Audio Streaming over Bluetooth: An Adaptive ARQ Timeout Approach

Ling-Jyh Chen, Rohit Kapoor, Kevin Lee, M. Y. Sanadidi, Mario Gerla
UCLA Computer Science Department, Los Angeles, CA 90095, USA
{cclljj, rohitk, kevin, medy, gerla}@cs.ucla.edu

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 l. 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 measurements. We show through 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 technology.

Keywords—Audio Streaming; Bluetooth; Adaptive ARQ; BER

1. 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.

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 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 in a Linux Bluetooth testbed and show that it 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. Section 3 discusses related work. Section 4 and 5 present our approach and the details of its implementation. Section 6 reports our testbed and Section 7 concludes the work.

2. Bluetooth Overview

Bluetooth [2] 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 [3], 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.

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.

Mode	FEC	Packet		Symmetric	Asymmetric	
		Size (bytes)	Length (slots)	Throughput (kbps)	Throughput (kbps)	
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

Table 1: Different Bluetooth ACL connection modes

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 [4] [5] [6] have shown, these sources can severely degrade Bluetooth performance.

3. Audio Streaming

In this section, we first describe some related work in the area of audio streaming, and then discuss the issues related to streaming audio over wireless.

3.1. Related Work

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, such as 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, [7][8][9] propose that the sender should dynamically adjust its sending rate by considering the media encoding, the frame rate, and the priority of media parts. [10] allows the receiver to select the most appropriate stream when the connection is setup. Moreover, recent research has focused on utilizing the available bandwidth and making the stream friendlier with other TCP flows. For example, TFRC (TCP-Friendly Rate Control) [11] deploys an end-to-end, TCP equation-based approach to enhance streaming quality. VTP (Video Transport Protocol) [12] estimates the “Eligible Rate” and adapts its sending rate according to the estimation.

However, all the above works have an implicit assumption that packet losses and delays are due to network congestion, and the delay time of packet retransmissions in the link layer of nodes 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 can introduce a delay in each packet. Such a delay can limit the applicability of end-to-end approaches, since feedback to the source may be very slow.

In order to overcome the problem, S. Krishnamachari et al proposed a cross-layer approach [13] to adaptively change the maximum number of MAC layer retransmissions and FEC encoding level in the application layer by using the estimated MAC layer link quality (SNR). However, the link quality estimation is based on the average of previous link qualities. In our experiments, we have found that it is really difficult to get accurate estimation, due to the unpredictable nature of wireless links

3.2. Audio Streaming over Wireless

Most wireless technologies perform some kind of ARQ (Automatic Retransmission Request) to ensure the integrity of the link layer. But, as the link quality becomes worse, multiple retransmissions become more and more frequent. Setting a high retransmission ARQ limit can lead 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. A low retransmission limit, on the other hand, can lead to a high percentage of packets being dropped at the link layer, which also leads to poor audio quality.

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.

4. Adaptive ARQ RTO Approach

In this section, we present our proposed approach, Adaptive ARQ RTO (retransmission timeout), and compares it to other approaches.

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 [14] 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:

$$SRTT' = (1-\gamma)\times SRTT + \gamma \times RTT \qquad (1)$$

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

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.

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

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.

Application-Aware Link Layer:

In order to be able to measure the RTT for an RTP packet, the link layer needs to be application-aware, since a vanilla link layer can only deal with Bluetooth baseband packets. The application-aware functionality required is that the Bluetooth stack should be able to identify an RTP packet. The details of how this functionality is implemented are explained in Section 5.1.

It should be noted here that a non-application-aware link layer can result in reduced performance. For instance, if one fragment of an 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. This is our motivation for adding application-awareness to our link layer.

5. Implementation

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 [15], 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 Figure 1.

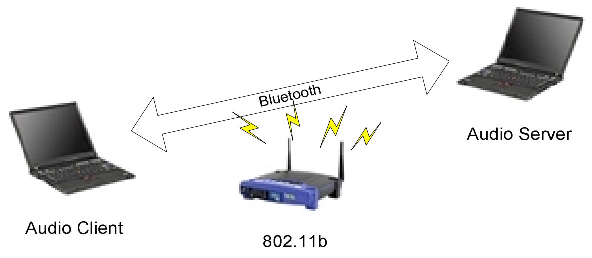

Figure 1: Bluetooth Testbed

The streaming protocol used in our experiments is RTSP (Real-Time Streaming Protocol) [16], 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).

5.1. Implementation on Bluez

Bluez is an open-source implementation of the Bluetooth stack on the Linux operating system. In the Bluetooth stack (shown in Figure 2), 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) [17] layer lies on top of the L2CAP layer and provides support for IP.

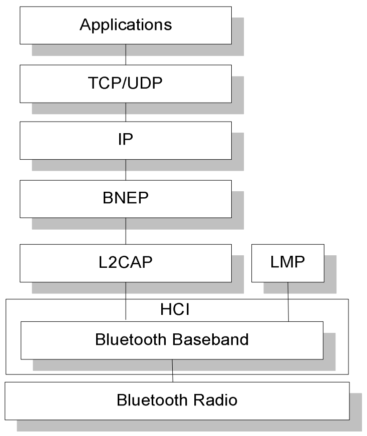

Figure 2: Bluetooth Stack

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).

5.2. Generating Interference

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 [18]) 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)*

Figure 3 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.

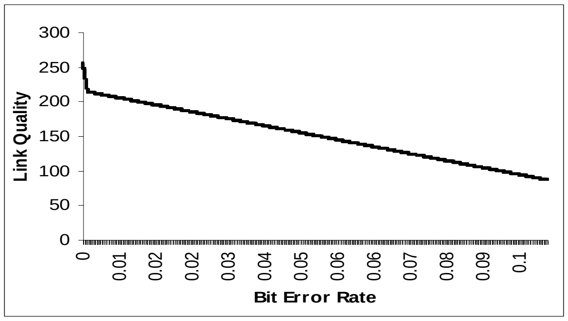

Figure 3: Link Quality vs BER for CSR chipset

6. 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.

6.1. Adaptive RTO

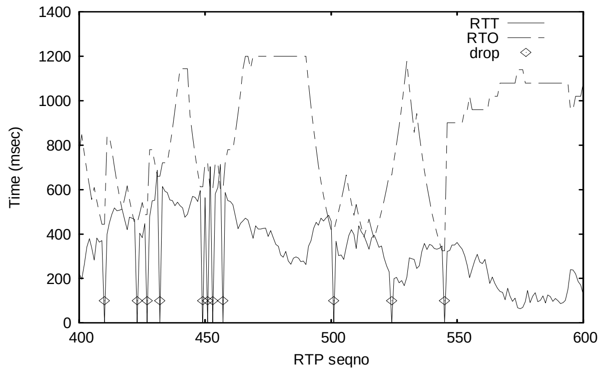

Figure 4: RTO adaptation of the proposed approach

In the first experiment, we show the RTO adaptation behavior of our proposed approach. In Fig. 4, 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.

6.2. RTP Packet Success Rate

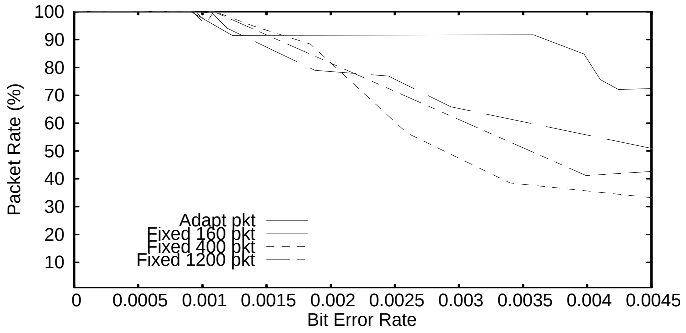

Figure 5: RTP packet success rate

Fig. 5 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.

6.3. RTP Packet Delay

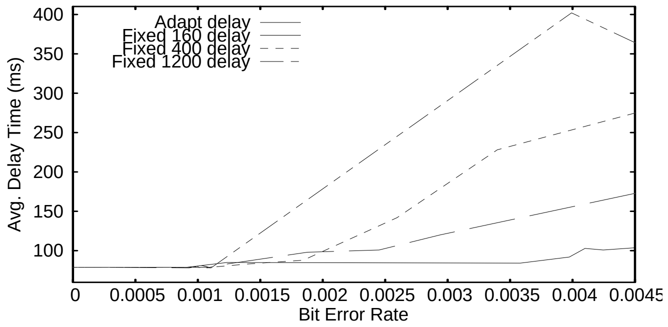

Figure 6: RTP packet delay

Figure 6 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 5 and 6 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.

7. Conclusion

In this paper, we proposed and evaluated an adaptive ARQ timeout scheme at the Bluetooth link layer to improve 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 any link layer. Moreover, the approach is not specific to Bluetooth only, but can be applied to the link layer of other wireless technologies such as 802.11b.

8. References

[1] Apple – iPod, www.apple.com/ipod/

[2] Specification of the Bluetooth System – Core vol.1 ver. 1.1, www.bluetooth.com

[3] 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

[4] 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

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

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

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

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

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

[10] Real Player, www.real.com

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

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

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

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

[15] Bluez - Official Linux Bluetooth protocol stack, bluez.sourceforge.net

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

[17] Bluetooth Network Encapsulation Protocol Profile ver. 1.0, *www.bluetooth.com*

[18] Cambridge Silicon Radio, *www.csr.com*