VoIP 통화 품질 개선을 위한 적응 재생 버퍼 제어 기법

Adaptive Playout Buffer Control Method for Improvement of VoIP Speech Quality

강진아, 고성택, 임재윤*
제주디지털콘텐츠연구센터, 제주대학교*

Kang Jin-Ah, Ko Sung-Taek, Lim Jea-Yun*
Jeju DCRC, Jeju Univ.*

요약

실시간 음성 서비스를 지원하는 VoIP(Voice over IP) 시스템에서 음성 품질은 지연, 지터, 손실, 그리고 역전된 패킷 순서에 의해 손상된다. 본 논문에서는 적응 재생 알고리즘에 의해 지터를 보상하고 패킷 손실 보상을 수행하며 패킷 순서를 정렬하는 수신단의 적응 재생 버퍼 제어 기법(Adaptive Playout Buffer Control: APBC)을 제안하였다. 또한 임베디드 VoIP 시스템을 구현하여 구현 시스템에서의 APBC 성능을 측정한 결과, 처리속도는 257μsec로 실시간으로 처리하기에 적합하고 MOS(Mean Opinion Score)에 의한 음성 품질은 고정 재생 지연 알고리즘에 비해 18% 개선되었다.

Abstract

In a VoIP(Voice over IP) system which support the realtime speech service, speech quality is deteriorated by the delay, the jitter, the loss, and the reversed packet order. In this thesis, APBC for receiver site is proposed, which compensate the jitter by the adaptive playout algorithm and conceal the packet loss, and align the packet order. Also, a VoIP application system is implemented, and the performance of APBC is verified on the implemented system by measuring the processing speed and the speech quality. From the result, processing speed is 257μsec, which is fast enough to deal with packet being received in realtime. Also, the speech quality by MOS(Mean Opinion Score) is improved as 18 percent compared with algorithm of fixed playout delay.

I. 서 론

패킷 교환 방식이 사용되는 IP 망에서는 지연, 지터(jitter), 패킷 손실, 패킷 순서 뒤바뀜 현상 등이 발생하고 이는 VoIP(Voice over IP) 통화 품질을 저해한다.

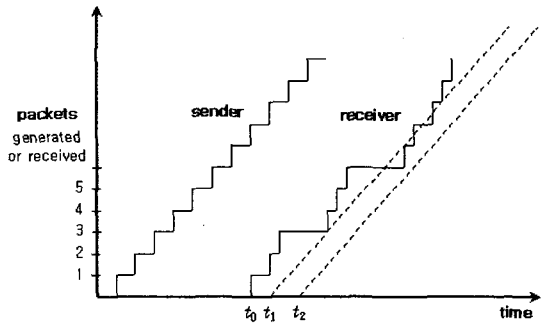
VoIP 시스템에서 전송 지연을 고려하여 지터를 보상하는 방법으로는 고정 재생 지연 방식과 적응 재생 지연 방식이 있다. 고정 재생 지연 방식은 음성 세션이 시작될 시에 모든 패킷에 대한 재생 지연 크기를 고정시키는 것이다.[1] 적응 재생 지연 방식은 음성 세션 동안에 재생 지연을 적절하게 조절하는 것으로, 최근 몇 년 동안 이러한 방식을 이용하는 적응 재생 알고리즘들이 논문을 통해 제안되고 있다.

지금까지의 VoIP 응용 시스템들은 대부분 고정 크기를 갖는 지터 버퍼를 이용함으로써 고정 재생 지연 방식을 사용하고 있다. 그러나 고정 재생 지연 방식은 지터를 효율적으로 처리하지 못하여 만족할만한 통화 품질을 제공하기 어렵다.

본 논문에서는 적응 재생 알고리즘을 도입하여 지터를 보상하는 수신단의 적응 재생 버퍼 제어 기법(APBC : adaptive playout buffer control)을 제안한다. 이를 위해 적응 재생 버퍼의 구조와 제어 기법을 설계하고 이에 기반하여 α-adaptive 알고리즘을 구현한다. 또한 APBC 기법이 패킷 손실 은닉 기법을 수행하고 RTP(Real Time Protocol) 헤더 정보를 이용하여 패킷 순서를 정렬하도록 구현한다. 이후에는 실제 고성능 임베디드 VoIP 시스템을 구현하고 APBC 기법을 적용하여 제안 기법의 효용성을 검증한다.

II. 적응 재생 알고리즘

1. 적응 재생 지연 조절



▶▶ 그림 1. Generation and reconstruction of packetized audio with fixed playout delay

그림 1은 규칙적인 패킷의 생성과 이를 수신단에서 고정 재생 지연 방식으로 재생하는 메커니즘을 나타내고 있다. 지터가 제거되어 규칙적인 시간 간격으로 손실 없는 재생을 하기 위해서는 t_2 시간에 재생을 하면 되는데, 이는 어느 정도의 패킷 손실이 허용됨에도 불구하고 모든 패킷들을 과도하게 지연시킨다. 대신에 t_1 시간에 재생을 시작하면 종단간 지연은 줄지만 패킷 손실이 발생된다. 따라서 보다 작은 t_1을 선택하면서 패킷 손실 허용치를 만족시킬 수 있는 재생 알고리즘이 필요하다. 그런데 음성 세션 시작 시에 재생 지연을 설정하는 고정 재생 지연 방식은 만족스러운 음질을 얻지 못하므로 이러한 재생 알고리즘들은 음성 세션 동안에 비교적 패킷 손실을 낮게 유지하면서 적응적으로 재생 지연(t_1)을 조절하도록 요구된다. 이러한 적응적 재생 지연의 조절은 음성 세션 내의 묵음 주기에서 수행된다.[2]

2. 적응 재생 알고리즘

2.1 자동 회귀 추정 기반 알고리즘

자동 회귀를 이용한 평균 망 지연 d_i는 식 (1)와 같다.

$$d_i = \alpha d_{i-1} + (1-\alpha) n_i \quad (1)$$

여기에서 n_i는 현재 수신 패킷에 대한 망 지연이고 α는 알고리즘의 수렴율을 조절하는 가중 인자이다. d_i에 대한 편차는 식 (2)과 같다.

$$v_i = \alpha v_{i-1} + (1-\alpha) | d_i - n_i | \quad (2)$$

식 (1)와 식 (2)에 의한 평균 망 지연과 이에 대한 편차는 다음 패킷에 대한 종단간 지연을 예측하는데 사용되며 이는 식 (3)과 같다.

$$ted = d_i + \beta v_i \quad (3)$$

여기에서 β는 보다 안정적으로 종단간 지연의 예측치가 실제치보다 커지도록 하는 가중 인자로 β 값이 클수록 늦게 도착함에 따른 패킷 손실량은 작아진다. 논문들에서 β값은 4로 제시되었다.

2.2 α-adaptive 알고리즘

식 (1)에서 α의 선택은 d_i가 얼마나 빨리 망 지연에 대한 변화를 쫓을 것인가를 결정하며 α값의 작은 변화는 지연-손실 상반 관계에 확연한 영향을 미친다. 일부의 패킷들이 도착하고 난 뒤에 하나의 음성 구간에 대한 재생 지연을 최적의 값으로 결정할 수 있도록 α값을 적응적으로 조절하고자 한 것이 α-adaptive 알고리즘이며 그림 2와 같이 수행된다.

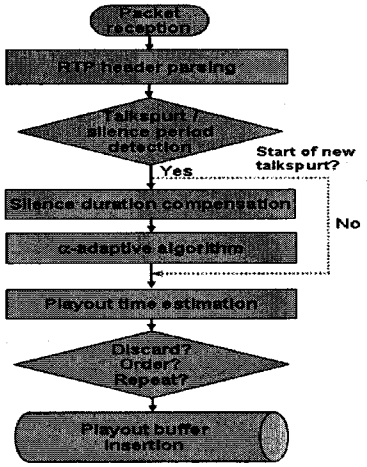
1. α=0.998002, *increment*=0.0001, $\alpha_2=\alpha-increment$
2. Calculate *ted* based on α and ted_2 based on α_2 using (1), (2) and (3). Use ted_2 for actual playout.
3. loss1 = Loss(L, α), loss2 = Loss(L, α_2)
   /* Loss(X, Y) calculates the loss based on $\alpha = Y$ in the previous X talkspurts */
4. if loss2 < loss1
   if $\alpha_2 < \alpha$ then $\alpha_2=\alpha-increment$
   if $\alpha_2 > \alpha$ then $\alpha_2=\alpha+increment$
5. if loss2 > loss1
   if $\alpha_2 < \alpha$ then $\alpha_2=\alpha-increment$
   if $\alpha_2 > \alpha$ then $\alpha_2=\alpha+increment$
6. Repeat steps 3 to 5 every L^{th} talkspurt.

▶▶ 그림 2. Process of α -adaptive algorithm

III. 적응 재생 버퍼 제어 기법(APBC)의 설계 및 구현

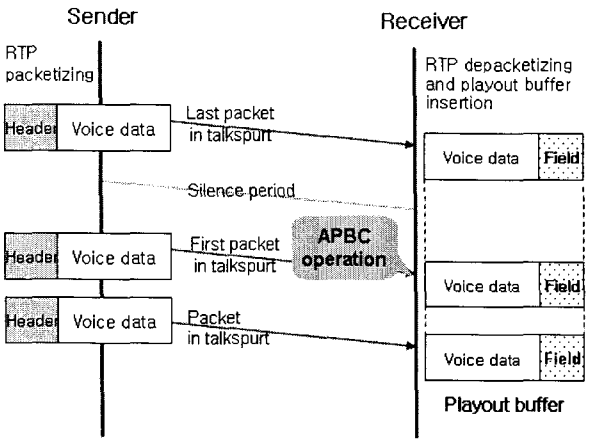
1. APBC 수행 절차

APBC는 α-adaptive 알고리즘을 실제 VoIP 시스템에 적용하고 이 외에도 부가적인 통화 품질 개선 알고리즘을 수행하도록 본 연구에서 설계한 재생 버퍼 구조와 제어 과정을 말한다. APBC는 음성 세션의 시작 단계에서 재생 버퍼를 생성한 후, 매 음성패킷 수신에 대해 RTP 헤더를 분석하고 음성 구간의 시작을 판별하며 α-adaptive 알고리즘을 수행한다. 이에 따라 버퍼링 지연을 프레임 수로 구하여 재생 버퍼의 적절한 위치에 음성 프레임 데이터를 기록한다. 이 외에도 RTP



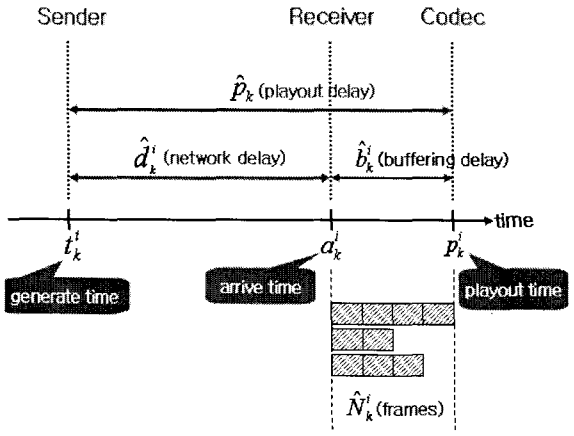
▶▶ 그림 3. APBC flowchart

헤더의 타임스탬프와 순서 번호를 비교하여 패킷을 순차적으로 정렬하고, 망에 의한 패킷 손실이 발생한 경우에는 이전 패킷을 반복 재생한다. 아래의 그림 3은 이러한 APBC 동작을 순서도로 나타낸 것이고, 그림 4는 실제 음성 통신 상의 APBC 동작 시점을 나타낸 것이다.



▶▶ 그림 4. Sender/receiver system model for APBC on voice communication

2. APBC 재생 지연 조절 매커니즘



▶▶ 그림 5. Playout delay adjustment in APBC

그림 5를 참조하여, α-adaptive 알고리즘에 의해 k-번째 음성 구간에 대한 재생 지연을 추정한 후의 i-번째 패킷의 버퍼링 지연은 식 (4)와 같이 계산된다.

$$\hat{b}_k^i = \hat{p}_k - \hat{d}_k^i \tag{4}$$

즉, 각 수신 패킷에 대한 망 지연이 변하기 때문에 버퍼링 지연을 조절하여 하나의 음성 구간에 대한 재생 지연을 일정하게 유지시키도록 한다. 버퍼링 지연은 그림 5에서와 같이 코덱이 한번에 처리하는 단위 프레임의 개수(N_k^i)로 변환시킨다. 즉, a_k^i 시점에 도달한 패킷이 재생 버퍼에 인가된 후 코덱에서 재생되기까지의 버퍼링 지연은 (버퍼 대기 프레임 수 × 코덱 지연)이 된다. 따라서 버퍼 대기 프레임 수를 조절하여 버퍼링 지연을 가변한다. 버퍼 대기 프레임 수는 식 (5)에서 N_k^i와 같다.

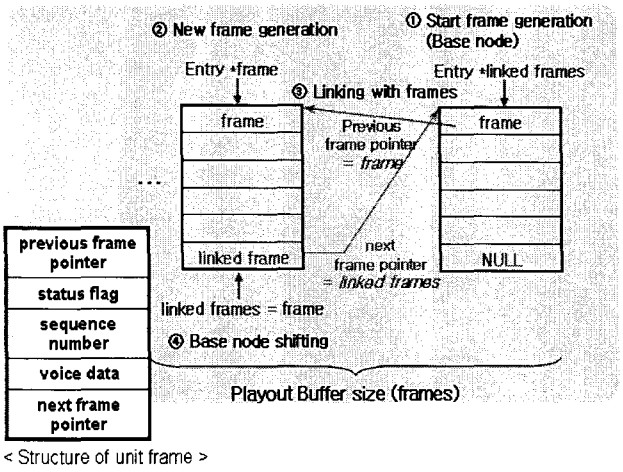
$$\frac{\hat{b}_k^i}{\hat{d}_{codec}} = \frac{p_k^i - a_k^i}{\hat{d}_{codec}} = N_k^i + \hat{r}_k^i \tag{5}$$

음성 구간의 시작에서 재생 지연을 프레임 수로 변환한 이후부터는 수신 패킷들에 대한 재생 시간을 프레임 간격으로 추정하며 이는 식 (6)과 같다.

$$p_k^i - p_k^{i-1} = t_k^i - t_k^{i-1} \tag{6}$$

3. APBC 재생 버퍼

3.1 재생 버퍼 생성 및 쓰기 동작



< Structure of unit frame >

▶▶ 그림 6. Process of playout buffer generation

그림 6에서 좌측 하단의 구조체는 재생 버퍼를 구성하는 단위 프레임 구조체를 나타낸다. 이는 하나의 수신 패킷에 대한 정보 멤버(member)와 음성 데이터 멤버, 그리고 단위 프레임 구조체간의 연결 주소 멤버로 이루어진다. 정보 멤버로는 음성 프레임, 망에서 손실된 프레임, 뒤바뀐 순서로 전송된 프레임, 그리고 묵음 프레임을 구분하여 나타내고 상태 표시자('status flag')와 RTP 헤더를 분석하여 얻어지는 순서 번호('sequence number')가 있다. 음성 데이터 멤버는 음성 코덱이 한번에 처리할 수 있는 단위 프레임 크기의 음성 데이터

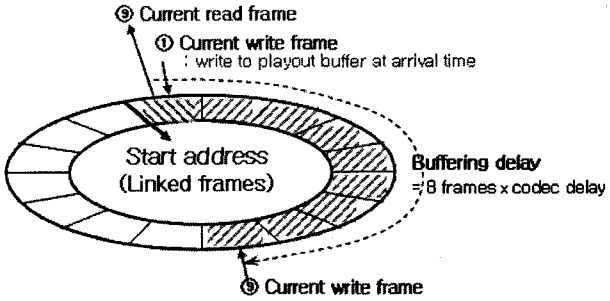
('voice data')를 저장하며, 연결 주소 멤버로는 이전 프레임 포인터('previous frame pointer')와 다음 프레임 포인터('next frame pointer')가 있다. 그림 6은 이러한 단위 구조체를 이용하여 재생 버퍼를 생성하는 과정을 나타낸다.그림 6과 같은 구조를 갖는 재생 버퍼에 수신된 음성 패킷의 프레임을 기록하는 과정은 다음의 의사 코드(pseudo code)로 나타내었다.

```
/* check status flag */
packet_interval=|current_sequence_number - previous_sequence_number|;
decision_wrap_around(packet_interval);
check_status_flag();
   → status_flag = PACKET_NORMAL;
   → status_flag = PACKET_AFTER_LOSS;
   → status_flag = PACKET_REVERSED;

/* decide whether this packet to be discarded or not */
if( p_k^i > a_k^i )
    status_flag = DISCARD_PACKET;
    return;

/* write packet data to appropriate location of frame within playout buffer */
switch ( status_flag )
  case PACKET_NORMAL :
       /* write to current frame */
      current_write_frame->flag = PACKET_NORMAL;
      current_write_frame->sequence_number = current_sequence_number;
      memcpy( current_write_frame->voice_data, current_voice_data, 24 );
      current_write_frame = current_write_frame->next;
  case PACKET_REVERSED :
     /* search the appropriate location of frame */
      frame = current_write_frame->previous_frame;
      for( i = 0; i < packet_interval+1; i++ )
           frame = frame->previous_frame;
           if ( (current_sequence_number >= frame->sequence_number) &&
              (frame->status_flag == PACKET_AFTER_LOSS) )
                /* write to current frame */
                  frame->status_flag = PACKET_NORMAL;
                  memcpy( frame->voice_data, current_voice_data, 24 );
  case PACKET_AFTER_LOSS :
      /* check status_flag of lost frames */
      for( i = 0; i < number_of_lost_frames; i++ )
           current_write_frame->flag = PACKET_LOSS;
           current_write_frame->sequence_number =
                              previous_sequence_number + i + 1;
           current_write_frame->data = 0x00;
           current_write_frame = current_write_frame->next;
      /* write to current frame */
      current_write_frame->flag = PACKET_NORMAL;
      current_write_frame->sequence_number = current_sequence_number;
      memcpy( current_write_frame->voice_data, current_voice_data, 24 );
      current_write_frame = current_write_frame->next;
```

▶▶ 그림 7. Pseudo code of writing on playout buffer

3.2 읽기 동작

최종적으로 수신단의 음성 코덱은 APBC 읽기 동작을 통해 재생 버퍼 상에 프레임 단위로 기록된 음성 데이터를 가져와서 아날로그 음성 신호로 복원한다. APBC 읽기 동작을 수행하는 프로세서는 APBC 기록 과정과는 독립적으로 수행되며, 음성 세션이 시작되어 첫 번째 패킷이 수신되면 그 첫 번째 패킷의 버퍼링 지연만큼 대기한 후에 구동이 시작된다. APBC 읽기 동작은 단순히 재생 버퍼로부터 순차적으로 프레임 데이터를 읽어 와서 재생하면 되는 것으로, 단 상태 표시자가 'PACKET LOSS'를 나타내는 경우에는 현재 읽으려는 프레임 이전으로 가장 근접하게 저장되어 있는 프레임 데이터를 검색하여 반복 재생한다. 그림 8은 이러한 APBC 읽기 과정을 나타낸 것으로 APBC 기록 후에 버퍼링 지연만큼 프레임 데이터가 쌓이면 읽기 동작을 수행함을 나타낸다.



▶▶ 그림 8. APBC read/write operation

IV. VoIP 시스템 구현 및 APBC 적용

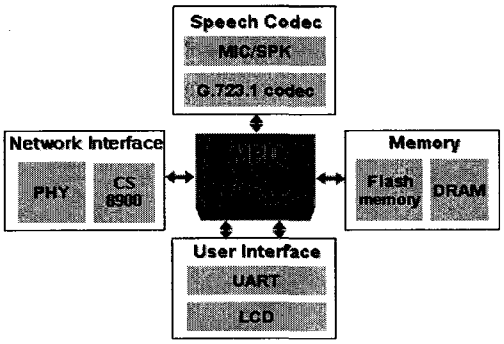
1. 임베디드 VoIP 시스템 구현

임베디드 시스템의 주요 성능을 결정하는 마이크로프로세서는 핸드헬드(handheld) 컴퓨팅 응용에 적합한 고성능 저전력 프로세서인 XScale 계열을 선택하였다. 표 1은 시스템 설계에 대한 주요 파라미터를 나타낸 것이다.

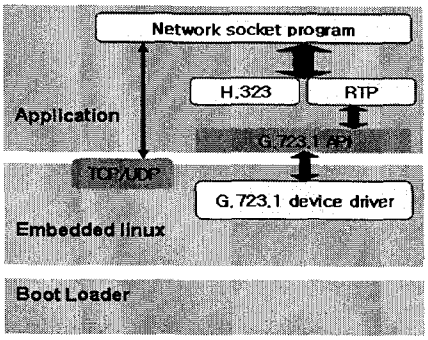
[표 1] Design parameters of embedded VoIP system

System Parameters	Format	Performance
Microprocessor	Intel XScaleTM (ARM, 32bit)	400Mhz
Network Interface	IEEE 802.3 Ethernet	10BASE-T
Speech Codec	G.723.1 (OakDSPCoreTM)	50MIPS fixed point DSP core
Memory	SDRAM	64Mbyte
	Flash memory	16Mbyte
Operating System	embedded Linux	kernel version-2.4.18
VoIP Signalling Protocol	H.323	version 2

그림 9와 그림 10은 각각 설계한 임베디드 VoIP 시스템의 하드웨어 구성도와 소프트웨어 구성도를 나타낸 것이다.



▶▶ 그림 9. Hardware structure of implemented system



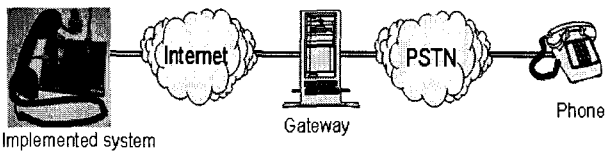
▶▶ 그림 10. Software structure of implemented system

2. APBC 성능 시험

```
(1) WHILE(there is a packet in a trace file)
(2) Fetch a packet;
(3) First checkpoint;
(4) Packet processing of the algorithm;
(5) Microsecond checkpoint;
```

▶▶ 그림 11. Measurement of APBC processing time

그림 11은 구현 시스템에서의 APBC 수행 속도를 측정한 방법을 나타낸 것이며 수행 속도는 257μsec로 측정되었다. 이는 음성 코덱의 단위 음성 신호 처리 시간보다 월등히 작은 값으로, 본 논문에서와 같이 G.723.1 음성 코덱을 사용하는 경우에 코덱이 처리하는 시간에 비한 알고리즘의 수행속도는 0.8%에 해당한다. 이는 APBC가 실시간 처리에 적합하게 수행됨을 확인할 수 있다.



▶▶ 그림 12. Measurement of the real-time voice quality

그림 12는 APBC를 적용한 구현 시스템의 통화 품질 시험 구성도를 나타낸 것이다. 통화 품질 측정은 트래픽이 많은 IP 망 환경에 둔 상태에서 원격지의 VoIP 게이트웨이를 통하여 일반 전화와 통화 하는 경우에, 구현 시스템을 사용하는 사람 측에서의 MOS(Mean Opinion Score)를 측정하였다. 총 10인에 대하여 동일한 대화 내용으로 각 30초간 통화를 수행하였다. 표 2는 구현 시스템에 고정 재생 지연 방식을 사용하는 Open H.323을 적용하였을 때와 APBC 기법을 적용하였을 때의 통화 품질을 비교한 것으로 MOS 평균치를 나타내었다.

[표 2] The Performance comparison of APBC with Open H.323

	Open H.323	APBC
Speech Quality [MOS]	2.7	3.2

표 2에서 고정 재생 지연 방식을 사용하는 Open H.323을 적용하였을 때의 통화 품질은 2.7이고 제안 기법을 적용하였을 때의 통화 품질은 3.2로 측정이 되어 18%의 개선 효과를 가져왔다.

V. 결 론

본 논문은 VoIP 시스템의 통화 품질 개선을 위해 적응 재생 알고리즘을 도입하여 지터를 보상하고, 패킷 손실 은닉 및 패킷 순서 정렬을 수행하는 수신단 적응 재생 버퍼 제어 기법을 제안하였다. 또한 임베디드 VoIP 시스템을 구현하고 제안 기법을 적용하여 알고리즘의 수행 속도와 통화 품질 개선 효과를 검증하였다. 그 결과, 제안 기법의 알고리즘 수행 속도는 257μsec로 측정되어 실시간으로 수신되는 패킷들을 처리하기에 적합하다. 또한 통화 품질 만족도를 MOS로 측정한 결과, 고정 재생 지연 방식을 사용하는 알고리즘에 비해 18% 개선되었다.

본 논문에서는 VoIP 통화 품질 개선을 위한 수신단 패킷 처리 기법을 제안하고 VoIP 응용 시스템 상에서의 적용 가능성과 통화 품질 개선 효과를 검증함으로써, VoIP 응용 시스템이 보다 안정적인 통화 품질을 제공하기 위한 하나의 방안으로 제안 기법이 적합할 것으로 사료된다.

▌참 고 문 헌▐

[1] 유승화, 인터넷 전화, 전자신문사, pp.222-236, 2002.
[2] Kansal A. and Karandikar A., "Adaptive Delay Estimation for Low Jitter Audio over Internet", IEEE Global Telecommunications Conference, Vol.4, pp.2591-2595, 2001.
[3] Pinto J. and Christensen K. J., "An Algorithm for Playout of Packet Voice based on Adaptive Adjustment of Talkspurt Silence Periods", LCN, pp.224-231, 1999.
[4] Schulzrinne H., "Voice Communication Across the Internet: A Network Voice Terminal", Technical Report University of Massachusetts, p.34, 1992.
[5] http://www.ietf.org/rfc/rfc1889.txt