양방향 디코딩이 가능한 비트열을 이용한 앞자리 부호의 오차 검출과 정정

박 상 호*

요 약

허프만 코드를 네트워크를 통하여 전송할 때 다수의 버스트 에러를 검출하고 정정하는 방법을 제안하였다. 양방향으로 디코딩이 가능한 코드를 사용하였으며 금지된 심벌을 스트링에 삽입하여 일정한 블록마다 오류의 발생여부를 점검할 수 있게 하였고 비트를 추가하여 데이터의 크기는 증가하였으나 오류정정이 가능 하도록 하였다. 제안된 방법은 전체 파일 크기는 증가하나 실시간으로 오류를 검출하고 정정할 수 있다.

Error Detection and Correction of Prefix Codes using Bidirectionally Decodable bit Streams

Sangho Park*

ABSTRACT

This paper proposes multiple burst error detection and correction scheme for transmission of Huffman coded string. We use bidirectionally decodable codes and introduce insertion of forbidden symbol to find errors. Additional bits are added to original bit streams to correct errors. The total file size id increased but it can detect errors and recover errors real time.

Key words : Bidirectional Decode, Prefix Code

* 안동대학교 전자정보산업학부 교수

1. 서 론

통신선로를 통하여 압축된 데이터를 전송할 때 잡음으로 인한 비트 오류에 의하여 메시지가 변경되는 결과를 가져올 수 있다. 모바일 환경에서는 페이딩으로 인하여 일정한 시간 동안 데이터의 수신이 불가능하게 되는 경우가 발생하여 다량의 비트 오류가 발생한다. 회계 관련 문서나 비밀을 요하는 문서가 해커에 의한 데이터의 수정이 시도되어 수신 메시지가 바뀌는 경우도 발생할 수 있을 것이다. 이러한 수신 비트열의 오류는 심각한 문제를 야기 시킬 수 있다. 해킹으로부터 이러한 문제를 방지하기 위하여 암호화를 하고 워터마크를 사용하며 데이터 전송 시 오류가 없는 데이터를 수신하기 위하여 오류 발견 시 재전송하는 ARQ (Automatic ReQuest) 방식이 널리 쓰이고 있다.

앞자리부호(Prefix code)의 전송 시 비트오류가 발생하였을 때 양방향으로 디코딩하여 오류를 정정하기 위하여 양방향으로 디코딩이 가능한 코드가 제안되었다[1-2]. [1]은 허프만 코드의 정보를 순방향과 역방향으로 디코딩 가능하도록 하나의 affix 코드로 만드는 방법을 개발하였고, [2]는 각 심벌의 허프만 코드 비트열의 순방향 비트열과 역방향 비트열의 XOR한 비트열을 이용하여 양방향으로 디코딩할 수 있는 방법을 제안하였다. 그러나 제안한 양방향 코드가 에러의 정정에 사용될 수 있을 것이라고 제안하였으나 그 방언을 언급하지는 않았다. [2]의 양방항 코드를 이용하여 에러를 검출하고 정정하는 방법이 제안되었다[3].

산술코딩(Arithmetic Coding)으로 압축된 데이터의 오류를 검출하기 위하여 코드의 비트열에 데이터에서 발생할 수 없는 별도의 심벌을 코드에 집적하는 방법이 제안되었다[5]. 수신측에서 디코딩 하였을 때 발생할 수 없는 심벌이 나타나면 오류가 있음을 알 수 있다. 산술코드는 자체 동기 기능이 없으므로 산술코딩에 의한 비트 열들은 오류가 발생하면 코드의 특성 상 그 이후의 모든 코드는 잘못된 심벌로 복호하게 되어 치명적인 결과를 초래한다. [6]에서는 [5]에서 제안된 새로운 심벌을 추가로 포함시키는 방법을 이용하여 ARQ 프로토콜에서 산술 코드 데이터의 오류를 검출하는 방법을 제안하였다. [3]에서는 [2]의 양방향으로 디코딩이 가능한 코드와 [6]의 오류 검출방법을 허프만 코드에 적용하여 허프만 코드의 오류검출 및 정정 방법을 제안하였다. 수신된 비트열을 순방향으로 디코딩한다. 복호된 심벌을 검사하여 금지된 심벌이 나타나면 오류가 발생한 것이므로 역방향으로 디코딩이 가능한 코드를 역방향으로 디코딩하여 만든 심벌들을 이용하여 오류를 정정한다.

본 논문에서는 엔트로피 코드의 하나인 허프만 코드를 유무선 네트워크를 통하여 전송할 때 전송로 잡음과 페이딩에 의한 비트열 에러를 검출하고 정정하는 방안을 제안한다. 제안하는 방법을 [3]의 방법과 비교하면 [3]의 방법은 디코딩이 끝난 후 오류가 있는지 판단하여 오류가 발생하였으면 오류가 발생한 위치를 대략적으로 발견하고 그 주위의 비트 열들을 양방향 디코딩에 의해 정정한다. 그러나 본 논문에서 제안하는 방법은 전송하는 데이터에 일정한 수의 심벌마다 에러검출을 위한 심벌을 추가하여 디코딩 과정에서 블록단위로 양방향 디코딩에 의하여 에러정정을 한다. 제안된 방법은 디코딩 중 실시간으로 에러검출이 가능하므로 ARQ방식에서도 이용이 가능하다.

2. 양방향으로 디코딩이 가능한 앞자리 부호

본 절에서는 [2]에서 허프만 코드[4]의 양방향 디코딩을 위하여 제안한 방법에 대해 기술한다.

$S = s(1)s(2) \cdots s(N)$를 N개의 심벌들로 이루어진 스트링이라 하고, S를 인코딩한 허프만 코드 비트 열을 $B = B(1)|B(2)| \cdots |B(N)$라 하자. $B(i)$은 심벌 $s(i)$를 인코딩한 허프만 코드이며 $B(i) =$

$b_1(i)|b_2(i)|\ \cdots\ |b_{l(i)}(i)$이다. 여기서 $a|b$는 연결(concatenation)을 의미하며 $l(i)$는 심벌 $s(i)$의 코드의 비트 열의 길이이다. 코드 $B'(i) = b_{l(i)}(i)|\ \cdots\ |b_2(i)\ |b_1(i)$를 $B(i)$의 역방향 코드라 한다. 예로서 $B(i)$ = '10110011'이면 $B'(i)$ = '11001101'이다. 역방향으로 디코딩 가능한 비트 열 B'은 역방향코드들의 연결 $B' = B'(1)|B'(2)|\ \cdots\ |B'(N)$이다.

양방향으로 디코딩이 가능한 코드의 비트 열은 아래와 같이 정의하였다.

$$\begin{aligned} C &= (B|0^L) \oplus (0^L|B') \\ &= (B(1)|B(2)|\ \cdots\ |B(N)|0^L) \\ &\quad \oplus (0^L|B'(1)|B'(2)|\ \cdots\ |B'(N)) \end{aligned} \tag{1}$$

여기서 $L \geq l_{\max} = \max\ l(i)$이며 0^L은 L개의 '0' 비트 열이고 $\oplus$은 비트 별 XOR(Exclusive-OR)를 의미한다. 양방향으로 디코딩 가능한 코드의 비트 열은 순방향으로 디코딩이 가능한 비트 열 B에 L개의 '0'를 덧붙인 비트 열과 역 방향으로 디코딩이 가능한 비트 열 B' 앞에 L개의 '0'를 덧붙인 비트 열을 XOR하여 만들었다.

식 (1)에서 순방향으로 디코딩이 가능한 비트 열은 아래와 같이 만들 수 있다.

$$\begin{aligned} &B(1)|B(2)|\ \cdots\ |B(N)|0^L \\ &\quad = C \oplus (0^L|B'(1)|B'(2)|\ \cdots\ |B'(N)) \end{aligned} \tag{2}$$

역방향으로 디코딩이 가능한 비트 열은 식 (2)와 동일한 방법으로 아래와 같이 만들 수 있다.

$$\begin{aligned} &0^L|B'(1)|B'(2)|\ \cdots\ |B'(N) \\ &\quad = C \oplus (B(1)|B(2)|\ \cdots\ |B(N)|0^L) \end{aligned} \tag{3}$$

[2]의 방법을 설명하기 위하여 5개 심벌로 구성된 <표 1>과 같은 허프만 코드를 고려하자.

7개의 심벌의 스트링 S = *CBADABEA*라 하면, 순방향 코드 B = '0000110010101001 11'이고 역방향 비트 열 B' = '0001010100110110 01'이다. $C = B$ 0000 $\oplus$ 0000B' = '0000110111111110101010 01'이다. C의 비트들을 C = {$c_1\ \cdots\ c_{N'}$} 이라 하고 B = {$b_1\ \cdots\ b_N$}하자. 순방향 디코딩은 최초의 L개의 비트 $c_1 \sim c_4$ = $b_1 \sim b_4$인 '0000'에 대해 복호하면 '000'이 심벌 C가 된다. 복호된 비트 열 '000'의 역방향 코드를 C의 최초의 L 비트열 다음 비트부터 세 비트인 $c_5 \sim c_7$과 XOR하면 B의 그 다음 4비트 $b_4 \sim b_7$ = '0110'이 구해진다. '0110'에서 심벌 앞의 두 비트에서 '01'이 심벌 B가 된다. 동일한 방법으로 모든 비트들을 복호할 수 있다.

〈표 1〉 5개의 심벌에 대한 발생빈도와 허프만 코드

심벌	발생빈도	허프만 코드
A	40%	1
B	25%	01
C	20%	000
D	10%	0010
E	5%	0011

3. 양방향으로 디코딩이 가능한 앞자리코드 비트 열의 오류 검출 및 정정

본 절에서는 금지된 심벌을 이용하여 오류를 검출하고 양방향으로 디코딩이 가능한 비트 열을 이용하여 오류를 정정한다. 금지된 심벌이 나타나면 오류가 발생한 것으로 판단하도록 한 [3, 5-6]에서의 오류검출 방법과 달리 본 연구에서는 금지된 심벌을 일정한 간격마다 블록을 설정하여 데이터에 삽입한 후 금지된 심벌이 검출되지 않거나 블록에서 복호된 심벌의 수가 압축할 때의 블록내의 심벌의 수와 다르면 오류가 발생한 것으로 판단한다.

3.1 인코딩 방법

각 심벌의 발생확률이 각각 $p(k)$인 n개의 심벌로

이루어진 데이터 공간 $D = \{s_1, s_2, \cdots s_n\}$에서 N개의 심벌들의 연결(concatenation)로 이루어진 스트링 $S = s(1)s(2) \cdots s(N)$를 생성한 후 아래와 같은 방법으로 인코딩 한다. 여기서 $s(i)$는 $s_1, s_2, \cdots, s_n$ 중 하나이다.

① 발생확률이 e인 금지된 심벌을 데이터 공간 D에 추가하여 새로운 데이터 공간 D^+를 만든다. 이제 심벌의 개수는 n+1개이다. $e \leq 1$이다.

② N개의 심벌들의 연결(concatenation)로 이루어진 스트링 $S = s(1)s(2) \cdots s(N)$에 $N \times e$개의 심벌마다 금지된 심벌을 추가하여 새로운 스트링 S^+를 만든다. 스트링 S^+는 $N(1+e)$개의 심벌들의 연결로 구성되어있다.

③ 새로운 스트링 S^+에 대한 허프만 코드 $B = B(1)|B(2)| \cdots |B(N(1+e))$를 구한다.

④ $L = 3l_{max} + l_{burst} - 2$를 결정한다. 여기서 l_{max}는 허프만 코드워드 중 최고의 길이, l_{burst}는 정정 가능한 에러의 크기이다.

⑤ 허프만 코딩하여 만든 순방향 디코딩이 가능한 비트 열 $B = B(1)|B(2)| \cdots |B(N(1+e))$과 역방향 디코딩이 가능한 비트열 $B' = B'(1)|B'(2)| \cdots |B'(N(1+e))$을 이용하여 양방향으로 디코딩이 가능한 아래의 비트 열을 만든다.

$$\begin{aligned} C &= (B|0^L) \oplus (0^L|B') \\ &= (B(1)|B(2)| \cdots |B(N(1+e))|0^L) \\ &\quad \oplus (0^L|B'(1)|B'(2)| \cdots |B'(N(1+e))) \end{aligned} \tag{4}$$

여기서 $B(i)$는 심벌 $s(i)$의 허프만 코드이다. 스트링에서 심벌의 수가 $N+1$인 새로운 데이터 공간에서 각 심벌 s_i의 확률은 $(1-e)p(i)$가 된다.

3.2 디코딩 및 오류검출 방법

제 2절에서 기술한 방법과 같이 디코딩한다. 수신측에서 비트 열을 수신하면 C의 최초의 L 비트에 대해 허프만 복호를 하여 심벌을 생성한다. 금지된 심벌의 발생확률이 정해지면 금지된 심벌의 삽입 지점사이의 심벌의 개수인 블록의 크기 bs가 결정되므로 금지된 심벌이 나타날 때까지 몇 개의 심벌이 복호되었는지 확인하여 블록의 크기보다 심벌의 수가 작거나 크면 블록 내에서 오류가 발생한 것이다. 또 블록내의 심벌의 개수보다 많은 심벌을 복호하는 동안에도 금지된 심벌이 나타나지 않으면 오류가 발생한 것이다. 블록의 크기 bs 보다 얼마나 더 많은 심벌을 복호하는 동안 금지된 심벌이 나타나지 않으면 오류인가 하는 것은 스트링의 구성에 따라 달라진다.

디코딩 도중 오류가 발생한 것으로 판명되면 ARQ 프로토콜인 경우 그 블록의 시작 비트부터 재전송을 요구하고 ARQ가 아닌 경우 오류가 발생한 블록의 오류를 정정한다. 제안된 방법은 [3]과 달리 오류의 위치를 찾지 않는다. [3]의 방법에서는 오류가 검출되면 오류가 발생한 지점 이전의 모든 비트를 오류확인 하여야 하므로 대략적인 위치를 찾은 다음 그 주위의 비트들을 점검하여 오류정정을 한다. 그러나 제안된 방법은 오류가 검출된 블록 내의 비트만 오류정정하면 되므로 오류의 위치를 찾는 복잡성을 제거하고 오류를 찾는 시간을 줄이기 위하여 블록 내의 첫 비트부터 마지막 비트까지 오류정정을 수행한다. [3]과 또 다른 점은 [3]은 모든 비트열을 복호하고 난 뒤 오류 여부를 점검하므로 하나의 버스트 오류만을 정정할 수 있으나 제안된 방법은 블록단위로 오류점검이 가능하므로 최대한 블록의 개수만큼의 버스트 오류를 검출하고 정정할 수 있다.

4. 실험 및 고찰

제안한 방법의 장단점을 실험을 통하여 알아보았다. <표 1>에서 사용한 심벌에 금지된 심벌 X를 추가하였으며 X의 발생확률은 10%로 하였다.

따라서 원래의 심벌들의 발생확률은 $(1-0.1)p(i)$이므로 <표 2>와 같이 새로운 발생확률을 갖는 데이터 공간이 되었고 각 심벌들에 대한 허프만 코드도 <표 2>에 나타내었다. 전송하려고 하는 스트링은 <표 2>와 같이 금지된 심벌이 추가된 발생확률을 이용하여 허프만 코드를 발생한다.

〈표 2〉 금지된 심벌이 추가된 발생확률을 이용한 허프만 코드

심벌	발생빈도	허프만 코드
A	36%	00
B	22.5%	10
C	18%	11
X	10%	011
D	9%	0100
E	4.5%	0101

블록 내의 모든 비트의 오류정정을 위하여 L의 크기를 블록의 크기인 $N/(N\text{x}e)$로 하였다.

A, B, C, D, E의 다섯 개 심벌을 <표 1>의 확률에 따라 랜덤하게 1000개를 생성한 다음 허프만 코드로 압축하면 스트링은 2100비트이다. <표 2>를 이용하여 [3]의 방법으로 허프만 코딩하면 심벌의 코드가 2300비트, L이 400비트 이므로 스트링은 2700비트이다. 제안된 방법도 <표 2>를 이용하여 코딩하면 심벌의 코드, 2300비트 L이 300비트, 삽입하는 금지된 심벌의 코드가 400비트 이므로 스트링은 3000비트이다. 여기서 L은 한 블록의 심벌을 오류검출 및 정정할 수 있도록 하여 그 크기를 심벌의 수 와 심벌의 평균 비트수로 하였다. [3]의 방법과 제안한 방법의 스트링의 비트 열의 크기를 비교하면 금지된 심벌의 비트 수만큼 증가한다. 그러나 제안한 방법은 오류의 검출과 정정이 블록단위로 수행되므로 다수의 버스트에러를 검출하며 정정할 수 있고 그 검출 가능한 버스트 오류의 최대 수는 블록의 수와 같다. 생성된 스트링에 랜덤한 위치에 버스트 에러를 발생시켜 오류가 검출됨을 확인하였고 정정한 비트 열이 올바른 값을 가짐을 확인하였다.

6. 결 론

본 논문에서는 허프만 코드를 전송할 때 비트열 에러를 검출하고 정정하는 방안을 제안하였다. 제안한 방법을 [3]의 방법과 비교하여 장단점을 논하였다. 본 논문에서 제안하는 방법은 전송하는 데이터에 일정한 수의 심벌마다 에러검출을 위한 심벌을 추가하여 디코딩 과정에서 블록단위로 양방향 디코딩에 의하여 에러정정을 한다. 제안된 방법은 디코딩 중 실시간으로 에러검출이 가능하므로 ARQ방식에서도 이용이 가능하다. 제안된 방법은 요금에 따른 서비스의 품질을 달리하는 시스템에서 일반 품질을 요청한 가입자에게는 에러 검출 및 정정 기능이 없는 스트링을 전송하고 고품질의 서비스를 요청한 가입자에게 에러검출 및 정정 기능이 있는 제안된 방법으로 생성한 비트열을 전송하는데 사용할 수 있다.

참 고 문 헌

[1] A. S. Fraenkel and S. T. Klein, “Bidirectional Huffman coding”, *Computer Journal*, Vol. 33, No. 4, pp. 296-307, 1990.

[2] B. Girod, “Bidirectionally decodable streams of prefix code-words”, *IEEE Commun. Letters*, Vol. 3, No. 8, pp. 245-247, August 1999.

[3] M. Kitakami and S. Nakamura, “Burst error recovery for Huffman coding”, *IEICE Trans., Inf & Syst.*, Vol. E88-D, No. 9, pp. 2197-2200, Sept., 2005.

[4] D. A. Huffman, “A method for the construction of minimum-redundancy Codes”, *Proc. IRE*, Vol. 40, pp. 1098-1101, Sept. 1952.

[5] C. Boyd, J. G. Cleary, S. A. Irvine, I. Rinsma-Melchert, and I. H. Witten, "Integrating error detection into arithmetic coding", *IEEE Trans., Commun.*, Vol. 45, No. 1, pp. 1-3, Jan. 1997.

[6] J. Chou and K. Ramchandran, "Arithmetic coding-based continuous error detection for efficient ARQ-based image transmission", *Proc. Annual Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, pp. 1005-1009, Nov. 1997.

박 상 호

1979년 경북대학교 전자공학 (공학사)

1981년 영남대학교 전자공학과 (공학석사)

1989년 Syracuse University ECE (MS)

1995년 State University of New York at Buffalo, ECE (Ph.D.)

1996년~현재 안동대학교 전자정보산업학부 부교수