Error Correction in HDLC without Retransmission

Abdou Youssef
Department of CS
The George Washington University
Washington, DC 20052, U.S.A.

Alan Ratner
National Security Agency
9800 Savage Road
Fort Meade, MD 20755, U.S.A.

Abstract *In this paper we develop an effective and efficient technique for HDLC symbol-error recovery without retransmission. This is very useful for several applications, such as monitoring and surveillance, especially of image document facsimile transmission. Our technique is rather general and works efficiently at a 100% error recovery rate under any modulation scheme as long as the symbol-error patterns that may occur are known a priori. This is the case under the V.27 and V.29 modulation standards, where all the error patterns have been identified.*

Keywords: Error recovery, HDLC, fax transmission.

1 Introduction

It is an unavoidable fact of life that data transmission is subject to interference and noise. A variety of approaches to error management and recovery have been developed and employed, ranging from retransmission to sophisticated error-correcting codes (ECC) [BLA83]. However, in many applications retransmission is not possible and little ECC protection is used. Examples of applications where retransmission is not an option include law enforcement transmission monitoring, surveillance activities, and retrieval from partially defected disks.

A primary application where little ECC protection is used is fax transmission over telephone lines. The communication protocol is the well-know and widely used HDLC protocol, which essentially partitions the data bitstream into 256-byte pieces (called frames), and adds 2 CRC bytes to each frame for error detection but not correction. The frames are separated with a one-byte flag of value 01111110. In order not to mistake substrings of value 01111110 within frames as flags, 0-stuffing is applied to the frames: inserting a 0 after every sequence of 5 consecutive 1s. Of course, errors can corrupt legitimate flags and/or create flase flags, thus making error recovery the more difficult.

It is, therefore, of interest to be able to recover from errors in HDLC data communication without relying on retransmission or any additional ECC capabilities. This problem is very challenging, and is made even more difficult in situations where modulation is used, as in fax over telephone lines, because errors are symbol errors spreading over multiple bytes, rather than single-bit errors. While a large body of work about error management exists, such as ECC [BLA83], joint source-channel coding, and error resiliency techniques, no work has been done on symbol-error recovery in HDLC without retransmission.

In this paper we develop an effective and efficient approach to HDLC symbol-error recovery without retransmission. The approach is rather general and works effectively under any modulation scheme as long as the symbol-error patterns that may occur are known a priori. This is the case under the V.27 [CCI84] and V.29 [CCI88] modulation standards, where all the error patterns (about a dozen) that may occur have been found by one of the authors [RTN96]. The approach will work under the newer modulation standards, such as V.17 and V.34, if the error patterns that may occur under those schemes are determined. This is the subject of future work.

The paper is organized as follows. Section 2 presents our error recovery approach. Section 3 gives our performance evaluation results. Finally, conclusions and future work are presented in Section 4.

2 The Error Recovery Approach

Our approach involves three principal components:

1. The error-detection module that detects the presence of errors in segments of the data.
2. The set of all error patterns that might occur.
3. The error-correction module, which corrects the errors in each data segment that was reported to be in error by the error-detection module.

In this paper, the error detection is done by the CRC checker of the HDLC protocol, and the segments are HDLC frames. The set of error patterns depends on the modulation scheme. For the case where modulation follows the V.27 or V.29 standards, the set of error patterns have been derived in [RTN96], and their probabilities of occurrence have been computed. (Those error patterns, each 32 bits long, would be included in the final version of the paper, for space limitations.) The main contribution of this paper is the error-correction module. To carry out error correction, we first develop a classification of the errors based on their effects and locations, and then write an algorithm that classifies each error and conduct class-based error correction. The next subsection gives the error classification, and subsection 2.2 presents the algorithm.

2.1 Error Classification

In the course of developing the error-correction module, we found it necessary to develop a new classification of errors, depending on the error locations and their impact. The new classes of errors are:

- **In-frame errors**. These occur when an error pattern falls fully within a frame.
- **Flag-saving cross-frame errors.** Those occur when an error pattern

overlaps with an inter-frame flag, or crosses from one frame to the next, without flipping any of the flag bits.

- **Flag-corrupting cross-frame errors.** Those are similar to the previous class except that they corrupt the flag. They cause incorrect deframing because of frame merging.

- **Destuffing errors.** They occur when an error pattern flips some of the 0-stuffing bits or some of the five 1's preceding a stuffing bit. In the first case, a false flag may be created, thus splitting a frame into two false frames. In the second case, the destuffer is fooled and does not destuff some of the stuffed 0 bits.

2.2 Error Correction

Our error-correction approach handles all those error classes successfully and efficiently. It is described in this subsection. For the sake of presentation clarity, we assume at most one error-symbol per frame, and we occasionally make reference to V.29 error patterns. However, the algorithm can be easily modified to handle more than one symbol-error per frame, and applies to other modulation schemes as long as the error patterns are known. The presentation and discussion of the algorithm follows.

In each frame reported to be in error by the CRC checker, every error pattern is tried in every bit location in the non-destuffed frame. The error patterns are tried in their decreasing probability of occurrence. After every trial, the destuffer and CRC checker are called. If no errors are reported, the frame is considered corrected; otherwise, another error pattern is tried from the current bit location. If all bit locations have been tried with the current error pattern without success, the next error pattern (in the ordered list of patterns) is tried, starting from the first bit in the frame. This is done until the frame is reported correct or all the patterns and bit locations have been tried. If the error is an in-frame error and the end-flag is a legitimate flag, then the error-correction module as outlined thus far will surely correct the error. If, on the other hand, the error belongs to a different class, other measures are taken, as described next.

Consider flag-saving cross-frame errors. A pattern can cross a frame boundary over a flag without corrupting it if the pattern has at least 8 consecutive 0s. This is indeed the case with each of the error patterns in V.29. Of course, such flag-saving crossing can happen if the error pattern starts at a bit location so that an 8-zero run of the pattern coincides with the flag. For each error pattern, those bit locations are easily determined. Luckily for efficiency, they are very few in number. Note that a pattern may overlap a flag without crossing it to the next frame. The flag can still be left intact if the pattern has only 0s in the pattern-flag overlap region. This is the case with 6 of the patterns: 3 end with just one 0, and three end with two 0s. In any case, the handling of cross-frame errors is done as follows.

Let F be a frame corrupted by an error pattern that crosses or just overlaps its end-flag without corrupting it. This frame is correctly deframed, but the error-correction module cannot correct the error yet because the error in it is a partial error pattern that does not exist explicitly in the collection of error patterns. Remedying this situation is, however, rather straightforward. Prefixes of error patterns are tried at the end of frame F, and, when necessary, corresponding error-pattern suffixes are tried on the following frame. All appropriate prefixes of all different error patterns are tried at all the appropriate bit locations near the end of F. At each trial, the CRC checker is called to check for errors. This is done until the frame is corrected or all the possibilities are exhausted. Another way for dealing with cross-flag errors is to try different error patterns in a small region surrounding the flag. After each pattern application, if a flag emerges, it is treated as a candidate legitimate flag, and the frame preceding it is processed. This has the advantage of addressing both flag-saving and flag-corrupting cross-frame errors in one unified way, presented next with a focus on flag-corrupting errors.

In the presence of flag-corrupting cross-frame errors, the corrupted flag causes the surrounding frames to merge into a large, false frame. To avoid missing such a flag, we exploit the fact that the flag must be in a small region R near the 2096-th bit after the previous legitimate flag. Flag recovery is then performed in that region as follows. For each error pattern, the pattern is applied at each bit location in region R. Whenever a flag emerges, the tentative frame before it is destuffed and CRC checked, and, if needed, in-frame error correction is attempted on it. If this succeeds, then the current flag is taken to be the correct flag, the frame is considered to be corrected, and the next frame is processed afterwards. If it fails, the loop is repeated with the next bit location, or, when at the end of the region R, with the next error pattern started from the beginning of R.

Finally, destuffing errors are handled indirectly. If a destuffing error does not create a false flag, then the number of destuffing error bits cannot be more more 12, because no V.29 error patterns flips more than 12 bits. Therefore, the destuffer may wrongly destuff at most 12 bits, or may miss destuffing at most 12 bits. Therefore, under the at-most-one-error-per-frame assumption, such destuffing errors are detected by checking if the length of the destuffed frame differs from 2096 bits but by no more than 12 bits. In that case, the same in-frame error correction is done on the pre-destuffed frame. If, on the other hand, the destuffing error creates a false flag, it will create two false frames, each of length smaller than expected. This is remedied by always looking for the next flag only after 2096 bits from the last correct flag, in a relatively small region R, and then applying the same technique outlined in the previous paragraph. This automatically takes care of destuffing errors, including those that cause false flags.

We have assembled the above methods into an algorithm for handling all the four types of errors in a streamlined way, developed a C code implementation for it, and evaluated its performance. The pseudo code of the algorithm is presented next.

Procedure CORRECT-ERROR
Input: a potentially corrupted HDLC bitstream;
Output: an error-free bitstream, where all the HDLC bits are removed

Assumptions: V.29 modulation, with at most one error per frame. Those assumptions are not fundamental.

begin

1. *Search for the Next Legitimate Flag:* From the last correct flag, skip 2064 bits, and look for a flag in the region extending from the 2065-th bit to the 3200-th bit after that flag; this is the region where the legitimate next flag lies. Call this region R.

   Comment: The next legitimate flag starts after the 2096-th bit after the previous flag, but we extend the search region by 32 bits to the left so that if we need to correct a corrupted flag in R, we avoid applying partial error patterns for simplicity.

2. *Branching based on the Flag-Search Outcome:* If a flag is found in the region R, denote by F the frame between the previous flag and the newly found flag, and go to Step 3. If a flag is not found in region R, go to Step 6.

3. *Destuffing and CRC-Checking*: Destuff the frame F, getting a frame F'. If F' is of length 2096 bits and the CRC checker reports no errors on F', go to Step 1 to process the next frame.

4. *In-frame/Destuffing Error Checking:* If CRC fails on F' (in-frame error) or the length of F' is different from 2096 bits but by no more than 12 bits (destuffing error), then go to Step 5 to perform in-frame/destuffing error correction.

5. *In-Frame/Destuffing Error Correction*: For each error pattern and each bit position in frame F, apply (i.e., xor) the error pattern at that error position in the frame F (not F'), and call the destuffer and CRC checker. Whenever success is reported, go to Step 1 (the frame is treated as now corrected, the flag is a legitimate flag, and it is time to process the next frame). If no success is reached in all the trials of patterns and bit positions, go to Step 6.

6. *Flag-Error Correction*: For each error pattern and each bit position in region R, apply the error pattern. If a flag emerges, consider the nondestuffed frame between the previous legitimate flag and this flag. Call that frame F. Go to Step 5 and apply it on F.

end

3 Performance Evaluation

We conducted extensive performance evaluation of the algorithm for the purposes of: (1) testing for correction coverage, that is, error-correction success rate, and (2) measuring the speed of the algorithm and its C implementation. We conducted the testing on 8 CCITT test files compressed using JBIG, using a Pentium 400 MHz PC, and measured the error-correction time. To get reasonably accurate and comprehensive measurements of speed, different error rates were tried, ranging from 5% to 100% of corrupted frames, maintaining the at-most-one-error-per-frame assumption. For each frame-in-error rate, ten test runs were performed; the ten runs differ from one another in the error patterns that are injected, and in the bit locations of the errors, but the same frames are infected in all the ten runs. Note that the speed does not depend on which frames are infected; rather, it depends only on (1) the number of frames infected, (2) which error pattern infects a frame, and, to a much less extent, (3) the error location in a frame. By trying the error patterns in the order of decreasing probability, the correction time is significantly reduced on average.

The outcome of the tests met our expectations. The algorithm correctness was verified, and the error-correction success rate is 100%. The algorithm is able to correct all errors, whatever the error pattern, error location, or error class. Similarly with the speed, the code is quite fast, as shown in Figures 1 and 2. The amount of time it takes to correct in-frame/destuffing errors at an error rate of 5% (only one infected frame) ranges from less than one second to less than 3 seconds, while even at the highest rate (i.e., all the frames are infected) the time ranges from 22 seconds to 55 seconds, averaging at 38 seconds. The situation is even better for flag-corrupting cross-frame errors (Figure 2), where at the expected error rate of 5%, the time ranges from 0.5 seconds to 1.1 seconds (averaging at 0.78 second), while at the highest extreme where all the flags are corrupted, the correction time ranges from about 7 seconds to 30 seconds (averaging 19 seconds).

4 Conclusion

In this paper we developed and evaluated an effective and efficient technique for HDLC symbol-error recovery without retransmission. This kind of error recovery is very valuable for several applications, such as image document facsimile transmission. We showed that our technique achieves 100% error recovery rate under any modulation scheme where the symbol-error patterns are known a priori as in the case of the V.27 and V.29 modulation standards.

Future work involves finding error patterns for other more recent modulation standards, and for developing other error recovery techniques that do not rely on HDLC.

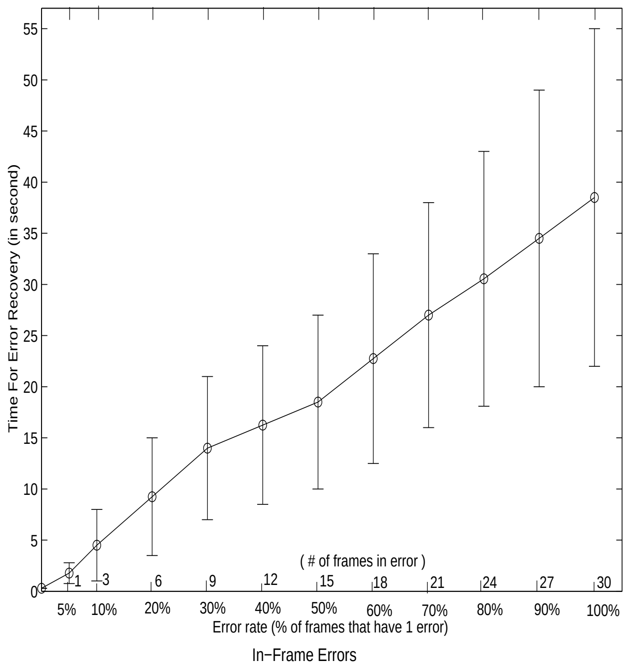

Figure 1: Speed of In-Frame Error Recovery

REFERENCES:

[ITU T.85] ITU-T Recommendation T.85, Application Profile for Recommendation T.82 - Progressive Bi-Level Image Compression (JBIG Coding Scheme) for Facsimile Apparatus, August 1995. Amendment 1,

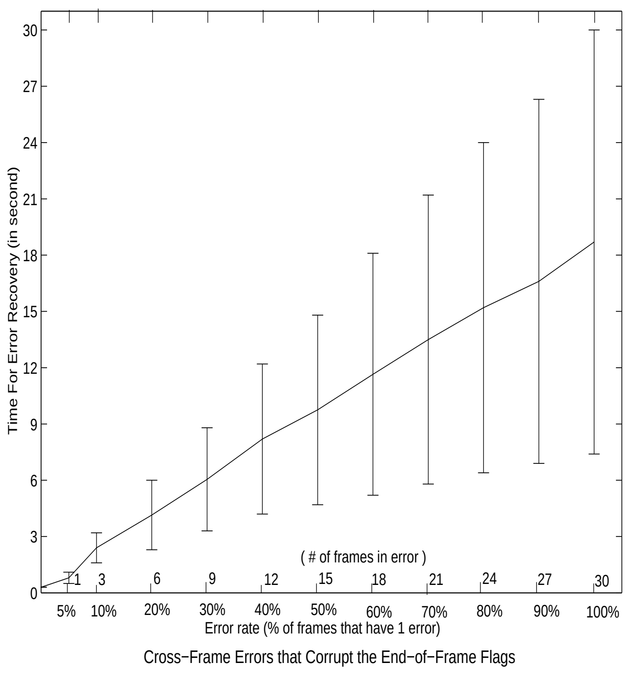

Figure 2: Speed of Recovery from Cross-Frame Flag-Corrupting Errors

October 1996. Corrigendum 1, February 1997.

[BLA83] R. E. Blahut, Theory and Practice of error Control Codes, Addison-Wesley, 1983.

[CCI84] CCITT Recommendation V.27 ter. 4800/2400 Bits per Second Modem Standardized for use in the General Switched Telephone Network, amended 1984.

[CCI88] CCITT Recommendation V.29: 9600 Bits per Second Modem Standardized for use on Point-to-Point 4-Wire Leased Telephone-Type Circuits, amended 1988.

[RTN96] A. Ratner, "Device for and Method of Correcting Errors in Formatted Modem Transmissions," Patent 5533033, 1996.