오디오 품질을 개선하기 위한 6개의 Jitter Control 알고리즘의 성능 분석

Performance Evaluation of Six Jitter Control Algorithms for Improving Audio Quality

나승구, 유홍준, 안종석 (동국대학교 컴퓨터공학과)
이태진 (전자통신연구원 방송미디어 연구부)

요약

음성 데이터의 패킷 지터(jitter)가 심할수록 오디오 플레이어가 오디오 데이터를 자연스럽게 재생하지 못하기 때문에 사용자는 원래의 음성을 거의 알아들을 수 없게 된다. 이 문제점을 해결하기 위하여 오디오 수신자는 전송받은 오디오 데이터를 바로 재생하지 않고 재생시간을 지연시키는 방법을 사용한다. 본 연구자의 조사에 의하면 이러한 재생시간을 지연하는 대표적인 지터 컨트롤 알고리즘으로 6가지 방식이 제안되고 있다. 그 중 세 가지는 NeVot, Vat, Open H.323 프로그램 등에 구현되어 실제로 사용되고 있다. 본 논문에서는 이들 6가지의 모델의 지터 컨트롤 알고리즘의 특성을 알아보고 어느 알고리즘이 효율적인지 알아보기 위해 현재 인터넷의 성능을 파악하고 이를 기초로 제안된 6가지 알고리즘 중 어느것이 가장 효율적인가를 파악하여 오디오의 음질을 개선하기 위한 방법을 제시하고자 한다.

1. 서론

인터넷은 송신자로부터 수신자에게 가능한 빨리 데이터를 전송하도록 최선의 서비스(best-effort service)를 제공하는 네트워크이므로 데이터가 수신자에게 도달하는 것을 확실하게 보장해 주지는 못한다. 그러므로 각 데이터가 네트워크에서 혹은 종단에서 지연이 생길 수도 있고 데이터의 손실이 발생할 수도 있다.

특히 패킷 지연(delay)으로 인한 지터나 패킷 손실은 인터넷 폰 또는 음성회의 시스템과 같은 실시간 멀티미디어 응용프로그램의 품질에 상당한 영향을 미치게 된다. 다행히도 이들의 영향을 최소화하기 위한 노력으로 패킷 지터에 대해서는 패킷을 길게 지연시킨다든지 패킷 손실에 대해서는 과도하게 손실이 발생하지 않도록 하는 등의 연구[1][2]가 진행되고 있다.

오디오 응용프로그램에서 송신자는 대화(talkspurt)와 묵음(silence)의 단계를 번갈아 가면서 오디오 신호를 발생시킨다. 대부분의 오디오 응용 프로그램에서는 대화단계에만 패킷을 발생시킨다. 예를 들면 대화단계에서 8000Hz로 샘플링(sampling) 한다면 매 20ms마다 160 바이트 만큼 패킷을 생성하게 된다.

생성된 패킷은 중간의 여러 라우터들을 거치면서 전파(propagation)지연, 프로세싱(processing) 지연, 큐잉(queuing) 지연을 겪게 되고 종단에서도 프로세싱 지연을 겪으면서 수신자에게 도착한다[3]. 이 때 각자의 패킷의 지연 상황이 다르게 되므로 지연 시간의 변화, 즉 지터가 발생하게 된다.

음성 데이터의 패킷지터가 심할수록 오디오 플레이어가 오디오 데이터를 자연스럽게 재생하지 못하기 때문에 결과적으로 사용자는 원래의 음성을 거의 알아들을 수 없게 된다.

이 문제점을 해결하기 위하여 오디오 수신자는 전송받은 오디오 데이터를 바로 재생하지 않고 재생시간을 지연시키는 방법을 사용한다.

본 논문은 다음과 같이 구성된다. 2장에서는 6가지 지터 컨트롤 알고리즘의 특성을 알아보고 3장에서는 인터넷의 성능을 측정하기 위한 실험과 분석을 하며 마지막으로 결론과 향후 연구과제에 대하여 기술한다.

2. 지터 컨트롤 알고리즘

패킷 지터의 문제점을 해결하기 위한 지터 컨트롤 알고리즘으로는 대표적으로 6가지 알고리즘이 있다. 재생시간까지 지연시키는 방법에 따라 두 가지 방식으로 나뉘는데 수신받은 패킷을 매 패킷마다 일정하게 지연시키는 방식을 고정 재생 지연(fixed playout delay) 방식이라고 하며 상황에 따라 동적으로 변화시키는 방식을 적응적 재생 지연(adaptive playout delay) 방식[3]이라고 한다. 본 장에서는 지터 컨트롤 알고리즘의 동작 메커니즘에 대해 설명하고 6가지 각 알고리즘의 특성에 대하여 기술한다.

2.1 지터 컨트롤 메커니즘

그림 1은 송신자가 i번째 패킷을 발생시켰을 때 수신자가 이 패킷을 수신하여 일정한 시간 동안 지연시킨 후 재생할 때까지 사용되는 여러 가지 시간을 표기[3][4]한 것이다.

t_i : 송신자 측 호스트에서 패킷 i가 생성된 시간
a_i : 수신자 측 호스트에서 패킷 i를 받는 시간
p_i : 수신자 측에서 패킷 i가 재생되는데 소요되는 시간
D_{prop} : 송신자로부터 수신자까지 전파지연(propagation delay) 시간
v_i : 송신자로부터 수신자까지 큐잉지연(queuing delay) 시간
b_i : 패킷 i가 수신자 측에 도착된 후에 버퍼에서 재생되기를 기다리는 시간 ($b_i = p_i - a_i$)
d_i : 송신자 측에서 생성된 패킷 i가 수신자 측에서 재생될 때까지 걸리는 시간 ($d_i = p_i - t_i$)
n_i : 네트워크에서 소요된 시간(전파지연시간 + 큐잉지연시간) ($n_i = a_i - t_i$)

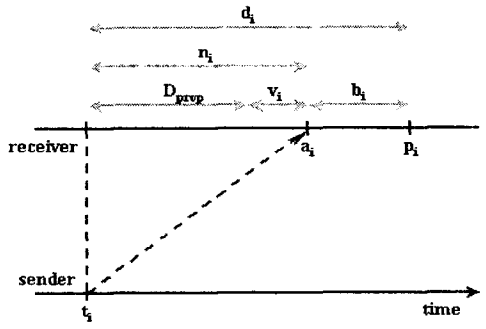

그림 1 패킷 i와 관련된 시간

패킷 i의 재생시간을 결정하려면 그 패킷이 대화단계의 첫번째 패킷인지 아닌지에 따라 두 가지로 나누어 생각해야 한다. 만약 패킷 i가 대화단계의 첫번째 패킷이라면 재생시간 p_i는 다음과 같이 계산된다.

$$p_i = t_i + \underline{d_i} + 4 * \underline{v_i}$$

여기서 $\underline{d_i}$와 $\underline{v_i}$ 는 대화단계의 종단간 지연에서 평균값과 편차를 의미한다.

그러나 대화단계의 첫번째 패킷이 아니고 대화단계에 있는 패킷들이라면 그 패킷의 재생시간은 대화단계의 첫번째 패킷이 재생되는 시점으로부터 오프셋(offset)으로 계산되어야 한다. 만약 패킷 i가 대화단계의 첫번째 패킷이었다면 그 다음 패킷 j는 대화단계의 패킷에 속하기 때문에 이 패킷의 재생시간은 다음과 같이 계산된다.

$$p_j = p_i + t_j - t_i$$

여기서 $\underline{d_i}$와 $\underline{v_i}$가 대화단계의 첫번째 패킷일 때만 이용된다 하더라도 패킷이 도착할 때마다 매번 계산된다.

2.2 대표적 지터 컨트롤 알고리즘

2.2.1 알고리즘 1

이 알고리즘은 지터 컨트롤 알고리즘 중에서 가장 기초적인 알고리즘에 해당되며 고정 재생 지연 방식에 해당된다. 또한 버퍼 큐를 가지는 다른 알고리즘과는 달리 연결 리스트(linked list)로 구성된 데이터 구조를 사용한다.

연결 리스트 구조에서 oldestFrame은 가장 늦은 타임스탬프를 가진 패킷이고 newestFrame은 가장 빠른 타임스탬프를 가진 패킷이다. 네트워크를 통해 새로운 currentFrame이 들어오면 연결 리스트의 어느 위치에 새로운 패킷이 들어가야 할지를 결정한다. currentFrame 패킷의 타임스탬프와 연결리스트의 각 패킷의 타임스탬프를 비교하여 타임스탬프 순서에 맞게 연결시킨다. 만약 currentFrame의 타임스탬프가 oldestFrame보다 크면 재생시간에 늦게 도착한 패킷이므로 버리게 된다. 새로운 패킷이 도착할 때마다 oldestFrame을 선택하여 복원(decompression)한 후 재생시킨다.

Open h.323 프로그램에서는 이 알고리즘을 사용하고 있으며 기본적으로 50ms의 고정 지연시간을 갖는다. 이 방식은 알고리즘 자체가 매우 간단하기 때문에 구현하기 쉽지만 동적 재생 지연 기능을 제공하지는 못한다.

2.2.2 알고리즘 2

패킷 i에 대한 지연시간의 측정은 RFC793[5]의 알고리즘을 근거로 계산되며 편차의 측정은 Van Jacobson[6]에 의해 제안된 알고리즘을 사용한다. 패킷 i의 평균 지연시간과 편차는 다음과 같이 계산된다.

$$\underline{d}_i = a * \underline{d}_i + (1-a) * n_i$$
$$\underline{v}_i = a\underline{v}_{i-1} + (1-a)|\underline{d}_i - n_1|$$

이 방식은 현재의 값을 과거의 값에 의존하게 하는 필터(filter) 방식으로 가중치 a의 영향이 크다. NeVot[11] 1.4버전에 구현되어 있고 a의 값은 0.998002로 사용한다. 이 방식은 동적 재생 지연 방식에 해당되며 구현이 간단하다.

2.2.3 알고리즘 3

Mills[7]의 제안을 근거로 하여 만든 것으로서 알고리즘 1의 변형 알고리즘이다. 이는 짧은 기간동안 패킷의 지연이 생길 경우 빠르게 적응하기 위해 TCP의 재전송 타이머가 측정하는 방법과 같이 a와 b라는 두 개의 가중치를 두어서 네트워크 상황에 따라 다르게 적용시키는 것이다.

if ($n_i > \underline{d}_i$) then
  $\underline{d}_i = b * \underline{d}_i + (1-b) * n_i$
else
  $\underline{d}_i = a * \underline{d}_i + (1-a) * n_i$

2.2.4 알고리즘 4

S_i 는 i번째 패킷의 이전 대화단계에서 받은 모든 패킷들의 집합이라고 가정할 때 지연시간은 다음과 같이 계산된다.

$$\underline{d}_i = \min_{j \in Si}\{n_j\}$$

즉 이전 대화단계에서 파악된 네트워크 지연시간 중 가장 최소값을 선택한다. 이 알고리즘 역시 구현이 간단하며 동적 재생 지연방식에 해당되는 것으로 NeVot 1.6버전에 구현되어 있다.

2.2.5 알고리즘 5

이 알고리즘의 특징은 Mills와 Bolot에 의해 보고된 스파이크(spike)[2] 현상에 의해 작동된다는 점이다. 스파이크 현상이란 송신자가 일정한 주기로 일정한 크기의 패킷을 전송하였을 때 수신자에서는 독립된 구간으로 나뉘어져서 패킷이 집중되게 되는데 이 때 몰리게 되는 시작지점에서는 패킷의 네트워크 지연시간이 급격하게 커지는 현상을 말한다. 그리고 스파이크 단계가 끝나면 지연의 변화가 없고 패킷의 집중도 적어지는 안정단계를 찾게 된다. 그림 2는 스파이크 현상을 보여주고 있다.

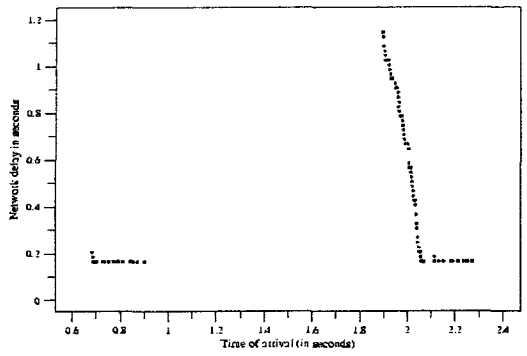

그림 2 스파이크 그래프

그림 3은 이 알고리즘의 가상 코드를 보여주고 있다. 3라인에서 연속적인 패킷의 차이가 임계치(threshold)보다 크면 스파이크 모드로 바뀐다. 스파이크 모드에서는 네트워크 지연시간을 18라인과 같이 계산한다. 7라인에서는 스파이크 모드의 경사도를 측정하기 위해 가장 최근에 받은 두 패킷을 이용한다.

```
1.  n_i = Receiver_timestamp – Sendertimestamp;
2.  if (mode == NORMAL)
3.    if (abs(n_i – n_i-1) > abs(v) * 2 + 800) {
4.      var = 0;  // beginning of spike
5.      mode = IMPULSE;
6.    } else {
7.      var = var / 2 + abs((2n_i – n_i-1 – n_i-2) / 8);
8.      if (var <= 63) {
9.        mode = NORMAL;
10.       n_i-2 = n_i-1;
11.       n_i-1 = n_i;
12.       return;
13.   }
14. }
15. if (mode == NORMAL)
16.   d_i = 0.125 * n_i + 0.875 * d_i-1;
17. else
18.   d_i = d_i-1 + n_i – n_i-1;
19.   v_i = 0.125 * abs(n_i – d_i) + 0.875 * v_i-1;
20. n_i-2 = n_i-1;
21. n_i-1 = n_i;
22. return;
```

그림 3 알고리즘 5의 가상 코드

이 방식은 동적 재생 지연 방식에 해당되는 것으로 구현이 다소 복잡하며 스파이크 모드가 감지되는 상황에서 사용할 수 있다.

2.2.6 알고리즘 6

알고리즘5는 스파이크 현상을 파악하여 재생시간을 지연시키는 알고리즘인데 반해 이 알고리즘은 대화단계와 묵음단계를 파악하여 재생시간을 조절한다. Vat[8]에서 사용하는 이 알고리즘은 재생시간을 적응적으로 조절하는 동적 지연 방식에 해당되는 것이다. 그림 4는 본 연구자가 분석한 Vat의 지터 컨트롤 알고리즘을 가상 코드로 작성한 부분이다.

```
1.  initialize variables;
2.  for (every received packet) {
3.    d_i = a_i – t_i;
4.    o_i = p_i – t_i;
5.    // start of new talk spurt
6.    if ( new_TALKSPURT)
7.      if (v_i < V_THRESH) {
8.        calculate newoffset;
9.        z = z + newoffset;
10.       p_1 =  t_1 + d_1 + z;
11.     } else
12.       p_1 =  t_1 + d_1 + b * v_1;
13.   // packet out of range
14.   else if (o_i > MAX_DELAY){
15.     calculate newoffset;
16.     p_i = p_1 + (t_i – t_1);
17.   else  // packet in range
18.     p_i = p_1 + (t_i – t_1);
19.   d_i = d_i-1 + n_i – n_i-1;
20.   v_i = 0.125 * abs(n_i – d_i) + 0.875 * v_i-1;
```

그림 4 알고리즘 6의 가상 코드

이 알고리즘은 매 패킷이 수신자에게 도착할 때마다 수행된다. 3라인에서는 패킷의 전송시간과 도착시간을 비교하여 네트워크 지연시간을 계산한다. 6-12라인까지는 대화단계의 첫 패킷일 경우에만 수행된다. 대부분의 실시간 응용프로그램들이 전송 프로토콜로 RTP[9]를 사용하기 때문에 대화단계의 첫번째 패킷임을 확인하는 것은 전송된 RTP 패킷의 마커 비트(marker bit)에 의해 가능하다.

7라인에서 네트워크 지연시간의 편차가 임계치 이하의 값이면(알고리즘 4의 스파이크 단계에 해당) 새로운 오프셋(offset)을 구하여 첫번째 재생시간을 계산한다. 오프셋이라는 시간적 요소의 역할은 패킷이 재생되

는 시간과 도착시간의 차이를 계산하여 좀 더 정확한 재생시간을 계산하도록 한다. 오프셋은 재생시간에서 패킷 도착시간을 뺀 차이 값이다. 11-12라인에서 편차가 임계치 이상이면 오프셋에 의존하지 않고 새롭게 재생시간을 구한다.

14-16라인에서는 패킷의 오프셋이 최대지연시간을 초과하게 되면 오프셋과 패킷 i에 대한 재생시간을 다시 설정한다. 재생시간에 늦게 도착한 패킷은 이 부분에서 버려지게 된다. 18라인은 정상적으로 도착된 패킷인 경우이다. 19-20라인에서는 네트워크 지연시간에 대한 평균과 표준편차를 계산한다.

3. 지터 정도 측정 실험

본 장에서는 지터에 영향을 주는 요소를 파악하기 위하여 스케쥴링과 네트워크 지터를 측정하는 실험과 스파이크 현상을 파악하기 위한 실험을 하였다.

실험을 위하여 동국대에 있는 Pentium-III(400MHz, Windows 2000)의 사양을 가진 호스트로부터 KAIST에 있는 Pentium-III(450MHz, Linux 2.2.14) 호스트로 패킷을 전송하였다. 동국대에서 KAIST까지 경유하는 라우터의 숫자는 traceroute를 이용하여 알아본 결과 13-15개로 파악되었다.

3.1 스케쥴러에 의한 지터 변화

좋은 오디오 품질을 제공하기 위해서 오디오 플레이어는 정확한 스케줄링을 제공하여야 한다. 오디오 작업 스케쥴링을 위해 두 개의 타이머를 사용하게 되는데 하나는 오디오 데이터를 샘플링(sampling) 하기 위한 샘플링 타이머이고, 또 다른 하나는 샘플링된 데이터를 압축시킨 후 네트워크로 전송하기 위한 전송 타이머이다. 본 실험에서는 Pentium-III 400MHz의 PC에서 오디오 데이터를 생성시키기 위한 스케줄러가 정확하게 시간을 지켜서 동작하는지를 알아보기 위한 실험을 하였다.

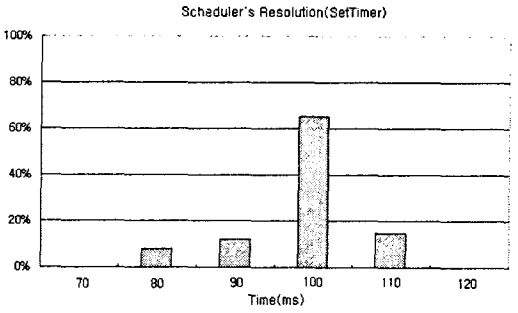

그림 5 SetTimer()에 의한 스케쥴링

그림 5는 오디오 샘플링을 위한 샘플 타이머의 동작 주기를 그래프로 표현한 것이다. 이 때 사용된 스케쥴러 함수로는 VC++의 SetTimer()함수를 사용하였고 타이머의 주기는 80ms로 지정하였다. 현재 Vat 프로그램에서는 PC 타이머의 낮은 해상도 때문에 20ms로 샘플링하는 것을 80ms로 샘플링하고 있다. 그림에서 보는 바와 같이 타이머의 주기를 80ms로 지정하였음에도 불구하고 8%만이 정확한 주기로 동작하였고 무려 80%가 20ms 이상의 오차가 나타나는 것을 확인하였다.

만약 샘플링 타이머의 동작이 늦어지면 오디오 샘플 데이터를 순간적으로 놓치게 되고 전송 타이머의 동작이 늦어지게 되면 패킷이 지연되어 전송되므로 패킷지터 현상이 발생한다. 그러므로 수신자에게 좋은 음질을 제공하기 위해서는 스케쥴러가 정확하게 동작되어야 한다.

3.2 네트워크에 의한 지터 변화

지터를 발생시키는 또 다른 요인은 네트워크 상에서 발생할 수 있는 프로세싱 지연과 큐잉 지연 때문이다. 우리는 네트워크에서 생기게 되는 지연을 파악하기 위하여 동국대에서 KAIST까지 80ms의 주기로 160바이트 크기의 UDP 패킷을 발생시켜 전송하였다.

표 1은 비교적 트래픽 양이 많은 오후 3:30-4:00까지 30분동안 실험한 결과를 보여준다. 표에서는 거의 100%에 가까운 데이터가 100ms 이내의 적은 양의 지터로 수신자에게 도착되었으며 오직 0.05%만이

800ms 이내의 지터를 경험한 것을 나타내고 있다. 이 실험에 의하면 인터넷에서 전송되는 오디오 패킷의 지터는 그리 심각한 정도는 아니다. 그러나 오디오 수신자에게서 파악되는 지터는 오디오의 품질을 저하시키기 때문에 네트워크 지연이 최소화 되어야 한다.

표 1 인터넷에서 지터 측정 결과

Jitter	Rate(%)	Jitter	Rate(%)
100ms	99.65	500ms	0.00
200ms	0.01	600ms	0.00
300ms	0.00	700ms	0.03
400ms	0.17	800ms	0.04

3.3 스파이크 현상 파악 실험

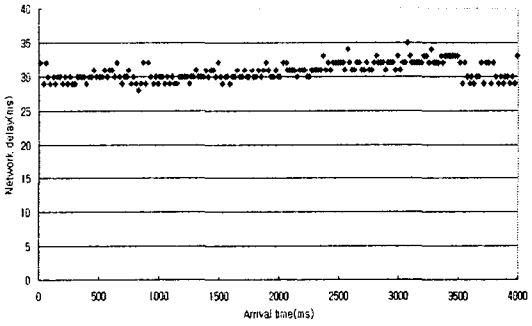

그림 6 도착시간과 지터 그래프

본 실험에서는 알고리즘 4에서 설명한 것처럼 스파이크 현상이 있는지를 측정하기 위해 동국대로부터 KAIST로 160바이트씩 80ms주기로 패킷을 전송하였다.

그림 6은 스파이크 현상을 파악하기 위한 그래프인데 Mills[7]가 언급한 스파이크 현상은 기대와는 달리 발견되지 않았다. 스파이크 현상은 네트워크 상황에 따라 영향을 많이 받는다고 판단되므로 항상 관측되기는 어렵다고 본다. 따라서 알고리즘 5의 경우 스파이크 현상이 있을 때만 유효한 알고리즘이므로 스파이크 현상이 없을 때는 알고리즘 6처럼 대화단계와 묵음단계에 의해 재생시간을 지연시키는 것이 더욱 현실적인 방법일 것이다.

4. 결론

본 논문에서는 패킷지터가 발생할 경우 오디오 오디오 수신자가 재생을 지연시키므로써 오디오의 음질을 향상시킬 수 있는 6가지 대표적 지터 컨트롤 알고리즘의 특성에 대해 살펴보았다. 구현 측면에서는 알고리즘 1,2,3은 구현이 간단한데 비하여 알고리즘 5,6은 다소 복잡하다. 재생지연 방식면에서는 Open h.323에서 사용하는 알고리즘 1만이 정적 재생 지연 알고리즘을 사용하고 나머지는 모두 동적 재생 지연 알고리즘을 사용한다.

현재 인터넷에서 지터에 영향을 주는 요소를 파악하기 위하여 스케쥴링과 네트워크 지터를 측정하였고 스파이크 현상을 파악하기 위한 실험을 하였다. 스케쥴링 실험에서는 전체 데이터의 92%씩이나 정확하게 스케쥴링하지 못했고 네트워크 지터 실험에서는 전체 데이터 중 99%가 100ms 이내의 적은 양의 지터를 경험하였다. 알고리즘 5에 언급된 네트워크 스파이크 현상은 본 연구자가 실험한 바로는 발견되지 않았다.

좋은 품질의 음질을 제공하기 위해서는 패킷 지연에 대해 동적으로 재생 지연할 수 있는 기능이 기본적으로 제공되어야 하고, 오디오 플레이어의 스케쥴링 능력이 정확하게 제공되어야 하며 네트워크 지연에 의한 지터도 최소화 되어야 한다.

향후 연구 과제로는 스파이크 현상 파악에 관한 실험을 계속 진행할 것이며 위에서 소개한 6가지 알고리즘을 구현하여 성능을 모뎀, LAN, WAN과 같은 다른 네트워크 환경에서 적용하여 실험하고 분석할 것이다.

참고문헌

[1] J-C. Bolot and Andreas Vega-Garcia, "Control Mechanisms for Packet Audio in the Internet," *Proceedings of IEEE Infocom*, 1996, pp. 232-239.

[2] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinn, "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks," *Proceedings of IEEE INFOCOM*, 1994, pp. 680-688.

[3] Keith W. Ross and James F. Kurose, "Computer Networking and Internet Protocols", 1996.

[4] J. Pinto and K. Christensen, An Algorithm for Playout of Packet Voice based on Adaptive Adjustment of Talkspurt Silence Periods, Proceedings of the IEEE 24th Conference on Local Computer Networks, October 1999.

[5]Jon Postel, editor, "Transmission Control Protocol specification", ARPANET Working Group Requests for Comment, September 1981, RFC793.

[6] V.Jacobson, "Congestion avoidance and control", Proc. 1988 ACM SIGCOMM Conf., August 1988, pp.314-329.

[7] D.Mills, "Internet Delay experimens", ARPANET Working Group Requests for Comment", December 1983, RFC889.

[8] Jacobson, V., and McCanne, S., "Visual Audio Tool", Lawrence Berkey Laboratory. ftp://ftp.ee.lbl.gov/conferencing/vat

[9] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", January 1996, RFC1889.

[10] http://www.openh323.org

[11] http://www.video.ja.net/mice