated.

2. Look ahead, on the similar packet chain of p', and determine who will "vote" on the new information in p'. For example, consider packets a-f of a hypothetical 7 record data circle shown below (Figure 4, 0 = top slot, 6 = bottom slot). The arrows represent the inPtr for the packets, and the dots represent "brand new data." Suppose that a vote was desired on the packet (a) below. The records in question would be 4, 5 and 6, the three new pieces of data. Packets (a) and (b) would be involved in the vote on record 4 since packet (c) overwrites record 4 with new data. Packets (a), (b) and (c) would vote on records 5 and 6, since they all have identical copies of those records.

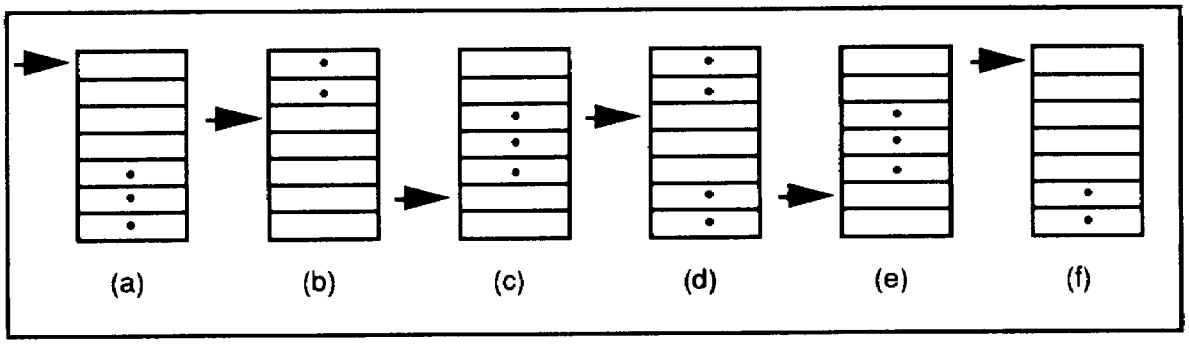

Figure 4

3. Go through the new records in p', one 32-bit word at a time, and see if all "voters" concur as to the value of each word.

- If they all agree for a word, then take no action and move on to the next word.

- If there is a disagreement between voters as to the value of a particular word, then perform a vote on that word. If there is 60% agreement on what the word's value should be, then change the word to that value in all voters[2].

4. Output p' and advance the head pointer. It should be noted that before this output was initiated several further checks could be made on the "sanity" of the vote. First, after the correction, the packet checksum could be recalculated. If the checksum now matched, one could feel fairly strongly that the correct action was taken. As another step, one could cast the corrected data to its appropriate type and insure that it fell within the expected bounds of the data type.

Implications of Time Stamped Circles

In virtually all circle data, each record will have a field within it denoting the time at which that record was most recently written in the circle. This is called the time stamp field. During a vote, there is special handling for time stamp fields.

When records are voted upon, the time stamps for those records are checked in all of the participant voters. If vmerge comes upon a time stamp that is different than the initial voter's (p' in the above example) time stamp, then voting participation stops there. For example, suppose packets 10 - 14 were the nominal "voters" for a record (record 0 in Figure 5). Packets 10 - 12 showed identical time stamps (t1) for this record, but packet 13 showed a different one (t2). The vote for record 0 would then be restricted to packets 10-12.

The reason for checking time stamps is to guard against the following:

- The inPtr for the packet might have completely wrapped around. Without time stamp checking, the record in question would have been over-written

[2] An important caveat is that packets with good checksums will never have their data changed. This prevents "voting out" of good data.

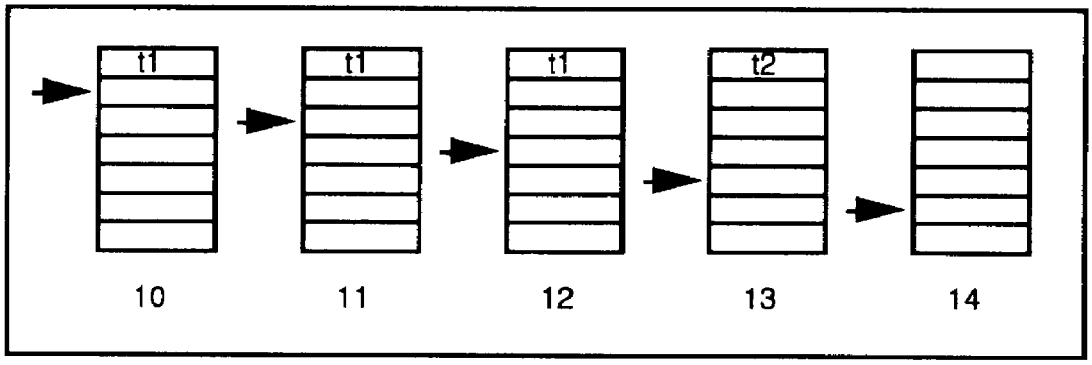

Figure 5: Sample bad time stamp in vote.

by other data, and vmerge would spuriously detect an error where there was none. Thus, a completely wrapped inPtr can be detected by its time stamp.

- There could have been a long gap -- some sort of telemetry hit -- between two packets. This would also result in spurious error reports.

The down side of time stamp checking is that vmerge "trusts" the time stamps, and does no checking on the time stamps themselves. Thus, mangled time stamps will pass through vmerge undetected. To guard against this the time stamp field could be cast to its appropriate type, then checked against its expected bounds.

Vertical Merge Test Results

In order to test the effectiveness of the vertical merge, a laborious procedure was followed. To begin with, a normal, uncorrupted telemetry stream, S, was selected as a baseline. The stream was corrupted in some manner, yielding S_c. Next, S_c was then decoded in the normal manner, producing S_{cd} (corrupted, decoded). Then, a merge was performed on S_c to produce S_{cm}. Finally, S_{cm} was decoded to yield S_{cmd} (corrupted, merged, decoded). By comparing the output of S_{cmd} to S_{cd}, we were able to draw conclusions about the effectiveness of the merge.

The general "health" of a telemetry stream was ascertained in two ways. First, a "data drop" utility was used to detect the number of dropouts in a telemetry stream, and the length in time of each dropout. Secondly, the lengths of the decoded corrupted output files and the lengths of decoded "pure" files were recorded to get an idea of the amount of new data that was recovered in the merge process.

There were two basic ways in which a file was corrupted; first, it could be "peppered" with regular or random single-byte dropouts, or, second, it could drop out whole

blocks of bytes periodically (referred to as "block spiking" or "block-at-a-time" corruption). Both techniques were employed to gain some insight into the types of corruption that vmerge was best at correcting.

There were a large number of circle types in the FE-3 telemetry Stream. Four representative ones were selected for monitoring during testing; namely, trFlyerPos, which contained data regarding the probe's position; tracker0, which contained target tracking information; tgTrackerInput, which held tracker coordination information; and teCpuADiag, which contained telemetry diagnostics.

Tables 1 and 2 reflected the effectiveness of the vertical merge on "peppered" data. The "p1000" columns represented a file with a byte hit every 1000 bytes; the "p2500" columns represented the same file with byte hits every 2500 bytes. The "pure"[3] column provides a baseline by giving the same figures for the uncorrupted version of the stream.

The data in Table 1 are of the format "number of data dropouts/aggregate dropout time." In certain cases, the vertical merge actually created more dropouts, by dividing long dropouts into several shorter dropouts. In no case, though, is total dropout time increased by doing a vertical merge. In fact, the dropout time is decreased significantly.

Table 2 shows the lengths of the ASCII output files for each circle. As in Table 1, circle output is never damaged by a vertical merge. In some cases, like tracker0, the benefits reaped by vmerge are very dramatic.

	$p1000_{cd}$	$p1000_{cmd}$	$p2500_{cd}$	$p2500_{cmd}$	pure
trFlyerPos	21/15.50	12/13.55	5/12.15	5/12.15	1/11.35
tracker0	26/30.45	34/18.90	46/21.70	17/14.75	1/11.40
tgTrackerInput	2/11.75	2/11.75	2/11.75	2/11.75	2/11.75
teCpuADiag	87/44.07	18/7.56	19/7.94	2/0.83	0/0.00

Table 1: Dropouts for "peppered" stream

	$p1000_{cd}$	$p1000_{cmd}$	$p2500_{cd}$	$p2500_{cmd}$	pure
trFlyerPos	50003	54026	56831	56838	58440
tracker0	31285	208455	180291	256423	290555
tgTrackerInput	22223	22230	22223	22230	22219^4
teCpuADiag	159890	219486	218832	229742	231016

Table 2: ASCII output file size for "peppered" stream

[3] Observe that the pure file may not yield clean results. This is a consequence of scheduling difficulties by the telemetry flight software. That is, a circle can be overwritten before being written out to the telemetry stream. See [REF001].

Tables 3 and 4 show the same results for a stream that has been "block spiked." The "b50k" stream has had a 256-byte block corrupted every 50,000 bytes. The "b100k" stream has had a 256-byte block corrupted every 100,000 bytes. Again, the results of the decode of the uncorrupted file are included in the "pure" column for a baseline measurement.

	b50k$_{cd}$	b50k$_{cmd}$	b100k$_{cd}$	b100k$_{cmd}$	pure
trFlyerPos	2/11.55	2/11.55	1/11.35	1/11.35	1/11.35
tracker0	4/12.00	4/12.00	1/11.40	1/11.40	1/11.40
tgTrackerInput	2/11.75	2/11.75	2/11.75	2/11.75	2/11.75
teCpuADiag	0/0.00	0/0.00	0/0.00	0/0.00	0/0.00

Table 3: Dropouts for "block spiked" stream

	b50k$_{cd}$	b50k$_{cmd}$	b100k$_{cd}$	b100k$_{cmd}$	pure
trFlyerPos	58040	58047	58441	58448	58440
tracker0	284561	284568	290556	290563	290555
tgTrackerInput	22219	22226	22220	22227	22219
teCpuADiag	231016	231023	231017	231024	231016

Table 4: ASCII output file size for "block spiked" stream

The decoder itself (tdfe3) did such a good job[4] with this type of data corruption that the vertical merge made very little improvement. In fact, for the circles shown, no improvement at all was made by vmerge. Tdfe3 completely restored the b100k stream on its own, and very nearly restored the b50k stream. Thus, the vertical merge buys one very little when trying to fix "block-at-a-time" type corruption.

Vertical Merge Field Results

Before the FE-3 launch from Wallops Island, a number of "dress rehearsals" were run to test out the telemetry system. Because the vehicle was telemetering from the ground, and because there was considerable competition in the RF arena, the telemetry stream was corrupted very badly during these "dress rehearsals."

Since the corruption was severe and of the "peppering" variety, the vertical merge dramatically improved the health of the telemetry stream. This improvement can be observed in Tables 5 and 6. Note that the entire cntrlCycle circle was restored by the vertical merge, while significant portions of the other two circles shown were restored (observe that there is no "pure" baseline, since this is not a contrived example).

---

[4] Again, the reason why tdfe3 handles "block spikes" so well is because most of the circles were specifically designed to handle data for at least three sample times; thus, when the block spike takes out a packet (remember a circle is telemetered upon an update), the data is recovered when the next packet of a similar type is encountered, i.e. the decoder finds the circle pointer incremented by 2 from the prior time it encountered the circle.

Similar results were observed at the Nevada Test Site (NTS), where the FE-3 flyer was placed on a tether for captive intercept tests.

	wif1008$_d$	wif1008$_{md}$
cntrlCycle	264/193.70	0/0.00
teCpuADiag	169/84.50	82/35.76
teCpuBDiag	176/82.12	81/35.03

Table 5: Dropouts for "dry run" stream

	wif1008$_d$	wif1008$_{md}$
cntrlCycle	484780	2116079
teCpuADiag	334392	413480
teCpuBDiag	339396	412604

Table 6: ASCII output file size for "dry run" stream

Horizontal Merging

Another merge concept involves the use of duplicate captures of a telemetry stream by different ground stations Such a merge is known as a horizontal merge -- since the merging is occurring across the captured streams. In a horizontal merge (hmerge) algorithm, one needs to "synch up" the various copies of a telemetry stream, so that one can compare supposedly identical packets and draw conclusions about which stream contains the best copy of that packet. Depending upon flight telemetry hardware, the complexity of this synchronization process can vary. For the FE-3 mission, in which the telemetry hardware made it possible to transmit "stale" packets to ground stations, synchronization turned out to be somewhat complicated.

Conclusion

The vmerge algorithm makes use of telemetered circle data from a damaged telemetry stream to arrive at a merged telemetry product. It works particularly well on streams that have been "peppered" with many one-byte hits. The key to the algorithm is the collection and voting of word data having the same associated time stamp. The result is a telemetry product that, when decoded, will have a great deal less data gaps than a non merged decoded stream. Vmerge has less of a dramatic effect on streams which contain "spike" hits. This is because the FE-3 telemetry decoder can still extract most lost data due to the fact the FE-3 circles typically span at least three data samples.

In our dry runs at Wallops Island, and in our tether tests at the Nevada Test Site (NTS), our telemetry experienced a great deal of "peppering" type corruption. The vertical merge algorithm was effective at restoring these streams, and thus proved to be a very valuable tool for data recovery operations.

Acknowledgements

The authors gratefully acknowledges the following individuals: D. R. Antelman, W. W.
Brown, N. J. Cushing, J. Fleming, B. Hedeline, B. Henderson, M. Henderson, R. E. Lunow, J. A. Markevitch, G. G. Preckshot, J. J. Rhodes, F. Y. Shimamoto, T. J. Voss, B. A. Wilson, J. T. Weir, J. J. Yio, and J. Young.

References

[REF001] J.R. Kalibjian, A Packet Based, Data Driven Telemetry System for Autonomous Experimental Sub-Orbital Spacecraft, 1993 International Telemetering Conference (ITC) Proceedings.