Dynamic Send Buffer Active Tuning for Low Latency Streaming Media

Syed Hasan, Zhiyi Huang and Paul Werstein
Department of Computer Science
University of Otago
Dunedin, New Zealand
{shasan,hzy,werstein}@cs.otago.ac.nz

Abstract

The best-effort service model of the Internet is unsuitable for streaming applications which require a smooth and flexible packet transmission rate. TCP is unable to provide such a sending rate due to its strict adherence to congestion control. We study the effect of the transport protocol's send buffer size on the performance of streaming media and propose a dynamic send buffer tuning approach which also provides congestion feedback to the application. Simulation results show that this technique improves streaming performance.

Key Words: Streaming media, Congestion Control, TCP, TCP-Friendly Rate Control (TFRC).

1 Introduction

There is a growing trend of streamed audio-video applications on the Internet. According to media research the number of streamed video increased by 38.85% in 2006 to 24.92 billion streams [1]. Faster and cheaper access bandwidth is enabling various streamed audio-video services to the end user. However, the user experience of streaming media is not satisfactory always. Recent research shows that on the Internet, about 13% home and 40% of business streaming sessions suffer quality degradation [7].

The smooth and flexible sending rate required for streaming is hard to achieve on the Internet which only provides 'best-effort' delivery of packets. In times of congestion, queues build up inside the routers delaying/dropping incoming packets. In order to mitigate this problem, streaming applications use techniques like client-side playout buffering and stream switching. The client-side playout buffer essentially borrows some current bandwidth to prefetch packets for protection against future rate reduction. The buffer size has to be large enough to ensure that in times of congestion it does not run out of packets and continues smooth playout. Stream switching allows changing the streamed bit rate depending on the amount of congestion. Despite these techniques, much of the performance issues for streaming depend on the underlying transport protocol. A transport protocol which provides a smooth low delay transmission of packets is highly desirable.

The success of the Internet can be attributed to TCP's congestion control mechanisms [10]. TCP controls the sending rate of applications in order to ensure fairness and avoid congestion. This proactive rate control is at odds with streaming applications since they cannot change or sustain a particular transmission rate whenever they want to. Unlike TCP, UDP is a fast, light weight protocol without any congestion control or retransmission functionality. This makes UDP an ideal protocol for transmitting audio-video data which can tolerate few packet losses. Applications using UDP have complete control over their sending rates and are responsible for avoiding congestion and ensuring fairness so that other flows are not starved. However, UDP is a connectionless protocol, and very often firewalls block UDP traffic for security reasons. In that case, applications revert to TCP and sometimes use HTTP on TCP to penetrate firewalls. A study shows TCP is used by 66% to 72% of all streaming sessions [7].

Realizing the need for an unreliable, congestion controlled and light weight transport protocol for a wide range of multimedia applications, a new transport protocol named Datagram Congestion Control Protocol (DCCP) [11] is being designed by the Internet Engineering Task Force (IETF). Applications using DCCP have an option to use TCP-friendly Rate Control Protocol (TFRC) [8] which provides a smoother transmission rate than TCP by slowly reacting to congestion. While there have been a significant amount of research on TFRC, most of these studies focus on TFRC's throughput variation and friendliness with TCP flows.

In this paper, we focus on the end-to-end latency of streaming applications when TFRC is used as a rate control algorithm. Latency is a denominating factor for a wide range of streaming performance measures. A low latency delivery of packets ensures that few packets are queued and allows fast switching of streaming rates when needed and makes streaming more interactive. When a sender's transmission rate is greater than the streamed bit rate, media frames are queued and delayed at the transmit/send buffer. High quality larger media frames can block low quality smaller media frames in the send buffer introducing delay in achieving the new media rate.

The focus of our paper is to tune TFRC for low latency streaming. Through our experiments, we have found that a significant portion of protocol latency comes from the send buffer. This latency can be eliminated by making simple modifications of the send buffer dynamically without changing the protocol itself. We call our modifications Dynamic Buffer Active Tuning (DBAT) approach. Previously this sort of dynamic adaptation of send buffer size has been proposed [15] for TCP to increase throughput of file transmission applications. Unlike their work, we examine the interaction of streaming applications with TFRC and focuses on reducing the latency for streaming applications.

This paper is organized as follows. Some related work is outlined in Section 2. In Section 3, the problem is illustrated and proposed modification is described in Section 4. Section 5 presents the performance results with our modification. Since this is an on-going project, Section 6 describes the current status and future work of the project.

2 Related Work

A send buffer adaptation approach has been proposed for increasing TCP throughput by Semke et al. [15]. They tune the size of the send buffer between 2*congestion window (CWND) and 4*CWND to increase the throughput of a high bandwidth-delay connection that is otherwise limited by the send buffer size. Hurley et al. [9] provides a low delay Alternative Best-Effort (ABE) differentiated service that trades high throughput for low latency. The ABE service drops packets in the network if the packets are delayed beyond their delay constraint. But this approach requires modifications in the core routers like most Differentiated Services based proposals [3]. Luo et al. presents the result of a measurement study based on large streaming media work load taken from thousands of broadband home users and business users hosted by a major ISP [7]. It shows that the median time to change to a lower bit rate stream was around 4 seconds and proposes coordinated streaming, a mechanism that coordinates client side buffering and rate adaptaton to reduce the stream switching delay. Doug [13] proposes TCP-RC, a receiver side modification of TCP which achieves low latency at the expense of reduced reliabily. Wang et al. [16] develop an analytical model for TCP based streaming and concludes that TCP generally provides a good streaming performance when the achievable TCP throughput is roughly twice the media bit rate with only a few seconds of start up delay. Krasic et al. [12] presents a framework for adaptive video streaming based on priority dropping. Chung et al. [5] develop a transport level protocol named Media Transport Protocol (MTP) which removes the burden of in order delivery from TCP. In an experimental study, Balan et al. [2] report that voice quality is not improved when TFRC is used for rate control. Goel et al. [6] propose a dynamic send buffer tuning approach where the buffer size is kept slightly larger than the TCP congestion window for TCP-based media streaming. Unlike their work, we focus on media streaming on TFRC.

3 Effect of Send Buffer Size on Streaming Performance

To handle the rate mismatch between the application's sending rate and TCP/TFRC's transmission rate, a transmission/send buffer is used. This buffer can introduce significant latency into the media stream if not tuned properly. For example, a 64KB buffer introduces a 1600ms one way delay into a 320 kbps video stream. This delay is a major obstacle for fast stream switch over as the new packets have to wait for all the old packets to be drained out. Whenever the stream switches to a lower bit rate, this situation aggravates as the queue is normally full before switching occurs. If there is congestion between the sender and receiver, the send buffer queue will continue to grow as the transport protocol is unable to send packets as quickly as it is accumulating packets from the streaming application. Flushing the send buffer before stream switching will underutilize the available bandwidth because the transport protocol has to wait for packets to be generated by the application.

The benefit of low-latency streaming is that the sending side has more control and flexibility over what data should be sent and when it can be sent. Most delay sensitive applications use a non-blocking socket so that the sending process is not blocked if the socket buffer is full. We assume that whenever the send buffer is full, the sending function returns and drops the packet. Thus a fixed fize send buffer may cause unwanted packet loss, if not tuned properly.

4 Dynamic Buffer Active Tuning

We believe that a fixed size send buffer is not optimum for low delay smooth streaming playback. It has to adapt with different states of the transport protocol. Both TCP and TFRC have to go through a slow start phase at least once in the lifetime of the connection. During slow start, the application keeps sending packets to the send buffer at a constant rate but the transport protocol starts emptying the buffer at a slow rate. For this reason, a comparatively large send buffer is required at the beginning, but when the connection reaches steady stage, we need a smaller buffer.

Our Dynamic Buffer Active Tuning (DBAT) approach automatically adjusts the send buffer size based on the state of the transport protocol. At the beginning of a streaming session, the streaming server uses packet pair based bandwidth probing techniques to select the highest streaming bit rate which is likely to be supported by the network. DBAT uses the RTT information gathered during this probing stage to calculate the buffering required during the slow start phase. During slow start, packets are queued at the send buffer until transmission rate equals the application's sending rate. Since slow start doubles the transmission rate every RTT, with the RTT information, we can calculate the time needed by slow start to achieve the streamed bit rate. By multiplying this time with the streamed packet-rate, DBAT computes the minimum buffer size required during the slow start phase.

Once slow start phase is over, DBAT tries to keep the queue size close to a threshold which is set equal to the available bandwidth-delay product. DBAT assumes that the probed bandwidth is a good approximation of the available bandwidth and uses this to compute the queue threshold. Delay is calculated based on the wighted RTT measurement.

DBAT is an active buffer tuning technique in the sense that it provides feedback to the application about its status. The application can infer incipient congestion when the send buffer queue size increases. In order to filter out transient response to queue size change, we use a weighted average on the queue size and provide feedback to the application only when the queue size crosses the queue threshold. Based on this feedback, the application may decide on the streamed bit rate much earlier.

5 Simulation Results

5.1 Experimental Setup

We use network simulator, ns-2 [14] as our preferred network simulator. Currently the standard ns-2 distribution does not have any streaming module included. However we found a streaming module named Goddard [5] which is suitable for our experiments. We integrated Goddard in ns-2 and conducted experiments using it. Goddard is based on the behaviors of Real Networks and Windows Streaming media [4]. During streaming, the Goddard server reselects the streaming bit rate in response to packet loss or rebuffering events that occur when the client playout buffer get emptied. Goddard server supports multiple levels of encoded media that are configured by giving the frame size and the frame rate for each scale level.

Goddard does not support TFRC. We modified the code so that we can use TFRC as a transport protocol for streaming. We found that the TFRC implementation in ns-2 does not have any real data transmission capability which is required by the streaming module. We changed the interface of this implementation so that data can be transmitted with each packet enabling Goddard client and server to exchange media frames. By default, the ns-2 implementation of TFRC has a infinite send buffer. We introduced a send buffer with adjustable size into TFRC. As for TCP, we modified the full-TCP implementation of ns-2 to support adjustable send buffer size. To the best of our knowledge, we are the first to conduct experiments involving the interaction of streaming application with TFRC in ns-2.

We use the dumb-bell topology for the simulation. The bottleneck link is 500 kbps with 200 ms delay. The various streaming rates supported are 80, 120, 240, 320 and 640 kbps. In all cases, one streaming flow is competing with a FTP flow. The FTP application starts at 0.1 second and stops at 100 seconds. The streaming flow starts at 30 seconds and stops at 120 seconds. If the streaming throughput matches the streamed bit rate most of the time, the user will experience smooth uninterrupted playout.

5.2 Experiment with Fixed Size Send Buffer

In this section, we illustrate how a fixed size send buffer affects streaming flows. We used fixed size drop tail send buffers. When the buffer gets full, the arriving packets are dropped until there is an empty space in the buffer. The server is unable to detect that packets are being dropped from the transport protocol's send buffer until the client sends reports back to the sever about the missing packets. Server reacts by changing the streamed bit rate if the packet loss crosses a predetermined threshold.

Figure 1 and 2 show that with TCP there is a significant difference between streamed bit rate and achieved streaming throughput. At the beginning, the server sends packet pairs to determine the available bandwidth and selects 320 kbps bit rate streaming. The achieved streaming throughput increases gradually from 1 kbps. By the time the throughput increases beyond 200 kbps, the streamed bit rate switches back to 120. Lack of feedback from the transport protocol keeps the server uninformed about the dropped packets from the send buffer queue until it gets a report from the client about the dropped packets. This is what happens in real streaming servers when they use TCP. Figure 1 uses a 64KB send buffer which is not flushed when the stream switches to a lower sending rate, and as a consequence, the streaming throughput does not reflect the streamed bit rate while the queue gets drained.

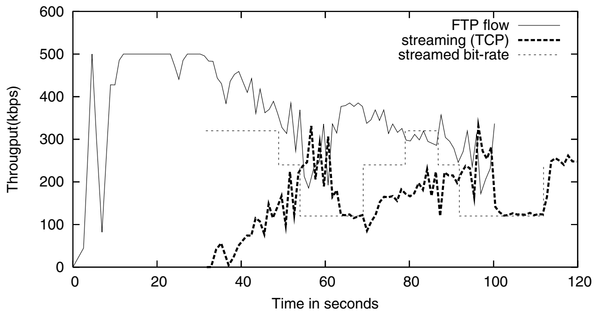

Figure 1: Streaming (TCP) with 64KB send Buffer

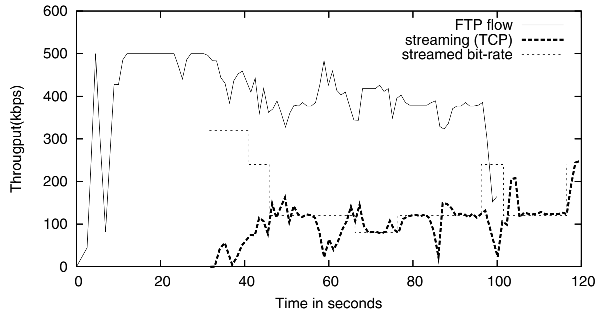

Figure 2: Streaming (TCP) with 16KB send buffer

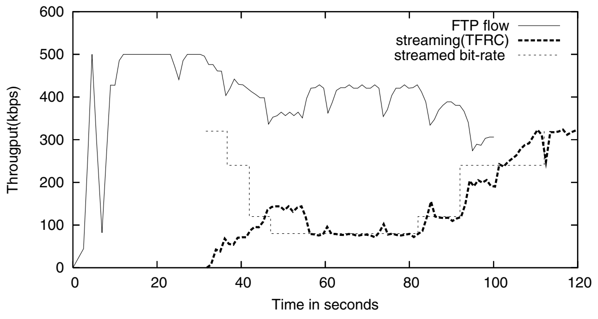

Figure 3: Streaming (TFRC) with 64KB send buffer

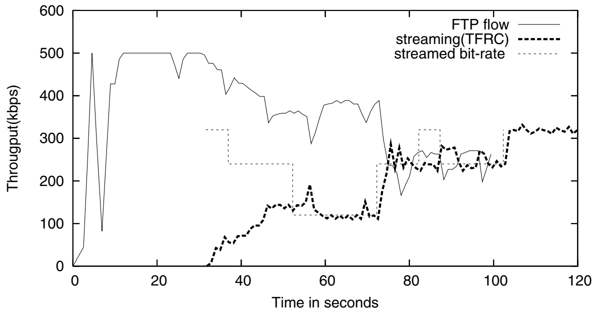

Figure 4: Streaming (TFRC) with 16KB send buffer

Figure 2 illustrates TCP with 16KB buffer size. The graph shows that even though the streaming throughput matches the streamed bit rate more closely, the throughput is very low. The lower send buffer size underutilizes the link. This is because TCP has to wait for packets to be generated by the application as soon as it has finished transmitting all the previously queued packets from the send buffer. While TCP is waiting for packets to send, other FTP flows sharing the bottleneck link grab the unused bandwidth, and the streaming flow loses its share.

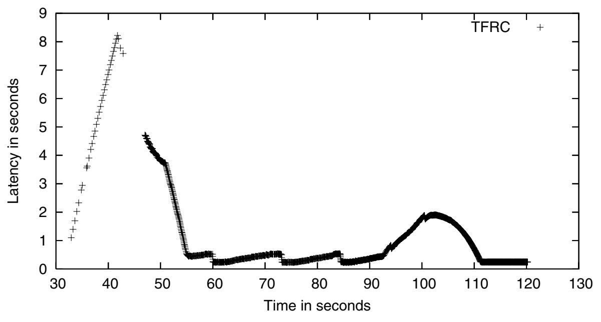

Figure 5: Streaming (TFRC) latency with 64KB send buffer

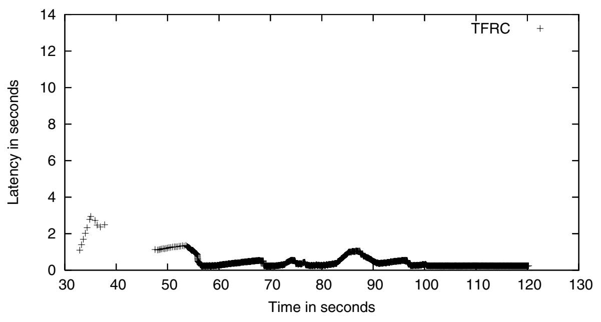

Figure 6: Streaming (TFRC) latency with 16KB send buffer

Figures 3 and 4 show streaming throughput when TFRC is used as a transport protocol. For TFRC based streaming, we have much better performance as the streaming throughput closely matches the streamed bit rate. TFRC provides a smoother sending rate than TCP by reacting to packet loss much more slowly. This gives the server time to adjust the streamed rate.

We calculate one way latency by subtracting the time when a media frame is sent by the server to the transport layer from the time when the media frame is delivered to the client side media player. So the latency includes the protocol latency as well as the link latency which is fixed at 200ms. Figures 5 and 6 present the one way latency for TFRC based streaming. Much like TCP, changing the send buffer size from 64KB to 16KB reduces the latency from 8 to 3 seconds. Just after the TCP/TFRC's slow start phase, media frames are dropped due to send buffer overflow, and the latency graph shows discontinuity around 40 seconds.

5.3 Experiment with Dynamic Buffer Active Tuning (DBAT) Send Buffer

When using our DBAT algorithm, the performance improvement for TCP and TFRC based streaming flows are shown in Figures 7, 8, 9 and 10. For TCP based streaming, there is just one stream switch over. This is because the streamed bit rate is chosen appropriately by the server which gets feedback from the transport protocol about congestion based upon send queue dynamics. The bandwidth is shared equally between the FTP flow and the streaming flow. Although there is a small variation in the streaming throughput, this small fluctuation is offset by the client side playout buffer.

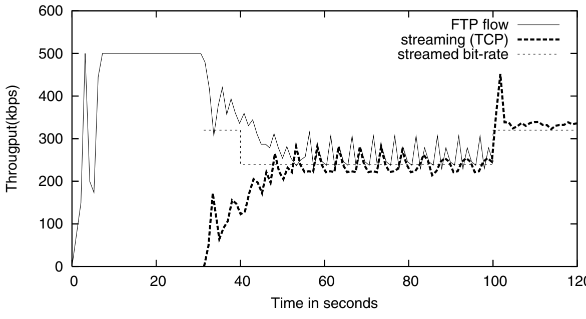

Figure 7: Streaming (TCP) with DBAT

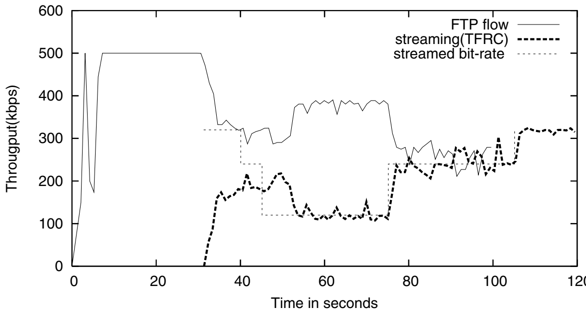

Figure 8: Streaming (TFRC) with DBAT

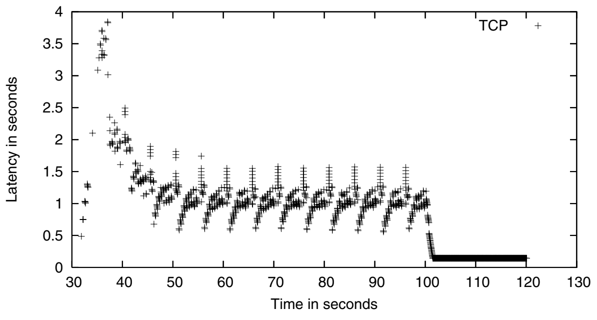

Figure 9: Streaming (TCP) latency with DBAT

Figure 8 illustrates TFRC based streaming throughput when DBAT is used. Although the throughput is smooth, it is not much different from the one without DBAT. The difference is that with DBAT the throughput reaches the streamed bit rate much faster than the one without DBAT, and this is crucial for good quality streaming. In both cases. the latency is improved as shown in Figure 9 and 10. For DBAT enabled TCP and TFRC, the maximum one way latency is 4 seconds and 2 seconds, respectively. The key thing is that DBAT reduces latency significantly without sacrificing throughput.

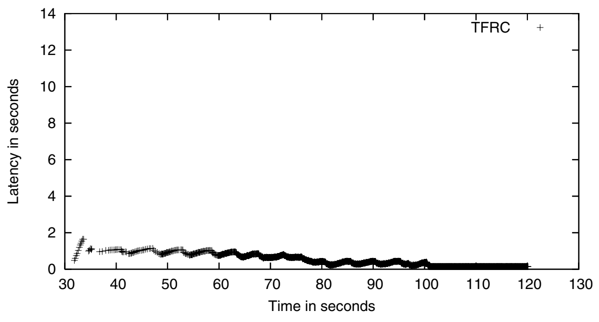

Figure 10: Streaming (TFRC) latency with DBAT

6 Future Work

Using Dynamic Buffer Active Tuning (DBAT) send buffer sizing improves the performance of streaming by reducing latency. It keeps the send buffer size at an optimum level and provides congestion feedback to the application. We conclude that although TFRC provides smoother throughput, there is room for improving TFRC when it competes with a TCP flow. This is because TFRC is less aggressive than TCP. Our on going work is improving TFRC's achievable throughput based on packet priority and more explicit feedback from the receiver. We are working also on a strategy which will couple the client side media playout buffer with the sender-side transport buffer. If the sender is informed about the receiver side buffer status, it can make better decisions in advance before the buffer runs out.

References

[1] Streaming Media Growth and Content Category Share: 2006-2010. Technical report, Accustream Research. http://www.accustreamresearch.com/.

[2] Vlad Balan, Lars Eggert, Saverio Niccolini, and Marcus Brunner. An experimental evaluation of voice quality over the Datagram Congestion Control Protocol. In *IEEE Infocom*, pages 2009–2017, Anchorage, Alaska, USA, May 2007.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang,

and W. Weiss. An Architecture for Differentiated Services. RFC 2475, December 1998, 1998.

[4] Jae Chung and Mark Claypool. Empirical evaluation of the congestion responsiveness of Real Player video streams. *Kluwer Multimedia Tools and Applications*, 31(2):171 – 193, 2006.

[5] Jae Chung, Mark Claypool, and Robert Kinicki. MTP: A streaming-friendly transport protocol. Technical report, Oregon Graduate Institute School of Science and Engineering, 2002.

[6] Ashvin Goel, Charles Krasic, Kang Li, and Jonathon Walpole. Supporting low latency TCP-based media streams. In *Tenth International Workshop on Quality of Service (IWQoS)*, pages 193– 203, Miami, May 2002.

[7] Lei Guo, Enhua Tan, Songqing Chen, Zhen Xiao, Oliver Spatscheck, and Xiaodong Zhang. Delving into internet streaming media delivery: A quality and resource utilization perspective. In *Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 217–230, Rio de Janeiro, Brazil, October 2006.

[8] Mark Handley, Sally Floyd, Jitendra Padhye, and Joerg C. Widmer. TCP Friendly Rate Control (TFRC): Protocol specification. Internet Engineering Task Force, RFC 3448, January 2003.

[9] P. Hurley, J.Y. Le Boudec, P. Thiran, and M. Kara. ABE: providing a low-delay service within best effort. *Network, IEEE*, 15(3):60–69, 2001.

[10] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM*, pages 314–329, Stanford, California, United States., 1988.

[11] Eddie Kohler, Mark Handley, and Sally Floyd. Designing DCCP: congestion control without reliability. In *ACM SIGCOMM 2006*, pages 27–38, Pisa, Italy, 2006.

[12] Charles Krasic, Jonathan Walpole, and Wu-chi Feng. Quality-adaptive media streaming by priority drop. In *13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 112 – 121, June 2003.

[13] D. McCreary, K. Li, S.A. Watterson, D.K. Lowenthal, et al. TCP-RC: a receiver-centered TCP protocol for delay-sensitive applications. In *12th SPIE Multimedia Computing and Networking Conference (MMCN)*, pages 126–130, January 2005.

[14] ns-2. The Network Simulator ns-2. http://www.isi.edu/nsnam/ns/.

[15] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. In *Proceedings of the ACM SIGCOMM'98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 315–323, 1998.

[16] Bing Wang, Jim Kurose, Prashant Shenoy, and Don Towsley. Streaming via TCP: An analytic performance study. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 908–915, New York City, NY, October 2004.