논문 00-25-2A-13 한국통신학회논문지 '00-2 Vol.25 No.2A

E1 프레임을 위한 간단한 실시간 무손실 압축 알고리즘

정회원 정 학 진*, 이 상 호**, 백 성 복*, 정 상 현*, 곽 승 훈*

A Simple Realtime Lossless Compression Algorithm for E1 Frame

Hakjin Chong*, Sangho Lee**, Seongbok Baik*, Sanghyeon Jeong*, Seunghoon Kwak*
Regular Members

요 약

본 논문에서, 우리는 E1 스트림을 프레임단위로 압축하는 간단한 실시간 무손실 알고리즘을 제안한다. 제안된 알고리즘을 위해 설계된 인코더에 대해 설명하고 기존의 무손실 알고리즘과의 성능 비교 결과에 대해 설명한다.

제안된 알고리즘은 E1 프레임의 압축가능 특성을 이용하고 있으며, 프레임내 각 채널의 특성에 대해 독립적으로 압축을 수행할 수 있다. 무손실 데이터 압축 분야에 있어서 거의 표준에 가깝게 인식되고 있는 Ziv-Lempel 알고리즘을 기반으로 한 대부분의 다른 응용들에 비해 알고리즘 자체가 훨씬 간단하고 지연도 거의 없을 뿐 아니라, 압축률에 있어서도 대등한 결과를 보이고 있다.

ABSTRACT

In this paper, we propose a simple realtime lossless algorithm to compress E1 stream for each frame. We also describe the encoder structure for the implementation of the proposed algorithm and the result of the performance in comparison with the two commercial lossless compression applications.

Proposed algorithm utilizes the compressible characteristics of the E1 frame structure and can compress without being affected by the heterogeneity of the channels in a frame. The algorithm structure is far simpler than the other algorithms based on the Ziv-Lemple that is recognized as a virtual standard. The comparison results show us this algorithm has similar capability to the other two non-realtime lossless algorithms.

I. 서 론

E1은 ITU-T가 G.704[1]라는 권고안을 통해 국제 표준으로 규정하고 있는 국간 전송 프레임 구조로서 8KHz의 프레임 속도를 가지며 256비트, 즉 32개의 채널로 구성되어 있다. 이 중에서 2개의 채널은 전송신호 전달용으로 사용되며 나머지 30개의 채널은 일반 가입자들의 음성 및 데이터 전달용으로 사용된다. 이와 같이 각각의 E1 프레임은 이질적인 특성을 갖는 채널들로 구성될 수 있고 고속의 실시간성이 유지되어야 한다는 제약사항을 가지고 있다.

압축 기술은 크게 두 가지 방식으로 구분된다. 첫째는 무손실 압축 방식으로서 원본 데이터의 완전한 복구를 보장하는 기술이고, 둘째는 손실 압축 방식으로서 데이터 복구 시 어느 정도의 왜곡이나 손실이 발생하지만 무손실 방식에 비해 높은 압축률을 보이는 기술이다.

* 한국통신 가입자망연구소 초고속단말연구실, ** 충북대학교 전자계산학과
논문번호 : 99404-1007, 접수일자 : 1999년 10월 7일

본 논문에서 압축 데이터로 취급하고 있는 E1 스트림은 서로 다른 여러 사용자들이 점유하고 있는 채널들이 프레임 단위로 묶여있는 형태이고, 채널을 통해 전달되는 내용도 이질적일 수 있으므로 무손실 압축방식이 적합하다.

무손실 압축방식 중 대표적인 것으로는 허프만 코딩 방식, 산술 코딩(Arithmetic Coding) 방식, 그리고 Ziv-Lempel 알고리즘 등을 들 수 있다. 허프만 코딩이나 산술 코딩 방식은 자료에 대한 통계적 특성, 즉 자료내의 각 심벌들에 대한 출현 확률의 상이성을 토대로 압축을 수행한다. Ziv-Lempel 알고리즘은 사전(Dictionary)식 압축방법이라고 할 수 있으며 특정 귀절(특정 길이를 갖는 심벌들의 연속)이 반복될 경우 나중 나오는 귀절을 첫 귀절에 대한 포인터로 대치하는 방식이다.

그 외에도 특정 분야에 적합한 몇 가지 무손실 압축 방식들이 있는데, 연속되는 공백(Null) 문자를 공백문자임을 나타내는 하나의 신호와 연속된 공백 문자의 총 개수의 쌍으로 표현하는 공백압축(Null Compression)방식, 매 8바이트 마다 공백문자의 위치를 표시한 한 개의 위치 정보 바이트를 추가하고 그 대신 공백문자를 제외시키는 비트맵(Bit Map)방식, 그리고 팩스나 이미지 데이터에 많이 적용되는 방식으로 같은 값을 가지는 연속되는 요소(예, 픽셀)를 요소에 대한 대표 값과 연속된 길이 값으로 대치하는 연속길이(Run Length)방식 등을 들 수 있다.

Ⅱ. 알고리즘 설계

1. 기존 알고리즘과 문제점

E1 스트림을 프레임단위로 압축하기위해 기존의 압축 방법들을 적용하는 데는 몇 가지 문제점이 있다. 먼저, 음성관련 압축 방법들은 압축률은 높지만 채널단위로 압축이 수행되며, 각 채널에 대해 손실이 허용되는 알고리즘을 사용하고 있으므로 적용이 불가능하다. 또한 Ziv-Lempel계열의 무손실 데이터 압축 알고리즘들은 기본 구조상 계산비용이 크고 실시간성을 보장하기가 어렵다는 문제점과 E1 스트림이 가지는 압축가능 특성을 반영할 수 없어 효율성이 저하된다는 문제점을 가지고 있다.

고속의 전송방식인 E1 스트림에 대한 압축을 고려할 때, 지연시간과 계산 비용은 실용성 측면의 성패를 결정짓는 가장 중요한 요인 중 하나라고 할 수 있다. 따라서 본 논문에서는 계산 비용이 큰 Ziv-Lempel 알고리즘을 배제하고, E1 프레임의 각 채널 내에서 시간축으로 인접한 두 채널 값이 변하지 않을 확률이 높다는 점 만을 이용하면서 무손실성과 실시간성을 동시에 성취하기 위해 비트맵(Bit Map) 방식에 기초를 둔 알고리즘을 제안한다.

2. 동작원리

설명을 위해 그림1에 연속되는 3개의 E1 프레임의 최초 8개의 채널에 대해서 예시하였다.

	Ch1	Ch2	Ch3	Ch4	Ch5	Ch6	Ch7	Ch8
Frame2	56	00	FF	34	34	AB	A2	00
Frame1	34	00	FF	30	00	B0	A2	00
Frame0	12	00	5F	34	00	B0	A1	00

그림 1. 원시 E1 프레임 중 8개 채널

그림 2는 인접한 두 프레임을 비교하여 값의 변동이 없는 채널을 00으로 표현하고 값이 변화한 채널은 현재의 채널 값을 넣어서 표현한 것이다. 0번 프레임은 각 프레임을 비교하는 과정에서 초기 기준 값의 역할을 하므로 1번과 2번 프레임에 대한 이전 프레임과의 차분 결과만을 도시하였다.

	Ch1	Ch2	Ch3	Ch4	Ch5	Ch6	Ch7	Ch8
Diff 2	56	00	00	34	34	AB	00	00
Diff 1	34	00	FF	30	00	00	A2	00

그림 2. 인접 프레임 차분 결과

그림 1의 Frame1과 Frame2에서 00인 값을 갖는 채널 인스턴스는 5개였으나, 그림 2와 같이 차분값을 구한 후, 00의 값을 갖는 채널 인스턴스가 8개로 증가한다.

그림 2에서 00인 채널 인스턴스를 비트 0으로 표현하고 00가 아닌 채널 인스턴스를 비트 1로 표현하면, Diff1에 대해서는 10110010(16진수 B2에 해당), Diff2에 대해서는 10011100 (16진수 9C에 해당)이라는 비트열을 각각 얻을 수 있다. 각각의 비트열 내에서 1로 표현된 채널 인스턴스만을 전송한다면 전송량을 현저히 줄일 수 있다.

	Bit Map	Ch1	Ch4	Ch5	Ch6
Block 2	9C	56	34	34	AB

	Bit Map	Ch1	Ch3	Ch4	Ch7
Block 1	B2	34	FF	30	A2

그림 3 압축된 전송블록

그림 3은 Diff1과 Diff2에 대해 압축된 전송블록을 도시한 것으로서 비트열 내에서 1로 표현된 채널들과 그 채널들의 위치를 알려주는 비트열로 구성되어 있다. 비트열을 저장하는 영역을 Bit Map이라고 표현하였다. Bit Map부분은 원래의 채널내용이 아닌 압축을 위한 추가영역이지만 이 추가영역을 포함시키고도, Diff1는 Block1로 변환되면서 3바이트가 축소되었고, Diff2는 Block2로 변환되면서 3바이트가 축소됨을 알 수 있다.

3. 알고리즘

압축 알고리즘은 다음과 같이 간단히 기술할 수 있으며 복원 알고리즘은 같은 원리로 작동하므로 따로 기술하지 않는다. 알고리즘에 사용된 주요 변수는 다음과 같다.

buffer[1..32]: 현재 처리중인 프레임을 저장, 바이트 단위

previous[1..32]: 이전에 입력된 프레임을 저장, 바이트 단위

diff[1..32] : 현재 프레임과 이전 프레임의 차분값을 저장

output[1..32]: 전송될 결과를 저장, 바이트 단위

bitmap[1..32]: 압축시 버려진 채널의 위치를 저장, 비트 단위

diff_null_count: 차분값이 0인 채널의 개수를 저장

```
Initialize;
while there are more frames do begin
    GetAFrame;
    ReadyToCompress;
    if Predict=TRUE then Compress;
    else Pack
    Transmit;
end;

procedure Predict
begin
    for i:=1 to 32 do begin
        diff[i]:=previous[i] - buffer[i];
        if diff[i] = NULL then begin
        diff_null_count:=diff_null_count + 1;
        bitmap[i] :=binary(0);
        end;
    end;
    if diff_null_count > 4 then return TRUE;
    else return FALSE
end;

procedure Compress;
begin
    for i:=1 to 4 do
        output[i]:=bitmap[(i*8-7)..(i*8)];
    k:=0;
    for i:=1 to 32 do
        if diff[i]NULL then begin
          k:=k+1;
          output[k]:=buffer[i]
        end
end;

procedure Pack;
begin
    for i:=1 to 32 do
        output[i]:=buffer[i]
end;
```

4. 계산비용 비교

비교 대상으로 선택한 LZW와 LZSS알고리즘은 4장의 실험에서 사용된 상용 압축프로그램인 유닉스용 Compress 프로그램과 마이크로소프트 윈도우용 ZipCentral 프로그램이 각각 채택하고 있는 알고리즘이다. 본 논문에서는 계산비용의 비교를 위해 Unit이라는 단위를 사용한다. Unit은 알고리즘 수준에서 구분 가능한 최소한의 스텝을 의미하며, 비교, 덧셈, 뺄셈, 대입 연산등이 이에 해당된다.

1) LZW 알고리즘

LZW 알고리즘은 1978년 Ziv-Lempel 이 발표한 LZ78[3]을 근간으로 1984년 Terry Welch가 고안한 것으로서, 입력스트링의 특정 귀절을 사전(Dictionary)에 저장하고 그 귀절이 반복될 경우 사전의 해당 인덱스로 반복 귀절을 대치하는 방식을 기본으로 하고 있으며 대략적인 압축 알고리즘을 기술하면 다음과 같다.

```
procedure LZW-Compress;
begin
w:=NULL;
while there are more characters do begin
    K:=ReadACharacter;
    if wK exists in the dictionary then begin
        w:=wK;
    else begin
        output the code for w;
```

```
        add wK to the dictionary;
        w:=K;
    end
end;
```

이 알고리즘에서 자원 소비가 가장 많은 부분은 첫번째 if문의 조건식 부분으로서, wK를 사전에서 검색하는 과정에 해당하며, 이 알고리즘의 효율을 높이려는 대다수의 논문들이 이 부분을 효율화 시키는 새로운 방식에 대한 제안을 주제로 하고있다. 일반적으로 사전의 초기 크기는 4096개의 항목으로부터 시작하며 필요에 따라 증가된다. 이진탐색방식(Binary Search)을 사용한다고 가정할 때, 사전의 초기 크기 4096을 기준으로 Log2(4096), 즉 12Unit이 바이트당 사전검색에 소요되는 최소 계산비용으로 계산된다.

2) LZSS 알고리즘

LZSS알고리즘은 1977년 Ziv-Lempel이 발표한 LZ77[2]의 출력 형식을 보완한 알고리즘으로서 1982년에 James Storer와 Thomas Szymanski에 의해 고안되었다. LZSS의 대략적인 알고리즘을 아래에 기술하였으며, 알고리즘에 사용된 주요 변수는 다음과 같다.

LookAheadBuffer: 현재 압축을 수행하고 있는 위치로부터 입력 스트림의 맨 끝까지의 범위
Window : Matching 작업에 사용된 문자열로서 입력 스트림의 맨 처음부터 LookAheadBuffer 바로 전까지의 범위 안에 존재
Position : Matching 작업에 사용된 Window의 시작 위치
Length : Matching 작업에 사용된 Window의 길이
MINIMUM_MATCH_LENGTH: 후보 Window들 중 길이가 가장 작은 것을 선택하기위해 사용되는 길이 임시 저장 변수

```
procedure LZSS-Compress;
begin
while LookAheadBuffer not empty begin
    get a pointer (Position, Length ) to the longest match
      in the Window for the LookAheadBuffer;
    If Length>MINIMUM_MATCH_LENGTH begin
        output (Position, Length);
        move the current encoding position Length
          characters forward;
    end
    else begin
        output the first character of the LookAheadBuffer;
        move the current encoding position 1 characters
          forward;
    end
end
end;
```

이 알고리즘은 LZW에 비해 메모리 자원의 소비는 적으나 계산비용이 많다. 특히 임의의 시간에 LookAheadBuffer에 대한 최적의 Window를 결정하기 위해 후보 Window들을 만드는 과정에 처리시간의 대부분이 소요된다. 일반적으로 Window의 크기는 4KByte에서 64KByte사이에서 동적으로 결정된다. 계산비용은 LZW에 비해 4배 이상 소요되는 것으로 알려져 있다[9]. 따라서, LZSS의 바이트당 최소계산비용은 48Unit으로 볼 수 있다.

1) 제안된 알고리즘

II장 3절에 기술된 알고리즘의 핵심부분은 previous와 buffer의 차분 값을 구하는 루틴과 그 결과가 공백인지를 조사하는 루틴이다. 모든 바이트는 이 두개의 루틴을 반드시 거쳐야 하므로 제안된 알고리즘의 최소 계산비용은 2Unit으로 볼 수 있다. LZW나 LZSS에 대해 수많은 개선 방안들이 제시되어 왔고 그 중 일부는 상용 제품에 적용되어 있으나 여기에서는 기본 알고리즘 자체가 가지는 계산 비용을 기초 대상으로 상정하였다. 수행 시간의 측면에서 상기한 세 가지 알고리즘을 비교해 보면, 제안된 알고리즘이 바이트당 2Unit의 계산비용이 요구되는 반면, LZW의 경우 최소 12Unit의 계산비용이 요구되고 LZSS는 48Unit이 요구된다. 따라서, 본 알고리즘은 LZW에 비해 6배, LZSS에 비해 24배의 계산비용 효율성을 갖는다.

Ⅲ. 구 현

1. 압축 블록

그림 4는 압축시킨 프레임을 저장할 블록의 구성을 도시한 것이다. 1번 비트에서 32번 비트까지의 영역에 해당하는 4개의 필드는 2.1절의 비트맵(Bitmap)배열을 저장하는 필드이다. 임의의 값으로 지정된 채널을 표현할 경우 해당 비트에 1 값을 설정하고 공백(Null)인 경우 0 값을 설정한다.

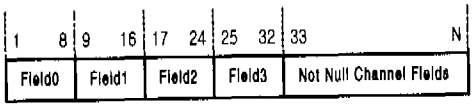


그림 4. 압축 블록의 구조

Not Null Channels 필드는 비트맵(Bitmap)배열에서 공백(Null)이 아닌 채널을 저장하는 필드이다. 비트번호 N은 공백이 아닌 채널의 개수에 따라 본 필드의 길이가 달라진다는 것을 나타낸다. 압축이 적용되지 않은 프레임에 대한 블록은 0번 필드에서 31필드의 내용을 현재 처리중인 프레임의 32개 채널의 내용으로 채운 구조이다.

실제 구현에 있어서는 전송된 블록이 압축 블록인지 비압축 블록인지를 구분하는 방법으로 두 가지의 방법을 사용할 수 있는데 본 논문에서 성능평가를 위해 사용한 방법은 2.3절의 플래그 비트 방법이다.

2. 플래그 비트 방법

플래그 비트 방법은 각 블록 앞에 1비트의 플래그 비트를 두어 해당 블록이 압축된 것인지 압축되지 않은 것인지를 표시한다. 플래그 비트의 존재는 정렬(Alignment)문제를 내포하고 있어서 구현상 불합리면이 있는데, 이를 해결하기 위해 8개의 블록을 한 개의 다중 블록으로 묶어 전송하는 방법을 사용할 수 있다. 그림 5에 다중블록 구조가 도시되어 있다. 8개의 연속하는 블록에서 Flag Bit부분을 따로 모아서 1바이트의 Flag Set로 구성함으로써 정렬 문제를 해결할 수 있다.

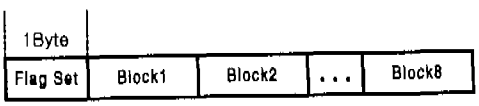


그림 5. 다중 블록

이 경우 전송측에서 8프레임 만큼의 지연이 발생하게 되고, 수신측에서의 처리 지연은 Flag Set을 접수하는데 필요한 1바이트 뿐이다. E1구조에서 1프레임을 전송하는데 소요되는 시간은 1/8000초이므로 8프레임을 버퍼링 시킬 경우 1/1000초의 지연이 발생하게 된다.

3. 길이 지정 방법

각 블록 앞부분에 플래그 비트를 추가하는 대신 해당 블록의 길이를 추가하여 전송하는 방식이다. 임의의 블록에 대해 그 블록의 길이를 보고 압축여부를 판정하는 것이 가능하다. 즉, 현재 전송되어 온 블록의 길이가 32바이트 이상인 경우, 해당 블록은 압축되지 않은 것으로 판정할 수 있다.

플래그 비트 방식과 비교해볼 때, 이 방식은 전송지연을 더 줄일 수 있다는 장점은 있지만 블록 당 7비트의 손실을 감수해야 한다. 그러나, 다른 하부 전송구조(예, AAL2)가 실제 전송에 이용된다고 생각할 때, 대부분의 하부 전송구조가 각 블록의 길이를 저장하는 길이 필드를 가지고 있으므로 이를 이용하면 본 알고리즘 자체가 유발하는 손실은 제거될 수 있다.

4. 인코더 구성

그림 6은 인코더에 대한 구성도이다. 각 기능 블록들의 동작 방법에 대한 설명은 다음과 같다. 입력된 각각의 E1프레임은 먼저 PREDICTOR를 통과하게 된다. PREDICTOR에서는 프레임 내 각 채널에 대해 현재 채널 값에서 이전 채널 값을 빼서 차분 프레임을 구한다. 차분 프레임내에 공백 채널이 5개 이상인 경우 압축대상 프레임으로 판정한다. 공백 채널이 4개 이하인 경우에는 비트맵에 4바이트가 소요되고 공백이 아닌 채널이 28바이트 이상을 차지하게 되어 블록의 총 길이가 32바이트 이상이 되므로 압축을 시도하지 않는다.

PREDICTOR가 압축을 요구하는 경우, BIT MAPPER에서 비트 맵을 작성하고 NULL CHANNEL SUPPRESSOR에서 공백 채널들을 제거한 후 결과를 BLOCK MAKER에서 압축 블록으로 구성한다. PREDICTOR가 압축을 요구하지 않는 경우에는 프레임 내용을 그대로 이용하여 BLOCK MAKER에서 비압축 블록을 구성한다.

현재의 프레임 내용을 다음 번 프레임에 대한 예측값으로 설정한 후 완성된 블록을 상대측으로 전송함으로써 한 프레임에 대한 인코더의 동작이 종료되며 다음 프레임을 받아 들여 압축 과정을 반복한다.

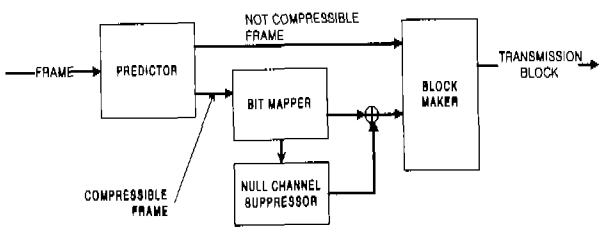


그림 6. 인코더 블록

Ⅳ. 실 험

실험에는 유닉스의 Compress프로그램과 마이크로소프트 윈도우용 Zip Central, 그리고 본 논문에서 구현한 간단한 실시간 무손실 압축 프로그램이 사용되었다.

1. 실험 데이터

실험에 사용된 E1 스트림은 음성 데이터, 즉 8비트 PCM형식으로 부호화(Encoding)된 1분 길이의 실제 통화 내용 30개와 0번과 16번에 해당하는 두 개의 신호(Signal)채널로 구성되어 있다. 0번 채널에 나타나는 프레임 정렬 패턴(Frame Alignment Pattern)과 경보(Alarm)비트(정상(Normal)상태 가정)는 표준[1]에 정의된 대로 이며, 보조(Spare)비트들은 1로, CRC 비트들은 1로 각각 가정하였다.

실제로 트렁크는 일상적으로 60%수준에서 트래픽 양이 유지되도록 권장하고 있고, 이를 E1의 경우에 적용하면 총 30개의 사용자용 채널 중 20개 수준에서 트래픽이 유지 된다고 볼 수 있다. 그러나, 트래픽은 본질적으로 상당히 유동적이므로 다양한 점유율을 고려해야 한다. 이를 위해 E1 프레임 내 30개 유성채널 중 통화중인 채널의 수를 0개, 10개, 20개, 30개로 나누어 실험하였다.

2. 압축률 비교

표1은 15,361,856바이트 크기의 약1분 길이 원시 E1스트림을 활성(Active)채널의 개수별로 세분하여 압축을 실시한 후 압축된 결과 자료의 길이를 나타낸 것이다.

표 1. 압축결과자료 크기 비교 (단위:Byte)

	Zip Central	Compress	Simple rt
0Channel	44696	78933	2460324
10Channel	1740605	2289017	3311764
20Channel	3787211	4769705	4623063
30Channel	4916443	6221865	5474356

그림 7의 감축률은 원시 데이터 중 몇 %가 감축되었는가를 나타내는 지표로서 다음과 같이 정의한다.

감축률 = (원시 데이터의 크기 - 압축 데이터의 크기)/
원시 데이터의 크기 X 100

(단위:%)

그림 7에서 프레임 내 모든 채널이 빈 채로 들어오는 경우, 본 논문의 알고리즘은 프레임 당 4바이트 이상의 오버헤드가 발생하여 다른 알고리즘에 비해 약 20%의 추가 손실이 있음을 알 수 있다. 그러나 빈 채널이 적어 질수록 효율이 증가하며, 모든 채널이 통화중인 경우에는 65% 이상의 감축률을 보이고 있다.

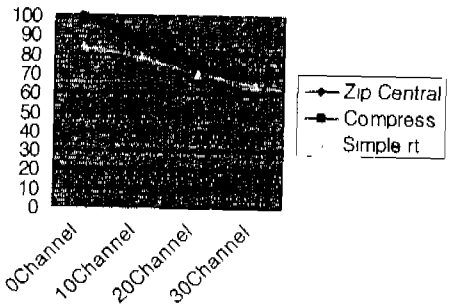


그림 7. 압축결과자료 감축률 비교

3. 평 가

실험에 사용된 채널의 내용은 모두 음성 신호이므로 모뎀신호 등의 데이터가 추가될 경우 결과가 달라 질 수 있다. 모뎀신호가 전송되는 경우 이미 압축이 수행된 자료인 경우가 많아 압축 효율이 저하될 수 있다.

비교 대상 알고리즘들은 실시간성을 고려하지 않고 있으므로 직접적인 비교가 어려운 면이 있다. 현재 실시간 무손실 알고리즘에 대한 논문[4,5,6]은 다수 발표되어 있으나 이미지나 음성 등 특정 영역에 국한된 경우가 대부분이므로 가장 일반적인 무손실 알고리즘 에 상기한 두 가지를 선택하였다.

Ⅴ. 결 론

본 논문에서는 동기화 방법, 전송구조 등 구체적인 전송 방법에 대하여는 언급하지 않고 있으며 다른 전송 방법(예, AAL2)을 이용하여 전송하는 것으로 가정하고 있다. 그리고, 무손실 압축분야에서 주류를 이루고 있는 Ziv-Lemple 알고리즘 대신에 공백압축기법을 이용하였기 때문에 구조가 단순하고 실시간성의 구현이 용이하다.

본 알고리즘은 E1 구조를 프레임 단위로 압축하므로 구성 채널 각각의 특성에 대해 독립적이다. 즉, 각각의 채널에 적재되는 내용에 관계없이 압축

을 수행할 수 있고 사용자 채널들을 집합형으로 고속 전송하는 구조라면 E1 구조 이외의 어떤 구조에도 적용 가능하다.

참 고 문 헌

[1] Synchronous Frame Structures Used At 1544, 6312, 2048, 8488 and 44736kbit/s Hierarchical Levels, ITU-T Rec. G.704, pp. 8-11, July 1995.

[2] J. Ziv and A. Lempel, A Universal Algorithm for Sequential Data Compression, IEEE Trans. on Inform. Theory, Vol. IT-23, No. 3, pp. 337-343, May 1977.

[3] J. Ziv and A. Lempel, Compression of Individual Sequences Via Variable-rate Coding, IEEE Trans. on Inform. Theory, Vol. 24, pp. 530-536, Sept. 1978.

[4] Venbrux, Jack, Pen-Shu Yeh and Muye N. Liu, A VLSI Chip Set for High-Speed Lossless Data Compression, IEEE Trans. on Circuits and Systems for Video Technology, Vol. 2, No. 4, Dec. 1992.

[5] Yeh, Pen-Shu, Robert F. Rice and Warner H. Miller, On the Optimality of a Universal Noiseless Coder, Proc. of the AIAA Computing in Aerospace 9 Conference, San Diego, CA, Oct. 19-21, 1993.

[6] Rice, Robert E, Some practical Universal Noiseless Coding Techniques, JPL Publication 79-22, 1979.

[7] James A. Storer, Thomas G. Szymanski, Data Compression via Textural Substitution, JACM 29, pp.928-951, 1982

[8] Terry A. Welch, A Technique for High Performance Data Compression, IEEE Comp., Vol. 17, No. 6, pp. 8-19, June 1984

[9] M. Burrows and D.J. Wheeler, A Block-sorting Lossless Data Compression Algorithm, SRC Research Report Digital, May 10, 1994

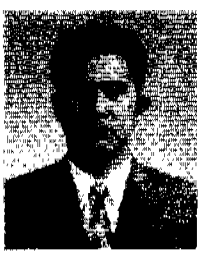
정 확 진(Hakjin Chong) 정회원

1982년:경북대학교 전자공학과 졸업(학사)
1984년: 경북대학교 대학원 전자공학과 졸업(석사)
1999년 충북대학교 대학원 전자계산학과 (박사과정 수료)
1985년~현재: 한국통신 가입자망연구소 실장
<주관심분야> 영상통신, ATM Networking, B-ISDN TE

이 상 호(Sangho Lee) 정회원

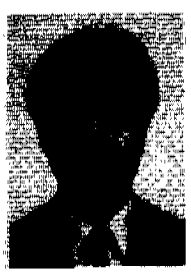
1976년 2월: 숭실대학교 전자계산학과(공학사)
1981년 2월: 숭실대학교 전자계산학과(공학석사)
1989년 2월: 숭실대학교 전자계산학과(공학박사)
1981년 3월~현재: 충북대학교 전자계산학과 교수
<주관심분야> 프로토콜 공학, 시뮬레이션, 정보보호, ATM 네트워크

백 성 복(Seongbok Baik) 정회원



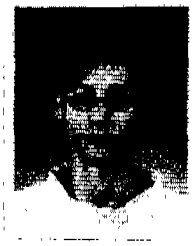
1990년: 강원대학교 전자계산학과 졸업(학사)
1992년: 숭실대학교 대학원 전자계산학과 졸업(석사)
1992년~1995년: 한국개발연구원
1995년~현재: 한국통신 가입자망연구소 초고속단말연구실
<주관심분야> Computer Networking, Internet Computing, IP Over ATM

정 상 현(Sanghyeon Jeong) 정회원



1987년: 충남대학교 계산통계학과 졸업(학사)
1989년: 숭실대학교 대학원 전자계산학과 졸업(석사)
1990년~현재:한국통신 가입자망연구소 초고속단말연구실
<주관심분야> ATM Networking, EDI

곽 승 훈(Seunghoon Kwak) 정회원



1995년 2월: 전북대학교 전자계산학과 졸업(학사)

1995년 2월~현재:한국통신 가입자망연구소 초고속단말연구실

<주관심분야> ATM Networking, B-ISDN TE, 통신 미들웨어