An Adaptive ARQ Timeout Approach for Audio Streaming over Bluetooth

Ling-Jyh Chen, Rohit Kapoor, Sewook Jung, M. Y. Sanadidi, and Mario Gerla

***Abstract*— Streaming audio has become a popular form of media on the Internet. As wireless personal area network architectures (e.g. Bluetooth) are now targeted to support multimedia traffic, streaming audio over these technologies will give rise to interesting applications. But, the variable nature of the wireless medium will not lend itself easily to supporting audio streaming. In this paper we focus on Bluetooth and propose an enhancement to the Bluetooth link layer ARQ mechanism to compensate for channel degradation and to better support audio streaming. Specifically, our scheme adaptively sets the ARQ timeout value based on current channel conditions. We show through simulation and testbed experiments that the adaptive ARQ improves the streaming quality significantly compared to the "vanilla" link layer of Bluetooth, especially in noisy environments. Our proposed approach is simple to implement and can actually be extended to the link layer of any wireless technologies.**

***Index Terms*— Audio Streaming; Bluetooth; Adaptive ARQ; BER**

I. Introduction

The last few years have seen an impressive growth in multimedia content on the Internet. One striking success in this area has been MP3 audio, which has led to the development of MP3 players, such as the iPod [1]. Another orthogonal area of growth has been wireless, both in wide area technologies such as 2.5G/3G and local area technologies such as 802.11b and Bluetooth. One interesting usage scenario of Bluetooth is the PAN (Personal Area Network), i.e. a network of personal devices such as laptop, PDA, camera, MP3 player etc. that a person carries with her.

Ling-Jyh Chen is with the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: cclljj@cs.ucla.edu).

Rohit Kapoor was with the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095 USA. He is now with the Qualcomm Inc. (e-mail: rohitk@cs.ucla.edu).

Sewook Jung is with the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: sewookj@cs.ucla.edu).

M.Y. Sanadidi is with the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: medy@cs.ucla.edu).

Mario Gerla is with the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: gerla@cs.ucla.edu).

A number of interesting usage scenarios arise when audio streaming (MP3) content is combined with wireless technologies, such as Bluetooth. For instance, a Bluetooth-enabled MP3 player can stream MP3 audio to a Bluetooth-enabled headset, allowing the user to be "locally" mobile while listening to songs. In another scenario, MP3 content could also be streamed through a 2.5/3G cellphone to a Bluetooth headset. Another scenario could be an MP3 player downloading songs from a LAN access point, and playing them real-time.

These and several other wireless streaming applications are envisioned. However, the varying nature of the wireless link can make audio streaming over wireless a challenging problem. In fact, a number of sources operate in the same ISM frequency band and thus can interfere today with Bluetooth and 802.11 (e.g. microwave ovens, cordless phones, etc).

Good quality audio streaming requires that audio packets reach the client at a consistent rate within regular delays, even though audio (e.g. MP3) is not typically very high-bandwidth in nature. Clearly, a bad channel can cause packets to be dropped/delayed, adversely affecting the quality of streamed audio.

A well-designed link layer can go a long way in solving some of these problems. Most wireless link layers of today employ some kind of ARQ mechanism to protect packets from a bad channel. While this can be greatly beneficial to non-real-time TCP traffic, it needs to be judiciously used when real-time/streaming traffic is being transferred. For example, if the ARQ retransmission limit is too high, packets can get severely delayed under bad channel conditions. On the other hand, a very low value of the ARQ limit can cause a large percentage of packets to be dropped at the link layer. Either of these situations will reduce the quality of streaming traffic. It is thus important that the ARQ retransmission limit be chosen in the correct range.

In this paper, we investigate adaptive settings for the ARQ retransmission limit based on wireless channel conditions. We show that an adaptive selection of the retransmission limit by far outperforms a fixed limit. We propose a simple algorithm to calculate the retransmission limit based on the transmission history of previous packets. Finally, we implement this algorithm both in the NS-2 simulator and on a Linux Bluetooth testbed. The experiment results show that our proposed approach improves performance of audio streaming under a wide range of channel conditions. While our solution was developed for Bluetooth, it can be easily applied to other wireless ARQ link layers.

The rest of the paper is organized as follow: Section 2 describes the Bluetooth technology and the related work. Section 3 and 4 present our approach and the details of its implementation. Section 5 and 6 reports experiment results from simulation and the testbed. Finally, section 7 concludes the work.

II. Background and Related Work

A. Bluetooth Overview

Bluetooth [23] is a short-range radio technology operating in the unlicensed 2.4GHz ISM (Industrial-Scientific-Medical) frequency band. Its original goal was to replace the numerous proprietary cables to provide a universal interface for devices to communicate with each other. But it soon became a good solution to interconnect devices to form so-called personal area networks [10], primarily due to the low cost, low power and small size of Bluetooth chips.

Bluetooth employs FHSS (Frequency Hopping Spread Spectrum) to avoid interference. There are 79 hopping frequencies (23 in some countries), each having a bandwidth of 1MHz. Frequency hopping is combined with stop and wait ARQ (Automatic Repeat Request), CRC (Cyclic Redundancy Check), and FEC (Forward Error Correction) to achieve high reliability on the wireless links.

Bluetooth units can be connected to each other to form a piconet, which consists of up to eight active units. One of the units acts as a master and the others act as slaves. All the data/control packet transmissions are coordinated by the master. Slave units can only send in the slave-to-master slot after being addressed in the preceding master-to-slave slot. Each slot lasts for 625 microseconds.

For real-time data such as voice, Synchronous Connection Oriented (SCO) links are used, while for data transmission, Asynchronous Connectionless Link (ACL) links are used. There are several ACL packet types, differing in packet length and whether they are FEC coded or not. Table 1 shows the different ACL packet types and their properties.

TABLE I: DIFFERENT BLUETOOTH ACL CONNECTION MODES

Mode	FEC	Packet		Symmetric Throughput (kbps)	Asymmetric Throughput (kbps)	
		Size (bytes)	Length (slots)			
DM1	yes	17	1	108.8	108.8	108.8
DM3	yes	121	3	258.1	387.2	54.4
DM5	yes	227	5	286.7	477.8	36.3
DH1	no	27	1	172.8	172.8	172.8
DH3	no	183	3	390.4	585.6	86.4
DH5	no	339	5	433.9	723.2	57.6

Note that, in the symmetric connection mode, both master and slave nodes will occupy the same amount of Bluetooth time slots (625 microseconds in each time slot); whereas in the asymmetric connection mode, the Bluetooth link will occupy 1/3/5 time slots (for DM1/DM3/DM5 or DH1/DH3/DH5 mode) in one direction of this link and only one time slot in the opposite direction.

However, due to operating in the unlicensed 2.4GHz band, Bluetooth can be subject to interference from other wireless technologies, such as 802.11b, HomeRF, cordless phones and other sources such as microwave ovens. As various studies [6][12][15] have shown, these sources can severely degrade Bluetooth performance.

B. Audio Streaming

Streaming audio (aka MP3 files) has become very popular on the Internet, and improving streaming quality has been a topic of active research. While receiving streaming multimedia, users expect smooth and uninterrupted quality and real-time effect, which has very different requirements than best-effort applications, e.g. FTP and HTTP. However, real-time data is very sensitive to network conditions, and any delays or packet losses generated by the network can degrade the stream quality. Much of the work in this area has looked at improving streaming over wired networks. For example, [17][18][20] propose that the sender should dynamically adjust its sending rate by considering the media encoding, the frame rate, and the priority of media parts. [16] allows the receiver to select the most appropriate stream when the connection is setup. Additionally, recent research has focused on utilizing the available bandwidth and making the stream friendlier with other TCP flows. For example, TEAR [19] deploys an end-to-end, TCP equation-based approach to adjust its stream sending rate. TCP-Friendly Rate Control (TFRC) [8], another equation-based approach, computes the subsequent sending rate on the sender side, instead of on the receiver side (i.e. TEAR). VTP (Video Transport Protocol) [2], a new streaming protocol inspired by TCP Westwood [24], adapts its sending rate according to the estimated "Eligible Rate". On the other hand, some commercial products claim to support adaptive streaming, e.g., Helix Universal Server [9] and Microsoft Media Server [13], though lack of product disclosure and related analysis hinders independent efforts to verify the claims or to evaluate the streaming performance.

However, all the above works have an implicit assumption that packet losses and delays are mainly due to network congestion, and the delay time caused by link layer retransmissions is negligible. Such an assumption is valid for wired connections, but not for wireless links, where the bit error rate is much higher than in wired links. While operating over wireless links, the retransmissions on the link layer usually results in a significant delay for each retransmitted packet. Such a delay can limit the applicability of end-to-end approaches and degrade the streaming quality.

C. *Link Layer Enhancements*

A wireless communication channel typically exhibits higher error rates due to attenuation, fading, scattering, or interference from other active sources. In order to provide more reliable data transmission over wireless links, modern wireless technologies usually employ different levels of retransmission and/or redundancy techniques in their link layer level against wireless errors [7]. For example, most wireless technologies perform ARQ (Automatic Retransmission reQuest) to ensure the integrity of the link layer, and some technologies, such as 802.11a and Bluetooth, utilize redundancy techniques to enhance transmission reliability by employing FEC coding in link layer packets.

However, such techniques may not lend itself easily to supporting audio streaming applications. For instance, as the link quality becomes worse, multiple retransmissions become more and more frequent. Setting a high retransmission ARQ limit can ensure the successful data transmission; however, it also leads to extremely large delays in audio packets, thus degrading the audio quality. Though most audio clients use a playout buffer to alleviate such problems, this does not solve the problem when the link quality is very bad for a sustained period of time (i.e. the buffer will become overflowed, and the data will then be dropped). A low retransmission limit, on the other hand, can keep the clients catching up the speed of audio streaming (i.e. keep the real-time effect); however, it leads to a high percentage of packets being dropped at the link layer, which also results in poor audio quality.

Additionally, redundancy methods such as FEC coding are only effective in counteracting random losses, but the connection is still vulnerable to burst channel errors. While wireless channel errors are well-known bursty and dependent in occurrences, FEC coding based approaches may still need large number of retransmissions when bursty errors are present, and the streaming quality will thus become degraded.

In order to overcome the problem, S. Krishnamachari et al proposed a cross-layer approach [11] to adaptively change the maximum number of MAC layer retransmissions and FEC encoding level in the application layer using the estimated MAC layer link quality (SNR). However, the link quality estimation is based on the previous measured link qualities, which may not be appropriate and applicable to the future link quality prediction. In our testbed experiments, we have found that the link quality oscillates very often and dramatically due to the wireless network dynamics, and the prediction of future link qualities is almost impossible to be achieved barely by the measured link quality in the previous history.

Thus, in order to maintain good quality of streaming, not only should the packet loss rate be kept small, but the delays of packets should also not be allowed to increase too much. We present our link layer solution to this problem in the next section.

III. Proposed Approach

In this section, we present the importance of the application-aware link layer and our proposed approach, i.e. Adaptive ARQ RTO (retransmission timeout).

A. *Application-Aware Link Layer*

It is evident that a non-application-aware link layer can result in reduced performance. For instance, if one fragment of an audio packet (say a RTP packet) is dropped (due to the RTO limit being exceeded), the non-application-aware link layer will still try to send the remaining fragments of the RTP packet, even though they are of no use since the receiving link layer will never be able to reassemble the RTP packet. This will result in wasted bandwidth and eventually degrades the overall application performance.

However, some sort of "*cross-layer optimization*" may be of great help in improving the performance of the upper layer applications. A well-tuned link layer should adapt its configurations/behavior in accordance to the present network conditions in order to maximize the application performance. More specifically, for the audio streaming discussed in this paper, the link layer needs to be able to measure the RTT for each RTP packet and perform different levels of adaptation in the link layer. The application-aware functionality required is that the Bluetooth stack should be able to identify and deal with a RTP packet since a vanilla link layer can only deal with Bluetooth baseband packets. The details of how this functionality is implemented are explained in section IV and the details of link layer adaptation are described in the following subsection.

B. *Adaptation of ARQ RTO value*

Bluetooth uses a stop-and-wait ARQ at the link layer and retransmits a packet until either the acknowledgement of a successful reception is received or the retransmission timeout is exceeded, at which point the packet is dropped. However, in most current Bluetooth chipsets, the default value of the retransmission timeout (RTO) is infinite. An infinite timeout value makes the link layer reliable, since packets are not dropped even when the channel conditions are very bad. However, this can lead to problems for real-time/streaming media, since packets may be severely delayed when link quality is poor.

A simple approach could be to use a fixed, finite RTO value. This will result in packets being dropped at the link layer, whenever the retransmission limit is exceeded. Since a severely delayed packet may be completely useless at the client, dropping such a packet may be a good idea anyway since subsequent packets have a higher chance of being transmitted. Thus, this approach may improve audio quality if the retransmission timeout (RTO) can be selected judiciously. Therein lies the problem of using a "fixed, finite RTO" since it may not be easy to find a timeout value suitable for all the cases, such as different link qualities.

Moreover, if such a setting is not appropriate, it may again degrade the audio quality. For example, if the fixed timeout value is too small, it will increase the drop rate; if the value is too large, it will lead to the same situation as the infinite RTO setting.

To overcome such problems, we propose an adaptive ARQ RTO approach that adapts the value of the link layer RTO based on the measured properties of previous packets. Thus, if the link layer has spent too much time on the previous few packets, it should decrease the RTO setting for the next packet, since the audio client has already experienced much delay from the previous packets. In other words, it is better to risk to drop the next packet (due to the decrease in RTO) than to incur another increase in delay. On the other hand, if the link layer has sent the previous packets very efficiently with short RTTs, it should increase the RTO value since the client has already saved time on previous packets and is capable of tolerating some delay for the next one. Thus, it pays to put some extra effort to reduce packet loss (by increasing RTO).

Namely, the link layer measures the *RTT* (round trip time) of each audio packet, say RTP [21] packet, which is the time for the whole RTP packet to get transmitted by the link layer (this implies the use of an application-aware link layer, as we discuss later). Using the value of the *RTT*, a smoothed RTT, *SRTT*, is calculated (Eq. 1), from which the *RTO* is calculated. The *SRTT* and *RTO* update equations are:

$$SRTT' = (1-\gamma) \times SRTT + \gamma \times RTT \tag{1}$$

$$RTO' = \begin{cases} \alpha \times RTO\,; \text{if RTT} < \text{SRTT} \\ \beta \times RTO; \text{if RTT} > \text{SRTT} \\ RTO; \quad \text{if previous packet is dropped} \end{cases} \tag{2}$$

where we take $\gamma = 0.25$, $\alpha = 1.1$, and $\beta = 0.9$. Note that the RTO is dynamically updated using a multiplicative increase/decrease scheme following the threshold check. RTO increases when RTT decreases and vice versa. This is very different from TCP, where the RTO is increased proportionally to RTT.

In addition, we also apply upper and lower bounds to the *RTO* value. The lower bound RTO_{min} is taken as 2 times T_{packet}, which is the time interval between two RTP packets sent on the server side. T_{packet} can be easily derived from the packet size of the RTP packet, and we will present the calculation in the next section. We found through our experiments that if the RTO value was set close to the T_{packet}, too many packets were dropped at the link layer due to the RTO limit being exceeded. Thus, 2 times the T_{packet} proved to be a good lower bound.

The upper bound RTO is proportional to the available buffer size, as shown in the following equation.

$$RTO_{\max} = T_{packet} \times Max(AvailableBuffer \times 75\%, 2) \tag{3}$$

where *AvailableBuffer* is the system maximum input buffer size minus the used buffer size, divided by the RTP packet size. This equation takes into account the fact that if the RTO for an RTP packet is too large, it may cause new incoming RTP packets to be dropped from the input queue due to overflow. In fact, we found that for very large values of RTO, a number of packets were dropped because the buffer was full. Limiting the RTO to this upper bound prevented such packet drops.

Note that, in Eq. 2, we do not update the RTO if the previous packet has been dropped due to timeout. However, because contiguous packet dropping is harmful to audio quality, we reset the RTO to RTO_{max}, using Eq. 3, if at least two of the last 5 packets have been dropped.

IV. Implementation

In order to evaluate our proposed approach, a Bluetooth simulation model has been implemented in NS-2 simulator [14], and a Linux-based Bluetooth testbed has been created. For simplicity, we used DH5 packets in all experiments and stream the MP3 audio using RTP protocol. The details of the implementation and experiment scenarios are presented in the followings.

A. Implementation of Bluetooth Simulation

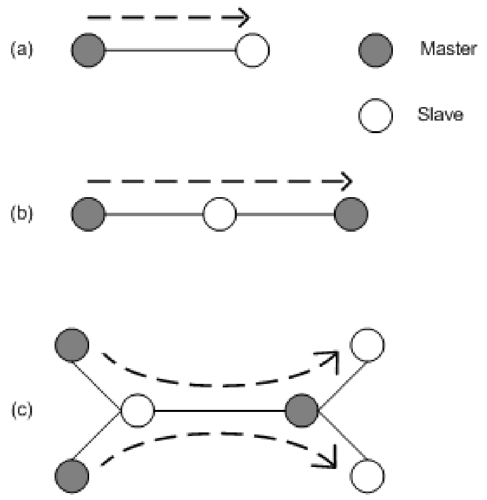

Fig. 1. Various Bluetooth Scatternet topologies

The Bluetooth simulation model was developed as an extension to NS-2 (Network Simulator). The simulation model implements most of the features of the Bluetooth baseband layer, such as frequency hopping, time division duplexing, multi-slot packets, fragmentation and reassembly of packets. In addition, the simulator enables scatternets and inter-piconet communication by defining gateway nodes to forward packets between piconets. Our proposed approach, as well as fixed ARQ RTO value approaches, is also included in the simulation model.

Fig. 1 shows the Bluetooth network topologies used in the simulation, where topology (a) was used to simulate one audio stream in a one hop connection (in one piconet), topology (b) was used to simulate one audio stream in a two hops connection (the streaming is sent from one piconet to the other one, where both sender and receiver are master nodes), and topology (c) was used to simulate two audio streams in a three hop connection (these two streaming flow share one bottleneck link). While topology (a) and (b) are designed to evaluate the packet loss rate and average delay, topology (c) is employed to evaluate the fairness of our proposed approach.

For simplicity, all simulation use DH5 packet type as the link layer packet type, and the error model is assumed to be uniformly distributed with a given bit error rate *b*. The link layer packet error rate P_{err} can be therefore obtained by equation 4.

$$P_{err} = 1 - (1 - b)^{339*8} \tag{4}$$

B. *Implementation of Bluetooth Testbed*

We implemented both the fixed RTO and the adaptive RTO method on our Bluetooth testbed. The testbed consists of two Linux based laptops, both equipped with a Bluetooth PCMCIA card. We installed BlueZ [4], which is an open source Bluetooth Stack on the Linux operating system, on both laptops. We also used some other 802.11b devices to generate the interference to our Bluetooth connection during our experiments. We used DH5 packets in all our experiments. The system setup is shown in Fig. 2.

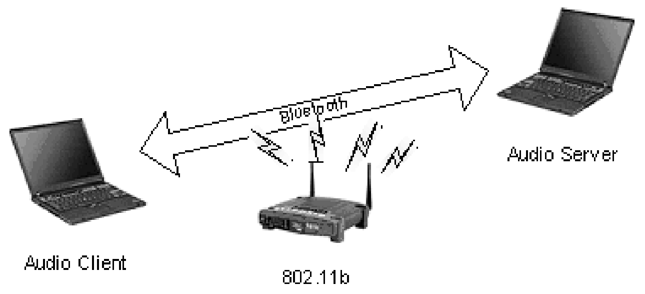

Fig. 2. Bluetooth Testbed

The streaming protocol used in our experiments is RTSP (Real-Time Streaming Protocol) [22], which is widely used on the Internet. RTSP is an application-level protocol to control the delivery of real-time multimedia data for both unicast and multicast. RTSP segments the MP3 stream into many small RTP packets at the server; the client can control the delivery (move backward, move forward, play, or stop) via the RTCP (RTP Control) protocol. Each RTP packet contains an RTP sequence number and may contain several MP3 packets.

The audio stream used in our experiments is a 128kbps bit rate, 44.1 MHz frequency, Layer II MP3 file. The MP3 packet size of this music is 417 bytes, and each RTP packet contains 3 MP3 packets plus the header information (16 bytes). Thus, the RTP packet size is 417*3+16=1267 bytes, and the T_{packet}, the time interval between two RTP packets sent on the server side, is 1267*8/128000 ≈ 80 msec. Since we use DH5 packets which have a maximum payload size of 339 bytes, each RTP packet will need at least 4 DH5 packets to be transmitted; therefore, the minimum transmission time for each RTP packets is 0.625*(5+1)*4 ≈ 15msec, where 0.625 msec is the Bluetooth slot time, (each DH5 packet consumes 5 time slot in one direction and 1 in the opposite direction).

BlueZ is an open-source implementation of the Bluetooth stack on the Linux operating system. In the Bluetooth stack (shown in Fig. 3), the HCI (Host Controller Interface) layer provides a command interface to communicate, access, and control the hardware layer. The L2CAP (Logical Link Control and Adaptation Layer Protocol) layer provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability, segmentation and reassembly operation. The BNEP (Bluetooth Network Encapsulation Protocol) [3] layer lies on top of the L2CAP layer and provides support for IP.

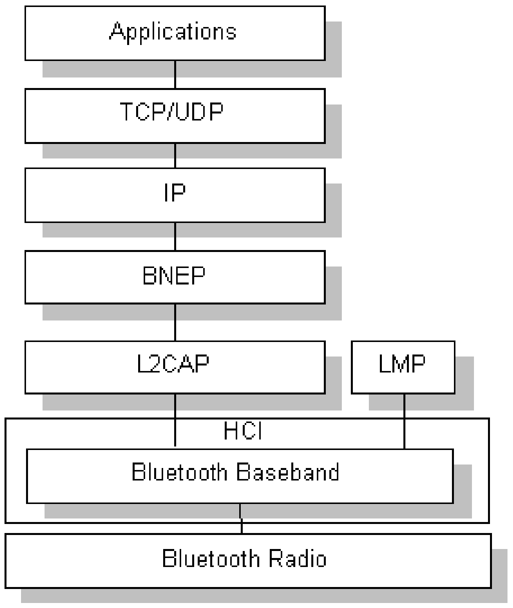

Fig. 3. Bluetooth Stack

To make the link layer application-aware, we extract header information in the BNEP layer from the received RTP packet. The BNEP layer passes the information downward the Bluetooth stack to the L2CAP layer and then the HCI layer. Each RTP packet is split into multiple fragments by the L2CAP layer. Once the fragments of the RTP packet arrive at the HCI layer, it queues all the fragments and stores the arrival time of the first fragment. The measured RTT (Round Trip Time) is the time for all the fragments of the RTP packet to be successfully transmitted. In case one fragment of the RTP packet is dropped due to the RTO being exceeded, the remaining fragments of the RTP packet are removed from the HCI layer and the baseband by using the *HCI_Flush* command (which is defined in the Bluetooth specs).

C. *Generating Interference for Bluetooth Testbed*

One challenge of our testbed setup was generating reliable interference. We found that it was very difficult to control the signal strength in the testbed since environmental factors make the link quality unpredictable. We used 802.11b devices to create interference for the Bluetooth connections, because both of them operate in the 2.4 GHz frequency band. Though increasing the physical distance between the two Bluetooth devices decreased the link quality, we found that it also increased the variance of the link quality and therefore makes conditions difficult to control. We were able to control the signal strength by keeping the two Bluetooth devices very close and controlling the traffic loads on interfering 802.11b connections.

To obtain the link quality from the Bluetooth chipset, we used the *Get_Link_Quality* function call. This call is defined in the Bluetooth spec [2] as the following manner:

Get_Link_Quality: This command returns the value of the Link Quality. It returns a number between 0 and 255, with the higher value representing a better channel. Each Bluetooth module vendor will determine how to measure the link quality.

As an example, for Bluetooth cards containing CSR (Cambridge Silicon Radio [5]) chipsets, the Link Quality is calculated from the Bit Error Rate in the following manner:

If BER (Bit Error Rate) = 0, LQ (Link Quality) = 255

*If BER <= 40/40000, LQ = 255 – BER * 40000*

*If 40/40000<BER <= 4000/40000, LQ = 215 – ((BER / 32) * 40000)*

If 4000/40000<BER <= 40000/40000, LQ=105– ((BER / 256) 40000)*

Fig. 4 shows the relation of the measured link quality value versus bit error rate for the CSR chipset, and we will make use of such relation to monitor and calibrate the channel and correlate to the RTO dynamics in the following experiments.

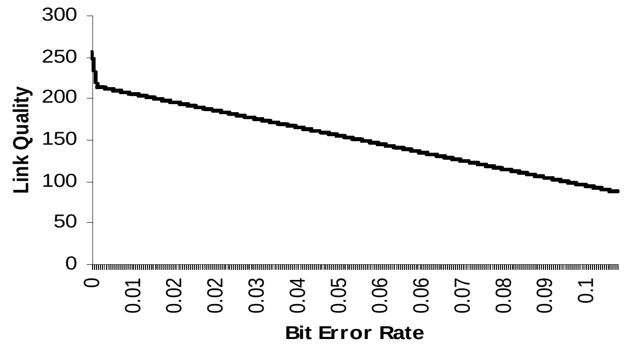

Fig. 4. Link Quality vs BER for CSR chipset

V. Simulation Results

In this section, we evaluate our proposed approach using simulation method. In the following subsection A and B, we present the simulation results showing the audio streaming throughput and packet delay in one hop Bluetooth connection (Fig. 1-a). In C, we show the results of the packet success rate of the audio streaming in both one hop and two hops Bluetooth connections (Fig. 1-a,b). Finally, we evaluate the fairness of our proposed approach by simulating two concurrent audio streams sharing one bottleneck link (Fig. 1-c). All audio streaming presented in this section was simulated using 128Kbps RTP traffic, and all Bluetooth connections were using DH5 link layer packet type. In the following subsections, we use "fixed 160, 400, 1200" to stand for the fixed RTO method with timeout values 160, 400, and 1200 msec respectively.

A. *RTP Packet Throughput*

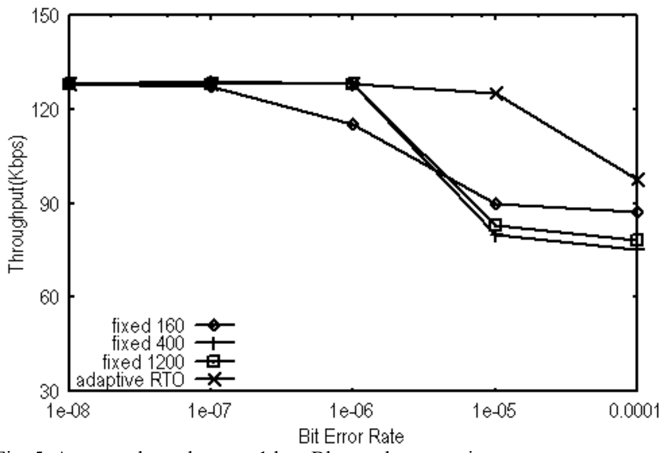

Fig. 5. Average throughput on 1 hop Bluetooth connection

Fig. 5 shows the throughput results of audio streaming in one hop Bluetooth connection with different bit error rates, where the *x* axis (bit error rate) is presented using logarithm scale. From the simulation results, it is evident that the adaptive RTO scheme always outperforms other fixed RTO schemes, especially when the bit error rate is higher than 10^{-5}. On the other hand, while the bit error rate is small (e.g. smaller than 10^{-7}), the throughput results of the adaptive RTO scheme and fixed RTO schemes are almost the same, since the data retransmissions are rate in this case.

B. *RTP Packet Delay*

Fig. 6 shows the average RTP packet delay in one hop topology with different bit error rates. From the results, the adaptive RTO scheme achieves the smallest packet delay in all the cases. While the bit error rate is as high as 0.0001, the adaptive scheme can reduce the delay by 50% and 20 % when compared with using fixed RTO as 1200 and 400 msec.

Note that, the packet delay presented in this paper is an average of packet delays from the successfully transmitted packets, regardless of those packets which are dropped in the link layer. Therefore, though fixed RTO as 160 msec can achieve almost the same average packet delay in the simulation results, it suffers more serious packet losses due to the small RTO value. As a result, the user-perceived audio quality will become degraded.

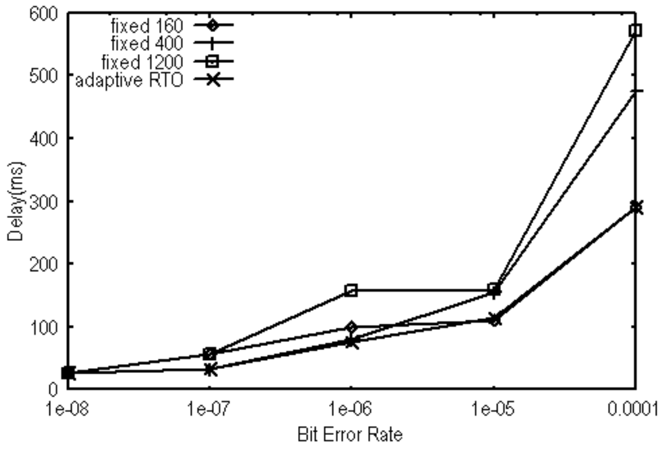

Fig. 6. Average packet delay on 1 hop Bluetooth connection

C. *RTP Packet Success Rate*

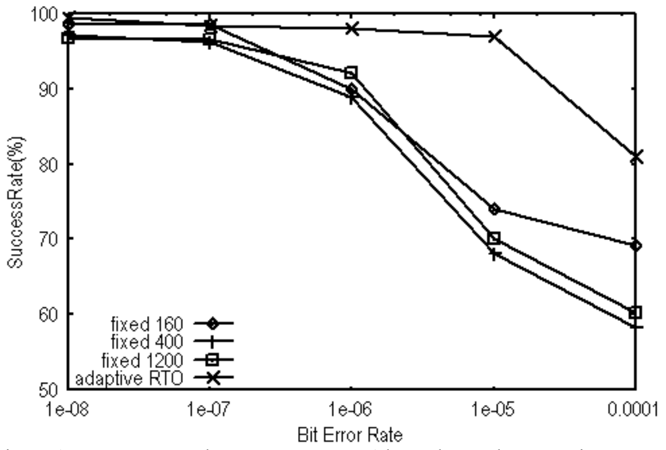

Fig. 7. Average RTP packet success rate on 1 hop Bluetooth connection

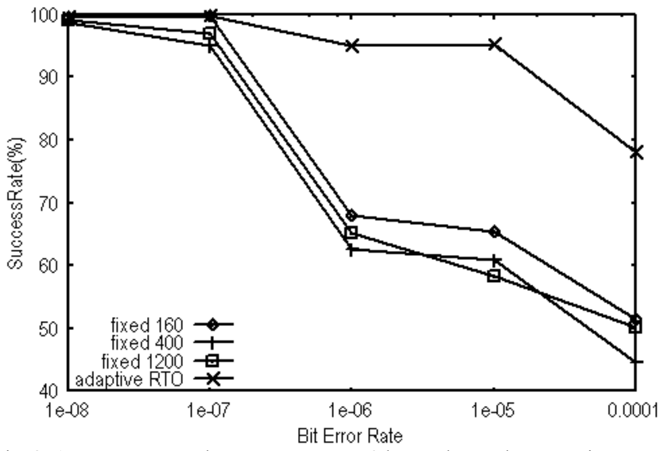

Fig. 8. Average RTP packet success rate on 2 hops Bluetooth connection

Fig. 7 and Fig. 8 show the average RTO packet success rate on 1 hop/2 hops Bluetooth connection. The packet success rate stands for the ratio of those audio packets which are successfully transmitted to the receiver. From the results, it clearly shows that the adaptive RTO scheme outperforms other fixed RTO scheme in all test cases. The performance improvement of using the adaptive scheme is more than 20% when operating on 1 hop connection with 10^{-5} bit error rate, and 40% when operating on 2 hops connection with 10^{-6} bit error rate. Additionally, while changing the Bluetooth connection from one hop to two hops, the average RTP packet success rate decreases in all schemes, especially when the bit error rate is high. However, the decrease in the adaptive RTO schemes is much less than the other fixed RTO value schemes, which means the adaptive RTO scheme is more robust with the increase of the number of connection hops.

Note that, when the bit error rate is as high as 10^{-5} (on 1 hop connection) or 10^{-6} (on 2 hops connection), the large fixed ARQ RTO scheme (e.g. fixed 1200) may not get higher packet success rate than the small fixed RTO scheme (e.g. fixed 160). This is some sort of counter-intuitive since larger ARQ RTO should provide more reliable data transmission, i.e. high success rate; however, while the error rate is high, such large ARQ RTO would require more data retransmission in the link layer, and therefore results in link layer buffer overflow if the buffer size is limited.

D. *Fairness Evaluation*

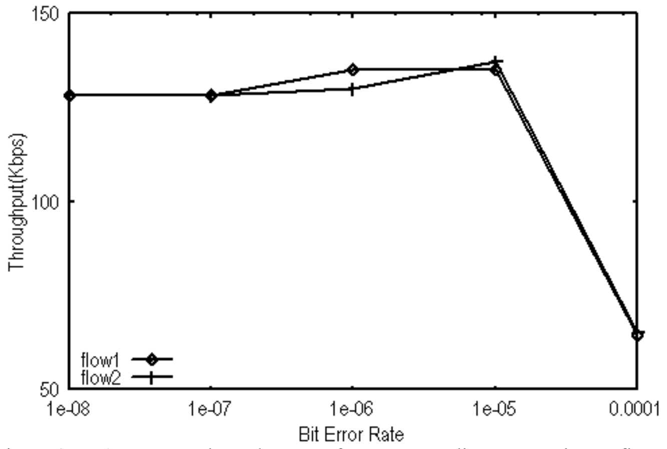

Fig. 9. Average throughput of two audio streaming flows with one shared bottleneck Bluetooth link

In the last simulation, we evaluate the fairness of the proposed adaptive ARQ RTO scheme using the topology shown in Fig. 1-c. From the results shown in Fig. 9, two audio streaming flows have very similar throughput results, i.e. they are able to fairly share the bottleneck link capacity and thus achieve fairness. More specifically, the results also indicate that the modified Bluetooth link layer not only improves the performance of audio streaming, but also keeps the fairness feature, which is an inherited property of the deployed Time Division Multiple Access (TDMA) MAC layer behavior, of Bluetooth connections.

VI. Experiment Results

In this section, we present experiment results showing the improvement in real-time audio quality when using the proposed approach. The testbed setup and implementation are described in section 5, and the experiments are repeated under different link quality conditions. In the following experiments, we use "Adapt" to stand for our adaptive scheme and "Fixed 160, 400, 1200" to stand for the fixed RTO method with timeout values 160, 400, and 1200 msec respectively. It should be noted here that in the default Bluetooth implementation, the ARQ timeout is infinite, i.e., the link layer never drops a packet.

Note that, because of the difficulty to control the link quality in the real experiments, the link quality distribution of each experiment is unique, even though the average BER might be the same. However, the average performance trend should be clearly observed.

A. Adaptive RTO

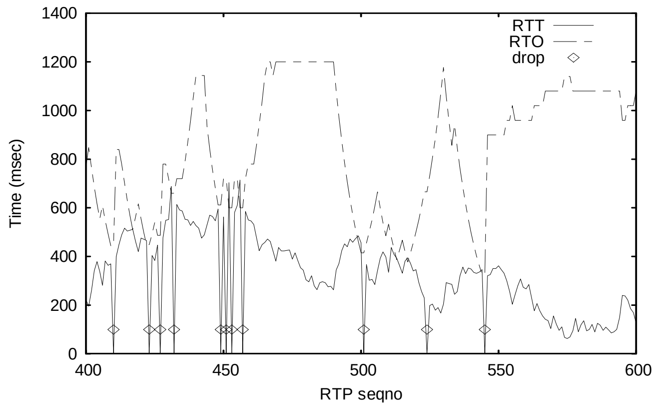

Fig. 10. RTO adaptation of the proposed approach

In the first experiment, we show the RTO adaptation behavior of our proposed approach. In Fig. 10, the solid line represents the RTT, i.e., the time between the arrival of an RTP packet at the Bluetooth baseband layer and completion of its successful transmission, and the dashed line represents the adaptive RTO value. The ARQ timeout events are marked as circles in the figure. It can be seen that the adaptive RTO value increases as the RTT decreases and vice versa, in accordance with Eq. 2.

B. RTP Packet Success Rate

Fig. 11 shows the RTP packet success rate on the receiver side, i.e. the percentage of packets successfully transmitted. Different BERs were generated by varying the load on interfering 802.11 connections, as explained earlier. Our adaptive scheme outperforms the fixed ARQ timeout schemes, clearly showing the benefit of changing timeout based on channel conditions. The "fixed 160" actually outperforms the higher ARQ timeout cases. Though this result may look counter-intuitive since a higher ARQ timeout is expected to drop fewer packets due to ARQ timeout, in reality higher ARQ timeouts also lead to a larger number of packets being dropped due to input queue overflow. This is exactly the reason why "fixed 1200" shows the least RTP packet success rate. The vanilla BlueZ link layer, which has an infinite ARQ timeout, performs similar to the "fixed 1200" timeout.

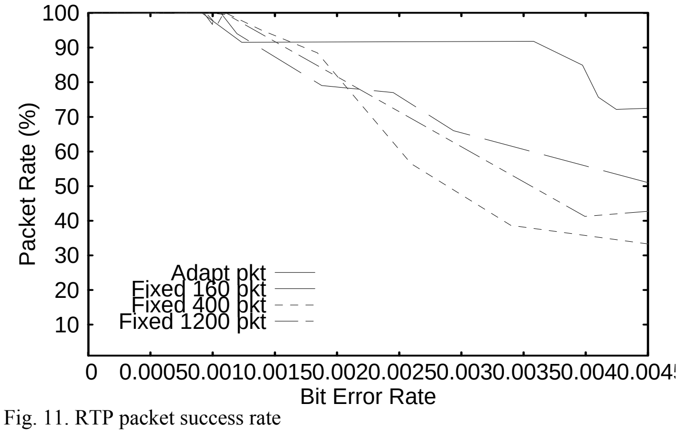

Fig. 11. RTP packet success rate

C. RTP Packet Delay

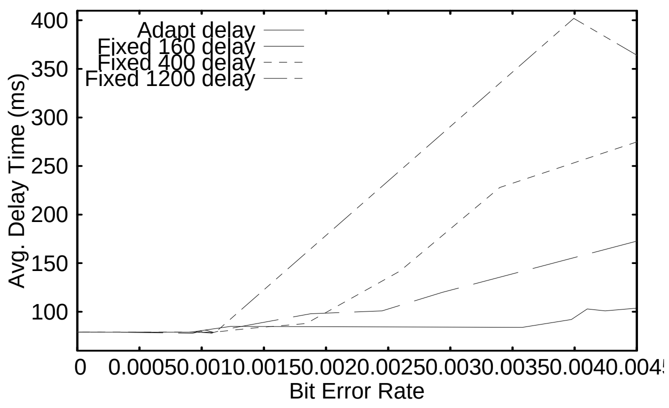

Fig. 12. RTP packet delay

Fig. 12 shows the RTP average packet delay at the audio client. While the packet success rate determines how much data is successfully transmitted, the average packet delay represents how smooth the quality is.

From the figure, it is obvious that the adaptive approach is able to achieve smaller average delays. Even with a BER of 0.0045, the average delay is still close to the minimum value, which means acceptable audio quality. For the fixed RTO cases, the higher the ARQ RTO timeout, the higher is the average packet delay. This is obvious since a larger ARQ timeout value leads to a larger number of retransmissions.

In balance, from the examination of the results in Fig. 11 and 12 one concludes that the adaptive packet scheme operate adequately with BER up to .004 (packet loss rate less than 10% and delay less than 50 ms). The "vanilla" scheme, with infinite RTO will fall apart for BER < .002. This is a remarkable improvement in performance.

VII. Conclusion

In this paper, we studied the influence of link layer behavior on streaming audio over Bluetooth. A cross-layer approach was proposed to adapt ARQ RTO value in accordance to the transmission of previous few streaming packets. We implemented a simulation model of the proposed adaptive ARQ timeout approach and created a Linux based testbed. A set of experiments was performed to evaluate the adaptive ARQ timeout scheme at the Bluetooth link layer in improving the quality of streaming audio. We compared our results with the fixed ARQ timeout methods; the results show that our method improves both average delay and the packet loss rate of RTP packets, particularly when channel conditions are bad. Moreover, the proposed approach is simple and can be easily implemented in the link layer of other wireless technologies such as 802.11b.

References

[1] Apple – iPod, http://www.apple.com/ipod/

[2] A. Balk, M. Gerla, M. Sanadidi, and D. Maggiorini "*Adaptive Video Streaming: Pre-encoded MPEG-4 with Bandwidth Scaling*", To Appear in Computer Network Journal.

[3] Bluetooth Network Encapsulation Protocol Profile ver. 1.0, http://*www.bluetooth.com*

[4] BlueZ - Official Linux Bluetooth protocol stack, http://bluez.sourceforge.net

[5] Cambridge Silicon Radio, http://*www.csr.com*

[6] M. Fainberg and D. Goodman, "Analysis of the interference between IEEE 802.11b and Bluetooth systems", VTC 2001 Fall, vol. 2, p.p. 967-971.

[7] G. Fairhurst and L. Wood, "*Advice to link designers on link Automatic Repeat reQuest (ARQ)*," *RFC 3366*, 2002.

[8] M. Handley, S. Floyd, J. Pahdye, and J. Widmer, "*TCP Friendly Rate Control (TFRC): Protocol Specification*", RFC 3448, January 2003.

[9] Helix Universal Server, http://www.realnetworks.com

[10] P. Johansson, M. Kazantzidls, R. Kapoor and M. Gerla, "*Bluetooth: An Enabler for Personal Area Networking*", Network, IEEE , Vol. 15, Issue 5 , Sept.-Oct. 2001, p.p. 28 -37.

[11] S. Krishnamachari, M. Schaar, S. Choi, X. Xu, "*Video Streaming over Wireless LANs: A Cross-layer Approach*", In Proc. of Packet Video 2003.

[12] J. Lansford, A. Stephens, R. Nevo, "*Wi-Fi (802.11b) and Bluetooth: enabling coexistence*", Network, IEEE, vol. 15, issue 5, Sept.-Oct. 2001, p.p. 20-27.

[13] Microsoft Media Server, http://www.microsoft.com/windows/windowsmedia

[14] Network Simulator (NS-2), *http://www.mash.cs.berkeley.edu/ns/*

[15] R. J. Punnoose, R. S. Tseng, and D. D. Stancil. "*Experimental Results for Interference between Bluetooth and IEEE 802.11b DSSS Systems,*" IEEE Vehicular Society Conference, October 2001.

[16] Real Player, http://www.real.com

[17] R. Rejaie, D. Estrin, and M. Handley, "*Quality Adaptation for Congestion Controlled Video Playback over the Internet,*" In Proc. ACM SIGCOMM '99, Aug.-Sep., 1999.

[18] R. Rejaie, M. Handley, and D. Estrin, "*RAP: End-to-end Rate Based Control for Real Time Streams in the Internet*," In Proc. IEEE INFOCOM'99, 1999.

[19] I. Rhee, V. Ozdemir, and Y. Yi. "*TEAR: TCP Emulation at Receivers – Flow Control for Multimedia Streaming*," Apr. 2000, NCSU Technical Report.

[20] D. Saparilla and K.W. Ross, "*Optimal Streaming of Layered Video*", In Proc. IEEE INFOCOM 2000, 2000.

[21] H. Schulzrinne, S. Casnet, R. Frederick, V. Jacobson, "*RTP: A Transport Protocol for Real-Time Applications*", RFC 1889, Jan. 1996.

[22] H. Schulzrinne, A. Rao, and R. Lanphier, "*Real Time Streaming Protocol (RTSP)*", RFC 2326, April 1998.

[23] Specification of the Bluetooth System – Core vol.1 ver. 1.1, http://www.bluetooth.com

[24] R. Wang, M. Valla, M. Y. Sanadidi, and M. Gerla, "*Adaptive Bandwidth Share Estimation in TCP Westwood*," In Proc. of IEEE Globecom 2002.

Ling-Jyh Chen received the B.Ed. degree in information and computer education from the National Taiwan Normal University in 1998 and the M.S. degree in computer science from the University of California at Los Angeles in 2002, respectively. He is currently a Ph.D. candidate at the University of California at Los Angeles. His research focuses on personal area networks, network protocols, and Internet measurements. He is a member of the Network Research Laboratory at UCLA.

Rohit Kapoor received the B.E. degree in computer science from the University of Roorkee, India in 1999 and the Ph.D. degree in computer science from the University of California at Los Angeles in 2004, respectively. His research focuses on the performance issues of Bluetooth piconets and scatternets, and Internet measurements. Since 2003, he has joined the research lab at Qualcom and focused on 3G and mobile systems.

Sewook Jung received the B.S. degree in physics from the Seoul National University, Korea in 1996 and the M.S. degree in computer engineering from the Seoul National University, Korea in 1998, respectively. He was an engineer ata Samsung Electronics from 1998 to 2003 and joined several projects for Bluetooth. He is currently a Ph.D. student at the University of California at Los Angeles. His research focuses on ad-hoc networks and personal area networks. He is a member of the Network Research Laboratory at UCLA.

M. Y. Sanadidi is an adjunct professor in the computer science department at UCLA. He received his Ph. D. in computer science from UCLA in 1982. As co-principal investigator on NSF sponsored research, he is leading research in modeling and evaluation of high performance Internet protocols. He was a manager and senior consulting engineer at Teradata/AT&T/NCR from 1991 to 1999 and led several groups responsible for performance modeling and analysis, operating systems, and parallel query optimization. From 1984 to 1991, he held the position of Computer Scientist at Citicorp where he pursued R&D projects in wireless metropolitan area data communications and other networking technologies. From 1981 to 1983, he was an assistant professor at the computer science department, University of Maryland, College Park, Maryland. His current research interests are in path characteristics estimation and its application in congestion control and adaptive streaming in heterogeneous (wired/wireless) networks.

Mario Gerla is a professor in the computer science department at UCLA. He received his graduate degree in engineering from the Politecnico di Milano in 1966, and his M.S. and Ph.D. degrees in engineering from UCLA in 1970 and 1973, respectively. He joined the faculty of the UCLA Computer Science Department in 1977. His current research is in the area of analysis, design and control of communications networks. Ongoing projects include the design and evaluation of QoS routing and multicast algorithms for IP domains, the design of wireless mobile, multimedia networks for mobile computing applications, and the development of measurement methods and tools for evaluating the performance of high-speed networks and applications.