

Experiments on QoS Adaptation for Improving End User Speech Perception Over Multi-hop Wireless Networks*

Tsuwei Mario Manthos Yuri Ilya
Chen Gerla Kazantzidis Romanenko Slain

[tsuwei, gerla, kazantz, yuri, ilyas]@cs.ucla.edu**

UCLA - Computer Science Dept - Wireless Adaptive Mobility Lab

ABSTRACT

Ad-hoc wireless networks cannot easily support multimedia applications because of the media high probability, burstiness and persistence of errors. Real-time constraints and multicast make the problem even more difficult. Therefore, in order to improve their performance over the existing best-effort networks, multimedia applications must adapt their operation to constantly changing network QoS. In this paper we propose a programming model that allows audio applications to adapt to changes in network QoS. In our scheme QoS information is continuously fed back from audio clients to the audio server, which uses this information to adapt the characteristics of an audio stream to fit the current network conditions. We have implemented an audio-on-demand application for Windows NT platform that uses this model. We present experiments that confirm the usefulness of our adaptation mechanism for improving the packet loss and delay jitter characteristics of an audio channel in networks with unpredictable QoS behavior. In introducing an ultimate speech layer, we incorporate techniques such as captioning, speech recognition and speech synthesizing. When the QoS notification indicates, this minimal layer takes over to maintain an acceptable level of meaningful communication. Our experiments, both in a simulated and in a real multihop wireless testbed, show that our QoS mechanism improves the characteristics of the audio channel. End user perception can be greatly enhanced, and meaningful communication can be sustained even at most adverse network conditions by using our speech transcription scheme.

1. INTRODUCTION

There is a wide variety of multimedia applications that deliver audio and video over a network. For instance, VIC [7] and VAT [8] are multimedia applications widely used on the internet. However, few of these applications have mechanisms which can take advantage of QoS information from the network, such as packet loss rate, delay jitter and available bandwidth so as to achieve a flexible operation under varying network conditions. As a result, adverse traffic conditions can cause significant degradation in the quality of a multimedia stream in environments like internet or wireless networks, where packet loss can be large and packet delivery time cannot be guaranteed. As reported in [9], the perceived audio quality drops sharply as packet loss reaches 20% for non-adaptive applications even if packet retransmission techniques are used to replace lost packets. Hence, applications operating in such environments must be able to dynamically adjust the characteristics of the multimedia stream to the changing network conditions. In this paper we

propose an adaptation scheme for audio applications based on our QoS Notification Programming Model, in which senders use QoS feedback information from the receivers to dynamically select audio encoding most appropriate to the reported network conditions. The selection of audio encoding is based on the principle of media scaling. By this principle, the bit-rate (and, hence, the quality) of an audio or a video stream is varied to be consistent with the available bandwidth.

Similar concept of application layer adaptation on data rate has been addressed [10]. However, this work focused on a traditional wired network, where lost packets are mostly caused by congestion. In that case, adaptation of data rate may be sufficient to improve the audio or video quality. Our work deals with wireless networks, where packet loss may also be caused by interference. The radio interference is not easily controlled through input rate regulation, therefore a different strategy must be used. In this paper, we describe and evaluate an adaptive, flexible audio application, AudioTool, specifically designed for wireless environments. We demonstrate the effectiveness of this tool in enhancing end user perception by using different QoS adaptation strategies and aggressive speech layering ideas like captioning. The experimental results collected are reported.

In order to improve end user perception, an ultimate encoding layer is introduced when AudioTool is used to stream speech audio. A text transcription is associated with the speech stream, either real-time using a speech recognition engine or from a caption file. The text traverses the path easier (smaller packets, very low bit rate) and more reliably (e.g. using redundancy). The data can be displayed at the receiver side in a caption window or, more importantly, when the QoS feedback suggests switching to this bottom layer, the speech can be reproduced using the transcription with a text-to-speech synthesizer. In this way the speech stream can still be comprehensible at the client side. Switching into this layer must be properly indicated by the adaptation process, so that it is triggered at the point where the layer above it, would not produce comprehensible speech. Experiments in our wireless multi-hop testbed show that the communication quality is substantially upgraded even when adverse network conditions persist for a long time.

Our application allows the receiver to be inside a moving vehicle. The driver can listen to the transmission without ever having to look at the display. Such application becomes useful in a combat or emergency situation when information

*This work was in part supported by the U.S. Department of Justice/Federal Bureau of Investigation, ARPA/ITO under Contract J-FBI-93-112 Computer Aided Design of High Performance Network Wireless Networked Systems, and by Intel under project "QoS Wireless Networks"

** Order of authors is alphabetic. Primary contacts are Manthos Kazantzidis (kazantz@cs.ucla.edu), Ilya Slain (ilyas@cs.ucla.edu)

lines are down but wireless LANs can be quickly deployed. Also, think of an ambulance driver receiving traffic information from other drivers and/or helicopter.

The rest of the paper is organized as follows: in the next section we give a general description of AudioTool. Section 3 describes in detail our QoS notification model both in terms of its components and the dynamics of its operation. Section 4 presents our audio stream adaptation mechanism. In section 5 issues related to the speech recognition, text-to speech synthesizer and synchronization with the audio stream are discussed. Section 6 presents experimental results with our AudioTool. We conclude in section 7 and indicate future work in 8.

2. OVERVIEW OF AUDIOTOOL

AudioTool is a Windows NT client/server application for delivering audio streams to clients over a network connection. It consists of the audio server application and the client playback application parts. The audio server accepts calls from the clients, accesses the requested audio record on the hard drive or switches to real-time speech mode, and streams it to clients over the network via UDP. The client application receives the audio packets and plays them out on the audio I/O device. The playback takes place in real-time except for the initial buffering delay. The buffer is necessary to compensate for possible delay jitter. The characteristic feature of this audio-on-demand application is its ability to dynamically adapt the characteristics of the audio stream to changing network QoS.

The client constantly monitors number of packets lost as well as delay jitter and periodically sends QoS feedback to the server via TCP. The server switches between layers (sampling rates) of the audio stream and changes the packet size dynamically based on the QoS information received. Once a user defined threshold for the error rate is reached, the client uses the text-to-speech synthesizer to reproduce the speech. As can be seen, any adaptation procedure as well as any layering can be easily coded in the AudioTool.

When the audio stream is real-time speech, a text transcription is generated using a speech recognition engine (or read from a file in case of pre-transcribed speech) and sent redundantly piggybacked to the audio packets. Every audio packet sent within a time window of 2 seconds contains the text recognized in the last 2 seconds. Even if only one audio packet gets through every 2 seconds the text -to-speech synthesizer will be able to produce meaningful and continuous speech at the client side.

Our application follows the principles of Application Level Framing (ALF) [11], which advocates closer cooperation between functions traditionally associated with network and application layers. This way, the application can take advantage of the QoS parameter information provided by the network to adapt to changes in network's behavior [12]. AudioTool implements the QoS notification programming model (see Section 3) to provide the audio server with the ability to monitor network conditions at the client end and react adequately when congestion occurs. More specifically, the server uses 1) the knowledge of bandwidth available on the link for the audio stream, 2) packet loss rate, and 3) the

amount of bandwidth required to support the audio stream, to find the optimal audio encoding parameters.

3. QOS NOTIFICATION PROGRAMMING MODEL

We have developed a network-independent programming model for streaming multimedia applications, which defines a general mechanism for QoS monitoring of network conditions by the receiver application and for feeding that QoS information back to the server.

The model is comprised of three main parts: 1. the network layer API which provides the interface to network services, 2. the Network Monitor module which collects and analyses QoS information from the network, and 3. the Application QoS Notification API which accepts this QoS information from the Network Monitor and processes it. Such separation of the model into three modules allows the application to achieve the desired network independence. The first two components shield the Application QoS Notification API from the particulars of the underlying network technology and hence, it can stay the same even though the services provided by the network are likely to evolve with time. We will now describe the model components in greater detail.

3.1 NETWORK LAYER API

Because AudioTool is a Windows NT application, we used WinSock 2 library as our network API. WinSock functions closely resemble those of the well-known Berkeley Sockets for UNIX systems.

We use only standard network operations such as sending and receiving packets, requesting a connection and accepting a connection request. In addition to the support for basic network I/O operations, WinSock 2 has a number of additional features that allow an application to utilize QoS services from the network when those services are, in fact, provided by the network. Specifically, it allows the application to negotiate the desired level of QoS during connection setup time or even re-negotiate the QoS contract after a connection has already been setup. These services are supported, correspondingly, by *WSAConnect()* and *WSAIoctl()* WinSock API functions, which take the extra flow-spec descriptor information containing the requested QoS information. Supplying QoS information to *WSAConnect()* or *WSAIoctl()*, however, is only meaningful when the underlying network provides QoS support. For instance, such QoS specification is essential when using WinSock 2 over an IP network with RSVP [13]. Our QoS Notification Programming model, however, does not assume that such QoS support exists. Therefore, with network without QoS support, the model needs to allow for measurement and reporting of network QoS parameters. The Network Monitor described in the next sections serves this purpose.

3.2 NETWORK MONITOR

The Network Monitor provides a quality of service abstraction for the application, so that it can always assume that the network provides QoS support, while in reality it may not. Its activity consists of the following three parts: 1) it monitors the multimedia stream from the server to the

receiver; 2) it measures QoS parameters of this stream; and 3) it reports this QoS information to the application. We will now address these three activities in greater detail and describe the implementation of Network Monitor in our AudioTool application.

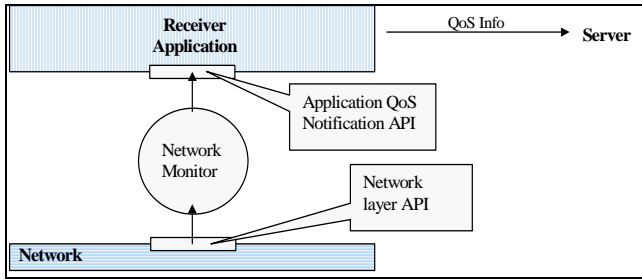


Fig 1: A schematic of QoS notification programming model. The Network Monitor and Network layer API are specific to a particular underlying networking technology, while the Application QoS Notification API stays network-independent.

3.2.1 MONITORING THE MULTIMEDIA STREAM

The Network Monitor analyses the information contained in every packet of the multimedia stream to infer the stream QoS parameters (Fig.2).

The protocol used to send data to the client is similar to RTP but is more simplified and contains only the necessary functionality (see Fig. 3). The server generates the sequence number and timestamp information when it sends a packet over to the client. It stores the two numbers in the audio packet's transport header. When the packet arrives to the destination, the receiver extracts these parameters from the header and passes them to the Network Monitor.

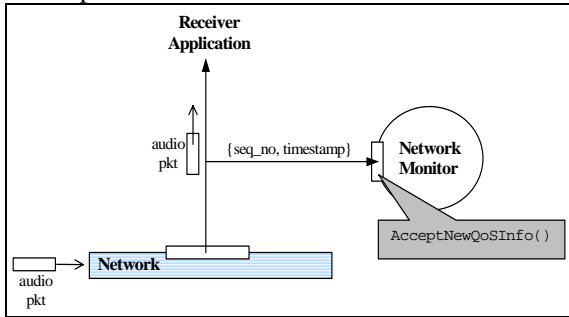


Fig. 2: Network Monitor analyses the packet stream by looking at sequence number and timestamp information in every packet's header. In our AudioTool application, the Network Monitor provides an API function `AcceptNewQoSInfo()`, which accepts information about audio packets coming in from the network

3.2.2 MEASURING QOS PARAMETERS

The Network Monitor uses the sequence number, the timestamp, and the local time information to determine two QoS parameters of the stream:(1) packet loss rate lr and (2) delay jitter jt . The Network Monitor divides time into measuring periods of duration $t_{mp} = 1$ sec. During each measuring period, it counts the total number of packets received n_{total} , and the number of packets lost n_{lost} . It also records the arrival and send times of the last packet in the measuring period: $t_{Last\ Arrival}$ and $t_{Last\ Send}$. The arrival time is taken to be the system time when the packet arrives to the receiver, while the send time is extracted from the packet header. At the end of every period k , the Network Monitor

computes the two QoS parameters with the following simple calculations:

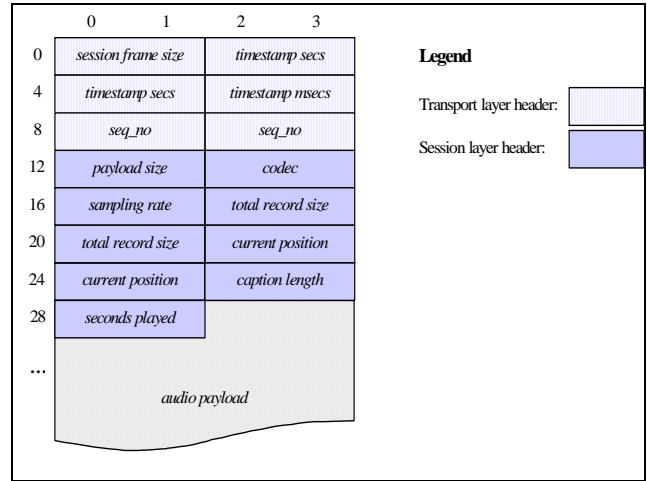


Fig. 3: An audio packet has a variable size of between 270 and about 1000 bytes, of which 30 bytes are used for header information; 240, 480, or 960 bytes are used for audio payload

$$lr(k) = n_{lost}(k) / n_{total}(k)$$

$$jt(k) = [InterArrivalTime(k) - InterSendTime(k)] / InterSendTime(k)$$

where

$$InterArrivalTime(k) = t_{LastArrival}(k) - t_{LastArrival}(k-1),$$

$$InterSendTime(k) = t_{LastSend}(k) - t_{LastSend}(k-1)$$

The two parameters are then reported to the receiver application.

3.2.3 REPORTING QOS INFORMATION TO THE APPLICATION

As shown on Fig.4, the Network Monitor reports three QoS parameters to application: the loss rate, delay jitter, and the amount of bandwidth available to the receiver. While the first two are determined by monitoring the actual packet stream, the third parameter is not computed but is provided by the user. In the future, available bandwidth can be provided by QoS routing [TSUWEI7-QOS].

The reason for the available b/w information being emulated, rather than being measured like the rest of QoS parameters is because the network layer currently lacks the facilities necessary to provide such information. If the QoS reporting functions were incorporated into the network layer (as proposed, for instance, in [14]), the Network Monitor would be able to obtain b/w information directly from the OS. In this case, there would be no need for QoS Emulator and the value for the available bandwidth will be based on actual measurements. It is important to note, however, that regardless of whether some of the QoS information is emulated or not, the Network Monitor module shields the application from QoS measurement details. Therefore, the future evolution of network layer services will not affect the generality of our QoS Notification Application Programming model.

3.3 QOS NOTIFICATION API

The final component in our QoS Notification model is the API that the receiver application provides for QoS reports from the Network Monitor (see Fig 5). As shown on Fig. 5

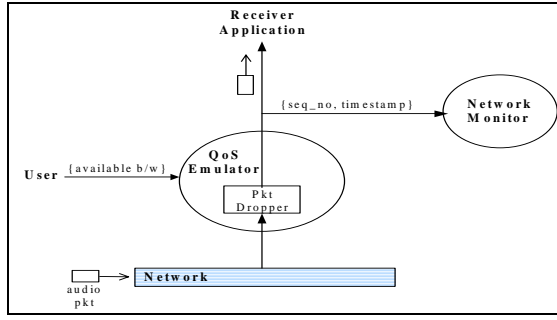


Fig. 4: Network Monitor compiles QoS information and reports it to the application.

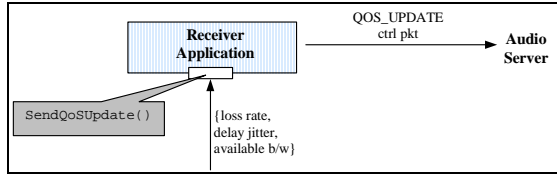


Fig. 5: Upon receiving new QoS information from the Network Monitor, the QoS Notification API consists of a single function, `SendQoSUpdate()`, which accepts the new QoS parameters from the Network Monitor. The receiver application then packetizes the QoS information and transmits it in a special `QOS_UPDATE` control packet to the audio server, which uses it to maintain the optimal audio encoding for the connection.

The rate at which the receiver application transmits `QOS_UPDATE` packets (see Fig. 6) equals the Network Monitor measuring rate, which is once every second. This relatively high rate is dictated by the requirements of real-time transmission of audio: if network conditions worsen at the receiver end, the audio server must react quickly to gracefully degrade the audio stream quality so that the client does not experience interruptions in the transmission.

	0	1	2	3
0	session frame size		message type	
4	available bandwidth			
8	loss rate			
12	delay jitter			

Fig. 6: QoS update message is used for reporting network conditions at the client end to the server. The three parameters reported are available bandwidth, loss rate, and delay jitter.

A structural representation of the application objects, the data QoS paths can be seen in Fig. 7.

4. SOURCE ADAPTATION TO QOS CHANGE

By receiving `QOS_UPDATE` packets, an audio server is continuously aware of the network conditions at the receiver end. Upon receiving an update, it makes a decision on whether to change the current audio sampling rate or leave it intact. This decision is based upon the following heuristics:

If $lr > LR_{UpperThreshold}$ then

$$SamplingRate_{Current} = OneStepDownSamplingRate(SamplingRate_{Current})$$

$$PacketSize_{Current} = PacketSize_{Current}/2$$

If $lr \leq LR_{LowerThreshold}$ and

$$SamplingRate_{Current} < BestFitSamplingRate(AvailableBW)$$

Then $SamplingRate_{Current} =$

$$BestFitSamplingRate(AvailableBW)$$

where:

lr is the loss rate reported by the receiver in the `QOS_UPDATE` message;

$AvailableBW$ is the bandwidth available to the receiver as reported in the `QOS_UPDATE` message.

$SamplingRate_{Current}$ is the sampling rate currently in use by the audio server;

$PacketSize_{Current}$ is the packet size currently in use by the audio server;

$OneStepDownSamplingRate()$ is a function that takes a sampling rate and returns the next lower sampling rate value.

In all, there are only a few possible sampling rates that the PC audio hardware can work with: 8,000 Hz, 11,025 Hz, 22,050 Hz, and 44,100 Hz. So, for instance, $OneStepDownSamplingRate(22,050 \text{ Hz}) = 11,025 \text{ Hz}$.

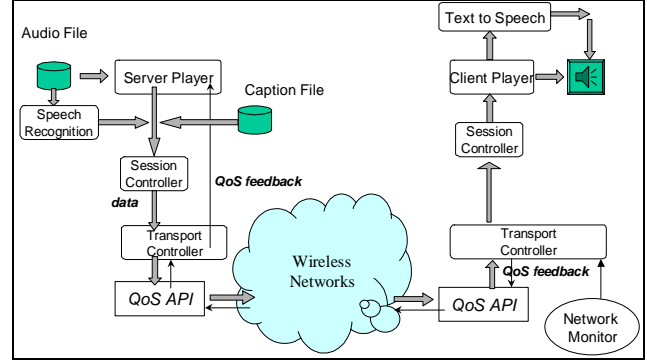


Fig. 7

$BestFitSamplingRate()$ is a function that takes the available b/w of a link as an argument and returns an appropriate sampling rate for PCM audio stream to be transmitted over this link with no packet loss. So, for instance, $BestFitSamplingRate(75 \text{ kbps}) = 11,025 \text{ Hz}$.

$LR_{UpperThreshold}$ and $LR_{LowerThreshold}$ are constants.

A state diagram illustrating the dynamics of this heuristic function's execution is shown on Fig.8.

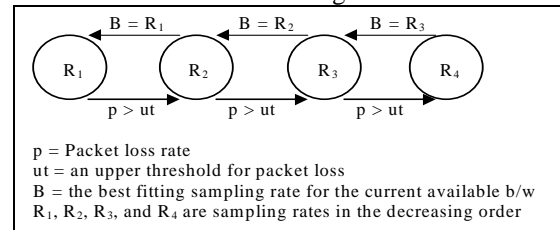


Fig. 8: A state diagram of sampling rate and packet size adaptation mechanism.

This heuristic is based on the assumption that the primary cause of packet loss is congestion. Hence, when the audio server decreases the audio sampling rate, and therefore, its transmission rate, the packet loss should decrease and the perceived speech quality should increase.

Another related heuristic decreases the audio packet size in order improve packet loss characteristics of the channel. When discussing experimental results in the next section we will show the effectiveness of these two approaches and of their combination.

Currently, we have defined $LR_{UpperThreshold}$ to be 10%. This number has been determined by subjectively evaluating the perceived speech quality in the presence of varying packet loss rates: we believe that 10% is about the highest tolerable packet loss rate for general-purpose audio connection, after

which the perceived audio quality drops dramatically. So, if the loss rate exceeds 10%, the server will switch to a lower sampling rate in the hope that the packet loss will decrease. $LR_{LowerThreshold}$ is defined to be zero.

In an environment subject to heavy external interference, such as the wireless network, it is very efficient to dynamically adjust packet rate as a function of interference. When interference is low, packet size should be large in order to minimize overhead. When interference is high, packet size should be reduced to minimize the probability to packet corruption.

In our experimental testbed, the WaveLAN radios drop corrupted packets, so, it is not possible for us to distinguish between loss due to congestion (which should be counteracted by reducing sampling rate) and the loss due to noise corruption (which should be corrected by shortening packet length). To overcome this problem, we assume that a given fraction (say 50 %) of the packet loss is due to interference. Thus, the server reduces the packet size by half whenever excessive lost packets are reported from the receiver, until its value reaches the minimum packet size for the WaveLAN hardware.

5. SPEECH RECOGNITION AND TEXT-TO-SPEECH

Besides the natural layering in sampling rates and varying the packet size to tolerate average network loss rates, we use captioning as an ultimate compression technique to be applied to real-time or preprocessed speech streams. A text transcription is associated with the speech stream either real-time using a speech recognition engine or from a caption file. The transcription is piggybacked to the audio packets in a redundant fashion, so that it is available even at times where the loss rate forbids any meaningful communication. The data can be displayed at the receiver side in a caption window at all times or, more importantly, when the QoS feedback suggests switching to this bottom layer, the speech can be reproduced using the transcription with a text-to-speech synthesizer. In this way we are able to sustain meaningful communication even at times when interference allows none but a single packet to be correctly received every 2 seconds.

5.1 SPEECH RECOGNITION ENGINE

In order to support real-time transcription of speech, a speech recognition engine is used.

Currently Microsoft Speech Recognition Engine version 3 is used to transcribe discrete speech on the fly. Discrete speech, unlike continuous, requires a speaker to pause between words.

A speech recognizer cannot be expected to work sufficiently for our application without training and fine-tuning of its parameters. Our system ideally requires continuous dictation and infinite vocabulary, but speech recognition technology is not in a position to supply this yet.

In order to get sufficient performance out of the speech recognizer we use a limited domain grammar [SR] made up of approximately 1000 words and no grammar rules. Currently user-defined vocabulary training is not supported. Because the speech recognizer works much better for the trained words, our grammar includes words that are used by Microsoft's test sets.

In general, application specific grammars and/or vocabularies can be used to limit the speech recognition engine's possible responses so that the resulting accuracy is acceptable and our captioning scheme can depend on them to reproduce the speech. For example a system used to transmit routing information in a specific city has a naturally limited vocabulary.

5.2 TEXT-TO-SPEECH SYNTHESIZER

We use Microsoft's text-to-speech synthesizer contained in the Microsoft Speech 3.0.

The synchronization between the bottom layer and the above is very important. Since a real speaker may speak with highly variable speed, a speed adaptation algorithm is needed to keep the speaker and the synthesizer in sync. In our implementation, we calculate a step to increase or decrease the speed of the synthesizer every 2 seconds. When the speech is pre-transcribed the synchronization can be performed manually (pre-synchronization). In our application we insert a special character in the caption file to denote the desired 2-second windows. This simple solution produced good results and decreased our development time (Fig. 9).

When we switch down to the bottom layer in real-time speech though, things are not so simple. Every packet contains the words recognized in the last 2 seconds. It is evident that the TTS will normally try to speak parts of the transmitted sentence more than once. Ideally, the TTS engine

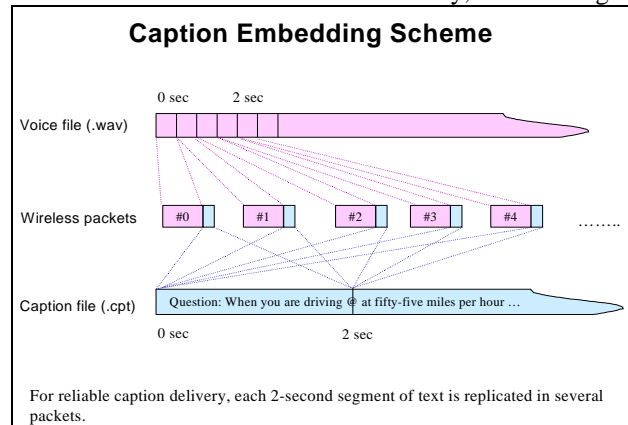


Fig. 9: The pre-transcribed text is separated in 2 second time windows and sent redundantly piggybacked to the audio packets

must be able to speak a word in the list exactly when its audio packets finish. In our implementation we tokenize the phrase into words and feed to the TTS only what "appears" to be a newly spoken word. This produces fully comprehensible speech except for the case of communicating discrete digits with audio (e.g. one-one-three will sometimes be spoken one-three). In absence of a more efficient scheme than the one we use for our experiments, the caption may ensure correctness of delivery by the human receiver (he has to actually look).

Other issues relevant to TTS synthesizer use are:

-How to set the threshold: we expect that a dynamic computation of the threshold will produce better results. It must be set in such a way that the switching between the speech synthesizer and the speaker is not frequent. We currently use a run-time user defined threshold to avoid extensive oscillations of the measured loss rate around it and thus avoiding switching on and off the TTS. Smoothing of the

loss rate measurement is also applied before comparison to this threshold. For a more detailed discussion see [1].

-How to measure the loss rate in the presence of silence: it is desirable not to send "silent" packets. In real-time speech this means however that measuring the loss rate will not be straightforward since we do not have an expected rate. The first packet received may be used to indicate the start of valid audio transmission. In our implementation, since we are primarily interested in observing the quality of the speech, we simply ignore the 100% loss rate measurement whether it comes from silence or real loss. This results in less accurate QoS measurements but it is much simpler.

The text-to-speech synthesizer has features that can help in the case of a multimember conference (multicast). A receiver can recognize or distinguish the speakers, when the text-to-speech synthesizer is on, by having the application manipulate parameters like the pitch of the speaking voice, personality of the voice (e.g. business, casual, computer, excited, singsong), gender of the voice and age of the voice.

6. EXPERIMENTAL RESULTS

To investigate the effectiveness of our adaptation model and the improvement in content preservation, a number of experiments were performed using the AudioTool. Three test environments were used: one that uses an emulated channel, and the other two using a real multihop wireless testbed.

For experiments with emulated channel, both server and client run on the same PC and packets are delivered through a virtual channel. This virtual channel can be manipulated to produce different levels of packet loss according to user specifications through the QoS manipulator.

The wireless testbed, on the other hand, consists of Lucent WaveLANs with multihop routing implementation [16]. Speech packets are transmitted wirelessly. Since there are several interference sources in the real world, the channel behavior can not be controlled.

A second set of real wireless testbed experiments has been performed for evaluating the effect of transcribing the speech. In these experiments we used the topology shown in Fig. 10. Our multi-hop testbed consists of a server, a client, a gateway and an interfering station. Both server and client run on NT/95 platform, whereas the gateway runs on Linux. A WaveLan is used for wireless networking. In this setting we can control our experiments better due to the static routing and the use of the interfering station. This station can transmit UDP packets at any given rate using a small socket application. Alternatively it can fill up the network with ping packets of variable sizes and patterns. Even though it resides in a different sub-network the packets are sensed from the WaveLan providing the desirable effect. We use this setting to create a real high loss environment to push our application in using the transcription to reproduce the speech.

6.1 EMULATED CHANNEL

The effectiveness of AudioTool is verified on the emulated channel. As shown in Fig. 11, we vary the available bandwidth on the channel manipulator. This causes congestion and packet drop on the emulated channel. The AudioTool reacts by reducing the sampling rate. Fig. 12 shows the histogram of packet loss rate and the corresponding

audio sampling rates. We can see that whenever packet loss rate exceeds 10%, the server drops the sampling rate one step (until it reaches 8 kHz, which is the minimum supported sampling rate). On the other hand, when there is no packet loss, the server increases the sampling rate if there is enough available bandwidth. In this experiment, we can see how the server has managed the packet loss rate by varying the sampling rate according to the reported QoS parameters from the client.

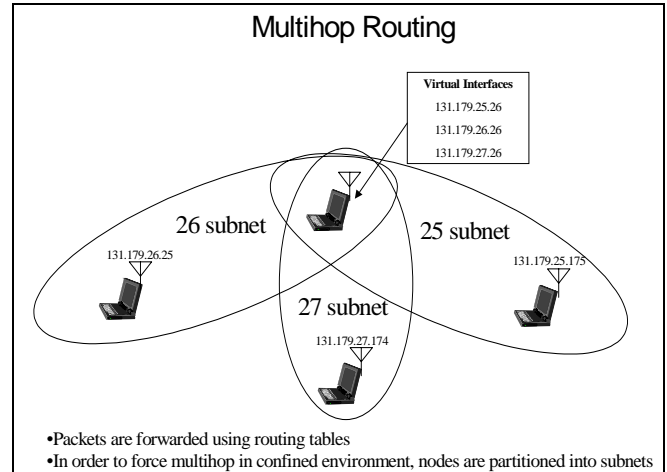


Fig. 10: The topology used in the experiments for testing the speech captioning feature

Human perception evaluation was done, by having a group of people listen to the playback generated by the client application. Overall, the audience was able to recognize most of the speech content during the playback, but it also detected the change in the audio quality whenever the server switched the sampling rate. This indicates that most of the content is preserved with the existence of packet loss.

6.2 WIRELESS NETWORKS

The second set of experiments was performed in a real mobile wireless environment, where the packet loss resulted from actual network conditions rather than from manual adjustment of the channel emulator, as used in the previous experiment. Fig.13 to Fig.16 illustrate the results for following QoS adaptation policies: Fig.13 shows the results when neither sampling rate nor packet size are used. Fig. 14 and Fig. 15 show the effect when either sampling rate or packet size is used, respectively. Finally, Fig. 16 shows when both sampling rate and packet size are used for QoS adaptation.

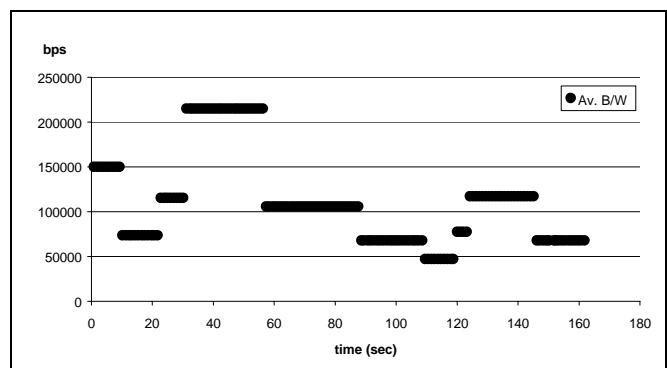


Fig. 11: Histogram of the available bandwidth on the emulated channel.

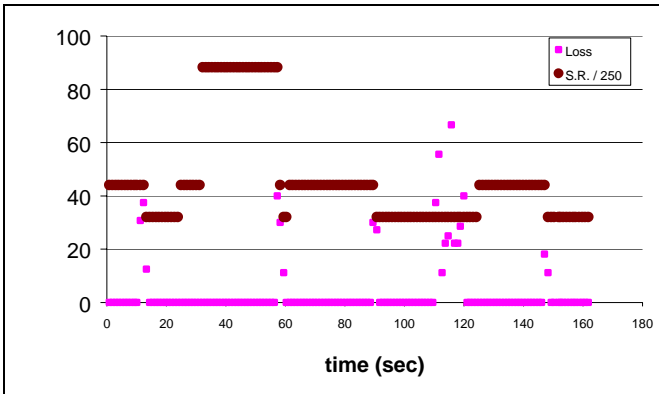


Fig. 12: Histogram of audio transmission in the emulated channel. Packet loss is expressed in percentage points. Sampling rate is in Hertz. The values for sampling rate have been scaled down by a factor of 250.

From these results, we make the following observations:

- The adaptive schemes react promptly to packet loss and jitter, leading to adjustments in sampling rate and packet size which improve performance and drive the system to the most effective operational regime.
- The heavy loss periods are much longer in the non-adaptive case than in the adaptive one (compare Fig. 13 and Fig. 16). Long audio “black outs” (several seconds) are extremely disruptive.
- Packet size adjustment appears to be particularly effective in reducing loss rate (as shown in Fig. 15). This was expected since random channel interference is the major problem in our experiments. In these conditions, the shorter the packet, the lower the probability of corruption. The long burst of lost packets in Fig. 15 in the interval (130-180) is due to particularly strong, persistent external interference (recall that we have no control on external interference in our experiments).
- The adaptation of both packet size and sampling rate provides the best performance. Height and width of loss bursts are smallest in this case (see Fig.16).

Interestingly, the traces also seem to show an apparent correlation between the observed jitter and packet loss at the receiver. The jitter follows the same tendencies as the packet loss, so that its increases and decreases follow those of the packet loss curve. This observation adds extra justification for our adaptation scheme. Although our scheme explicitly manages only the packet loss on the channel and does not directly attempt to decrease the jitter, it does, in fact, appear to improve both of these parameters because of the correlation that exists between the two.

6.3 LAB EXPERIENCE WITH PRE-TRANSCRIBED SPEECH

Extensive experiments with pre-recorded and pre-transcribed speech have been performed. They show that incorporating transcription and using a text-to-speech synthesizer at the receiver end allows for more comprehensible speech streaming. These successful experiments indicate that similar results can be expected when the transcription is produced by a speech recognition engine on the fly.

In these experiments a pre-recorded speech sample is manually transcribed in an accompanied text file. The text is separated in 2-second time windows as seen in Fig. 9.

We noted a considerable upgrade in sustained user perception. The speech synthesizer could take over and deliver meaningful speech to our ears continuously even when the loss rate was insisting on 99%. Theoretically we need at least one packet to go through in every 2-second window and this was what we experienced in practice too. Only when the interfering station was attempting to fill the ‘ether’ by *pinging* the gateway (using the -f option and 960 byte packets) did it completely capture the channel and noticeable intervals and “black-outs” occurred in our speech stream. The loss rate at those times was reported 100%.

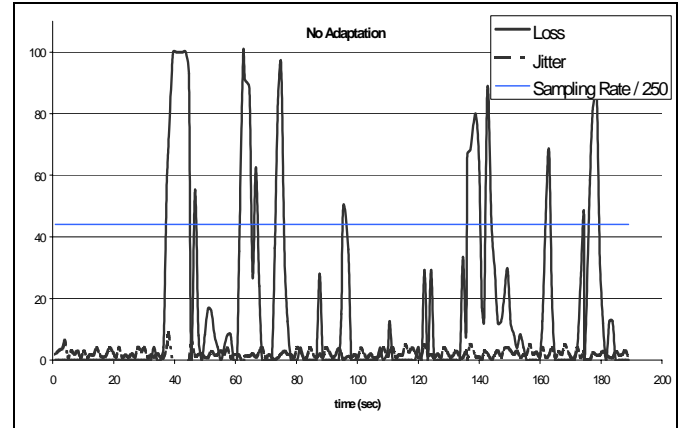


Fig. 13: No Adaptation

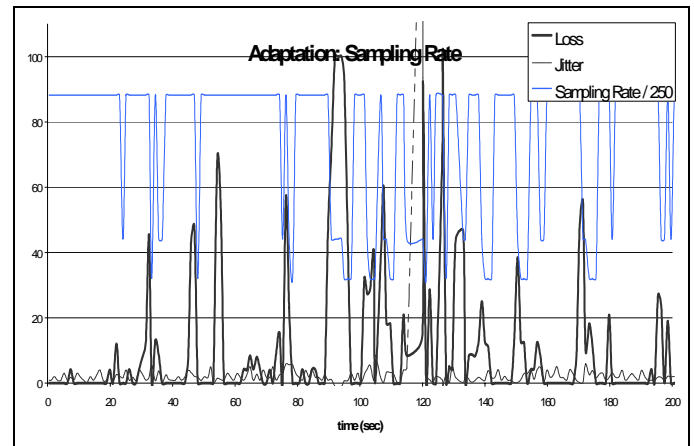


Fig. 14: Adaptation on Sampling Rate

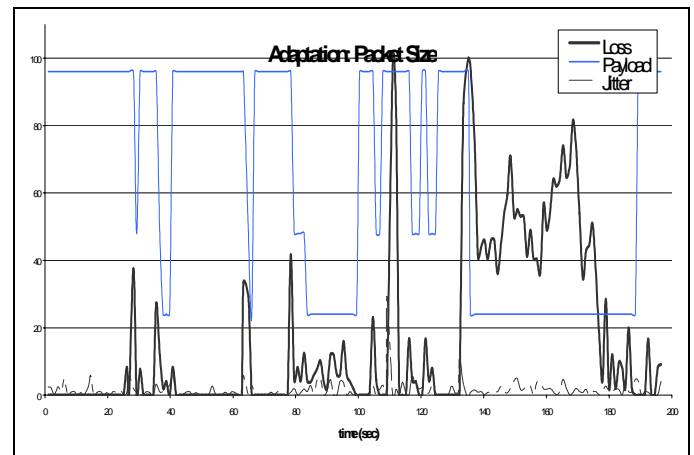


Fig. 15: Adaptation on Packet Size

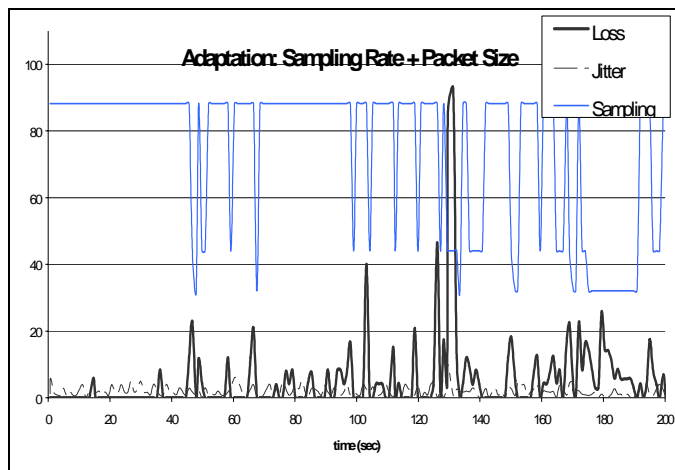


Fig. 16: Adaptation on Both Sampling Rate and Packet Size

We also run experiments using real-time speech. These experiments were primarily used to evaluate the speech recognition performance. When the engine is trained and the 1,000-word vocabulary is in use (also see Sec. 5.1) the SR engine provided sufficient hit rate for the captioning scheme to work.

7. SUMMARY AND CONCLUSION

In this paper we presented an adaptation model for audio applications that enables them to utilize QoS information from the network in order to adapt to the changing network conditions. The adaptation mechanism is based on a media-scaling technique, in which an audio server varies the audio sampling rate and packet size to keep the bandwidth consumption in line with the amount of available network resources. Such continuous adjustment of the source encoding based on the feedback information from the client brings down the overall packet loss and results in better perceived content quality.

We have implemented the AudioTool, an audio-on-demand client/server system, based on the adaptation model. We have run a number of experiments with this application on an emulated channel and have confirmed that the use of this model does bring about audio quality improvements. We have also used this application in a real mobile multihop testbed and have discovered a correlation between the packet loss and delay jitter, which suggests that our controls will contribute to an overall lower delay jitter as well as packet loss rate.

We also showed that the goal to keep a meaningful audio communication even when the sender, the receiver or intermediate hop stations enter a high interference zone is achievable by using text transcription. In real-time speech the performance of the speech recognition engine is the critical factor that will determine whether we can get the same results as the ones we get for pre-transcribed speech. However, with present speech recognition technology we have achieved 99% recognition success by limiting vocabulary and fine-tuning the engine.

8. FUTURE WORK

Future work in this area involves extending the model to allow the delivery of audio information to multiple clients

over either unicast or multicast networks. This extension will be compliant with RTP/RTCP [15] specification, which has explicit support for multiple recipients. The adaptation mechanism can also be improved by using other methods in addition to media-scaling, such as varying audio codecs to trade off bandwidth consumption for audio quality. For instance, an audio server may switch to CELP encoding (4.8 kbps) from its usual PCM encoding method (64, 88.2, 176.4 kbps), if it finds that lowering the sampling rate will still not bring down the bit rate of the stream to the appropriate levels.

REFERENCES

- [1] Manthos Kazantzidis, Tsuwei Chen, Yuri Romanenko, Mario Gerla, "An ultimate encoding layer for real-time packetized voice streaming and experiments over a multi-hop wireless network" - Second Annual UCSD Conference on Wireless Communications, San Diego, California, March 1999
- [2] Yuri Romanenko, "QoS Adaptation for Wireless Networks" -MS Thesis UCLA CS 1998
- [3] Ilya Slain, "A Programming Model for Adaptive QoS-aware Audio Applications" - MS Thesis UCLA CS 1997
- [4] T.-W. Chen and J.T. Tsai and M. Gerla, "QoS routing performance in a multi-hop, wireless networks" - IEEE 6th ICUPC Oct 1997
- [5] Ilya Slain, Yuri Romanenko, "Audio on Demand Application" -Design Notes UCLA CS
- [6] Microsoft Speech Suite Documentation <http://www.microsoft.com/stg> - Microsoft Corp.
- [7] S. McCanne and V. Jacobson, "vic: A flexible framework for packet video" - Proc. of ACM Multimedia 1995
- [8] I. Kouvelas and V. Hardman and A. Watson, "Lip synchronization for use over the internet: Analysis and implementation" - IEEE Globecom 1996
- [9] J.-C. Bolot and A. Vega-Garcia, "The Case for FEC-Based Error Control for Packet Audio in the Internet" - ACM Multimedia Systems 1998
- [10] D. Sisalem, "End-to-End Quality of Service Control using Adaptive Applications" - IFIP Fifth International Workshop on Quality of Service 1997
- [11] D. Clark and D. Tennenhouse, "Architectural considerations for a new generation of protocols" - Computer Communication Review vol20, num4, Sep 1990, pages 200-208
- [12] J. Du et al., "An Extensible Framework for {RTP-based} Multimedia Applications" - Network and Operating System support for Digial Audio and Video May 1997 pages 53-60
- [13] StarDust Technologies, "Windows Sockets 2 Protocol-Specific Annex, Rev. 2.0.3" May 1996
- [14] T.W. Chen et al., "A New Scheme for Fault Tolerance in Wireless Networks", The 27th Annual International Symposium on Fault-Tolerant Computing 1997
- [15] H. Schulzrinne et al., "{RTP}: A Transport Protocol for Real-Time Applications", RFC 1889 Jan 1996
- [16] R.M. Fernie, "Implementation of various routing algorithms for {TCP/IP} wireless networking, UCLA Computer Science Department Jun 1997