1-2000

Better-Behaved, Better-Performing Multimedia Networkign

Jae Chung
Worcester Polytechnic Institute, goos@cs.wpi.edu

Mark Claypool
Worcester Polytechnic Institute, claypool@wpi.edu

Better-Behaved, Better-Performing Multimedia Networking

Jae Chung and Mark Claypool

Computer Science Department
100 Institute Road
Worcester Polytechnic Institute
Worcester, MA 01609
Email: {goos|claypool}@cs.wpi.edu

ABSTRACT

The power and connectivity of today's Internet presents the opportunity for interactive multimedia applications across the world. However, today's Internet has been predominantly designed for TCP traffic, wherein the end hosts recognize lost packets as congestion and reduce their transmission rates appropriately. Unfortunately, TCP is not the protocol of choice for multimedia applications, because TCP's lossless transmission is stricter than required by multimedia flows and TCP adds considerable network jitter, greatly decreasing multimedia quality. UDP, the alternate transport protocol to TCP, does not respond to packet loss as a measure of congestion, often resulting in UDP flows that get an unfair share of their network bandwidth. In this work, we demonstrate that a proper network protocol can be built on top of UDP, providing well-behaved performance in the face of congestion. Moreover, we demonstrate such protocols provide far better multimedia performance than does TCP.

1. INTRODUCTION

The Internet is moving from a traditional data communication network for transferring text-based messages such as email and web traffic, to an underlying communication network for multimedia applications such as Internet phone, video conferencing and video on demand. The volume of traffic from both traditional and multimedia applications is increasing tremendously, placing renewed emphasis on congestion control and traffic fairness.

A major approach to avoiding congestion is through the use of *active queue management*, wherein a router signals impending congestion by dropping packets. The most well-known active queue management mechanism is Random Early Detection (RED). RED uses a weighted-average queue size and thresholds to detect impending congestion, and randomly drops incoming packets as the average queue size exceeds a minimum threshold [FJ93, FF97]. Active queue management requires that the end-hosts recognize dropped packets and respond by reducing their rate of transmission. In the Internet, TCP recognizes packet loss as an indicator of network congestion, and reduces transmission rate [Fl94]. To date, active queue management appears promising since by far the predominant transport protocol on the Internet is TCP [two99].

However, multimedia applications have different performance constraints than do traditional applications [CR99]. Traditional applications are very sensitive to lost packets, hence use TCP to guarantee that lost packets are retransmitted [BRS99]. Multimedia applications, on the other hand, can tolerate some data loss, but are very sensitive to variance in packet delivery, called *jitter* [CT99]. In the absence of jitter and packet loss, video frames can be played as they are received, resulting in a smooth playout, as depicted in Figure 1-top. However, in the presence of jitter, interarrival times will vary, as depicted in Figure 1-bottom. In Figure 1-bottom, the third frame arrives late at r*2*. In this scenario, the user would see the frozen image of the most recently delivered frame (frame two) until the tardy frame (frame three) arrived. The tardy frame (frame three) would then be played only briefly in order to preserve the timing for the subsequent frame (frame four). Detecting and retransmitting lost packets causes considerable jitter, making TCP unattractive to multimedia applications.

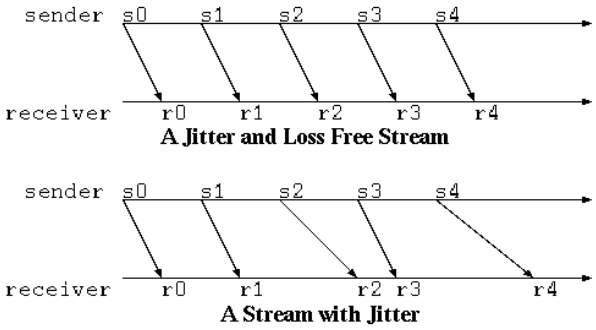


Figure 1. The above figures model packet video between sender and receiver. Each s*i* is the time at which the

sender transmits video frame *i*. Each r*i* is the time at which the receiver receives frame *i* .

Unfortunately, most non-TCP flows (often termed *misbehaved* flows) use UDP and get an unfair share of network bandwidth when there is congestion. This unfairness occurs because non-TCP flows do not reduce transmission rates while the TCP flows are forced to transmit data at their minimum rates [PSJ99]. Even worse, typical active queue management policies apply the same drop rate to each flow. This has led to new active queue management mechanisms such as Fair Random Early Drop (FRED) [LM97], that add per-active-flow accounting to RED, isolating each flow from the negative effects of others. FRED strictly punishes unresponsive or misbehaving flows to have an equal share of output bandwidth, while assuring that packets from flows that consume less than their fair share are transmitted without loss.

However, per-active-flow accounting is expensive in terms of router load and may slow down the routing speed as the number of flows increases. Even worse for multimedia applications, FRED punishes all non-TCP-like flows, which discourages multimedia applications from using any transport protocol besides TCP, despite the severe degradation TCP can cause to multimedia quality.

In this paper, we demonstrate that a well-behaved rate-based flow control mechanism can be built on top of UDP, providing significantly better quality for the multimedia application than does TCP while ensuring fairness to the TCP application. We demonstrate this for a generic rate-based streaming media application as well as for an MPEG-1 video client-server application. In addition, we demonstrate how a rate-based UDP flow can provide much better multimedia performance than does TCP.

Our design and implementation are carried out in NS, a popular Wide Area Network simulator developed at the University of California, Berkeley but used by many others for a wide variety of network research [ns2]. NS supports most of the common IP network components, including TCP (Tahoe, Reno and Vegas) and UDP transport agents, and several queue management mechanisms, including RED. Unfortunately, NS considerably lacks support for multimedia applications, only providing a basic mechanism to build Constant Bit Rate (CBR) media streams. NS does not support streaming Variable Bit Rate (VBR) multimedia, such as an MPEG client-server or Real Video. VBR applications are required for responsive multimedia applications that must maintain their strict timing constraints. Thus, a further contribution of this work is the NS-compatible source code for two flow-controlled multimedia applications[1].

The rest of this paper is laid out as follows: Section 2 describes our approach in the design and implementation of flow-controlled multimedia; Section 3 details experiments examining the behavior of our protocol and applications when running with other TCP flows; Section 3 also presents a simulation result showing that TCP is not the transport agent of choice for multimedia applications because it results in poor performance; and Section 4 summarizes our conclusions and lists some possible future work.

2. RESEARCH APPROACH

We designed and implemented two slightly different multimedia traffic generators (or multimedia applications) that respond to network congestion, extended from media scaling techniques proposed in [DHH93]. The two traffic generators have the same congestion control and avoidance (or flow control) mechanism, while the traffic they generate in response to congestion notification from the network is different. The first one, MM-APP, reduces or increases transmission rate by decreasing or increasing the transmission interval with fixed frame size. The second one, MPEG-APP, changes the transmission rate by selecting frames to transmit from an input MPEG trace file, where the frame sizes vary while the transmission interval of frames in the file are fixed in terms of frames per second.

The multimedia applications use sender and receiver behavior. Before transmitting the actual data, the sender and the receiver agree on five scale values (0 to 4), each of which is assigned to a different media encoding method and transmission policy (i.e. which frame to transmit) pair. The scale value 0 is assigned to a set from which a predetermined minimum sustainable media quality can be achieved, the next value is assigned to sets from which a better media quality can be achieved, and so on. It is assumed that the media encoding and transmission policy sets are carefully chosen so that the transmission rates resulting from the sets increase linearly as the scale value increases. Table 1 shows an example assignment.

Scale Value	Media Encoding and Transmission Policy Set	Avg. Transmission Rate (Kbps)
4	A	1100
3	B	900

[1] The source code will be available at `http://perform.wpi.edu/` by the time of publication.

2	C	700
2	D	500
1	E	300

Table 1. Example Media Scale Assignment

Thus, having five discrete and linearly increasing transmission rates assigned to the scale values, the sender starts from scale 0 transmitting at the lowest rate. The receiver detects congestion, determines the next transmission rate of the sender in terms of scale value, and notifies the sender of this scale value. The sender, being notified of the scale value, simply changes the transmission rate by using media encoding and transmission policy assigned to the scale value.

In detecting congestion, the receiver uses frame loss as the network congestion indicator. There are two circumstances where the receiver claims frame loss. The first is when the receiver gets a frame whose sequence number is greater than the expected sequence number. The second is when the receiver does not receive any frames within a timeout interval.

Proper setting of the timeout interval is critical. A timeout interval that is set too short will claim false frame losses, which will make the sender reduce the transmission rate needlessly. On the other hand, a timeout interval that is set too long will fail to detect multiple sequential frame loss effectively such that the sender reduces transmission rate later than other competing connections, which could result in an unfair portion of bandwidth. In our implementation, the Round Trip Time (RTT) of the connection is greater than the longest possible frame transmission interval:

```
RTT > Max_Intrvl: TOI = RTT

RTT ≤ Max_Intrvl: TOI = Max_Intrvl + α

(where 0 < α < RTT)
```

The receiver, when detecting congestion, reduces its scale value to half (integer division) and notifies the sender of this value by sending a small packet. When the receiver detects no network congestion within a RTT from the last checkpoint, it increases the scale value by one and notifies the sender of this value. This design of drop scale to half at congestion, and increase one scale up at a RTT is motivated by fast recovery algorithm that is found in Reno TCP implementations [FF96] and the TCP-friendly definition [FF98].

Now that the congestion control part of the multimedia traffic generators has been described, we next present the specifics of how MM_APP and MPEG_APP generate traffic associated with scales.

As briefly mentioned earlier, MM_APP directly associates transmission rates to scale values without targeting a specific media encoding and transmission policy. It assumes that every media encoding and transmission policy pair associated to the scale values generates traffic with a fixed frame size. In other words, it assumes that the transmission intervals are the only factors that cause the rate changes. Therefore, the resulting traffic can be characterized by CBR traffic of a fixed frame size and various transmission intervals associated with the scale values. Although MM_APP is not tied to a specific multimedia application, it is useful in that it is easy to change the transmission rate associated to each scale value and then test the media scaling scheme, thus eliminating the effect of a specific traffic characteristic of a particular application.

MPEG_APP, on the other hand, simulates a very specific client-server video application that is based on the MPEG-1 encoding scheme [MPFG97]. It implements five sets of MPEG-1 encoding and transmission policies and associates them with scale vales - in fact, it only changes the frame transmission policy leaving the encoding scheme unchanged. MPEG-1 encodes video at a given framerate and picture quality, generating a stream of frame types I, P and B, associated with a typical Group of Pictures (GOP), such as IBBPBBPBB. Among the three frame types, only I-frames can be decoded on their own. The decoding of a B-frame relies on a pair of I-frames and/or P-frames that come before and after the B-frame and the decoding of a P-frame relies on an I-frame or P-frame that comes before the P-frame. MPEG_APP supports MPEG-1 streams using the common GOP patterns IBBPBBPBB and IBBPBBPBBPBB. Table 2 shows the transmission policies on the two stream patterns, which is carefully selected keeping the dependencies in mind [WCK97].

Scale	Transmission Policy (Pattern 1)
4	I B B P B B P B B I
3	I B P B P B I
2	I P P I
1	I P I
0	I I

Scale	Transmission Policy (Pattern 2)

4	I B B P B B P B B P B B I
3	I B P B P B P B I
2	I P P P I
1	I P I
0	I I

Table 2. MPEG Transmission Policy Associated with Scale Values

MPEG_APP reads in an MPEG-1 trace file that contains frame information for a stream with the maximum frame rate (scale 4) as input along with the maximum frame rate and the longest possible frame transmission interval that is used for congestion detection at the receiver side. At every scale 4 transmission interval, MPEG_APP reads the frame information from the input file, and determines whether or not to transmit this frame using the current scale value and the transmission policy associated with the scale value. Figure 2(a) shows an example input file that contains frame information for a 30 frames per second IBBPBBPBB pattern stream in which the sizes of the I-, P- and B-frames are 11 KB, 8 KB and 2 KB, respectively. The frame sizes used in this example are the mean frame size of each type obtained while playing a short high quality MPEG-1 news clip. Figure 2(b) shows each transmission policy assigned to scale values with the estimated average transmission rate for the input trace file.

```
I       11000
B       2000
B       2000
P       8000
B       2000
B       2000
P       8000
B       2000
B       2000
I       11000
:       :
:       :
```

Figure 2(a). Input Trace File (bytes)

Maximum Frame Rate (Scale 4) = 30 frame/sec		
Scale	Transmission Policy (Pattern 1)	Estimated Average Trans. Rate (Kbps)
4	I B B P B B P B B I	1056
3	I B P B P B I	896
2	I P P I	736
1	I P I	544
0	I I	352

Figure 2(b). Transmission Policies and Associated Rates

Figure 3 visualizes the estimated transmission rate in Figure 2(b). The almost linearly growing estimated average transmission rates shows that the assignment of the transmission policies to the scale values works well with the given example MPEG-1 stream. This is because the linear increment of scale results in linear increment of transmission rate and the exponential decrement of scale results in exponential decrement of transmission rate.

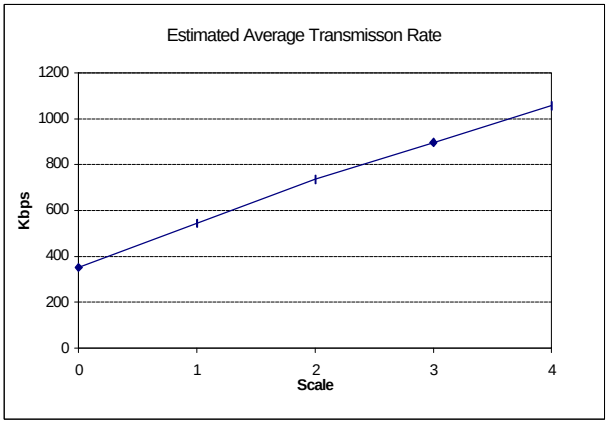


Figure 3. Estimated Average Transmission Rate of the Example in Figure 2.

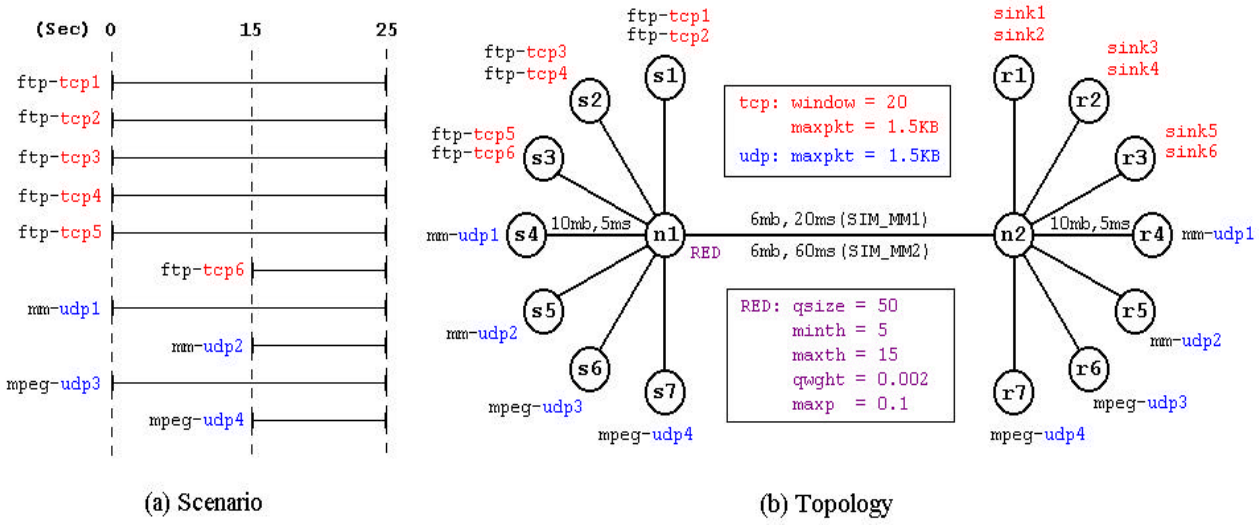


Figure 4. SIM_MM Scenario and Topology

3. EXPERIMENTS

We ran a series of simulations to validate the NS implementation of MM_APP and MPEG_APP, and to measure their fairness when competing for bandwidth with TCP flows under RED queue management. Here, we present the results of a simulation that exhibits the behavior of MM_APP and MPEG_APP. The simulation, referred to as SIM_MM, is designed to show the effect of available bandwidth and end-to-end delay on fairness. Figure 4 shows the network topology and the simulation scenario used for the simulation.

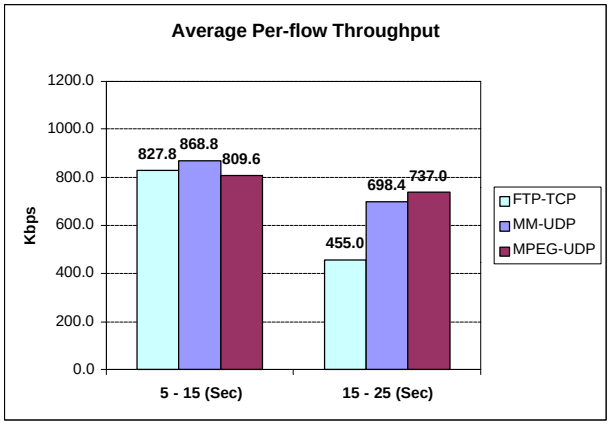
Each link that connected a source or destination node and a network node had a 10 Mbps link capacity and 5 ms of delay. The link that connected the two network nodes has 6 Mbps of link capacity and 20ms of delay. The network node n1 used RED queue management, for which the parameter set used was chosen from one of the sets that are recommended by Floyd and Jacobson [RK99].

For traffic sources, 6 FTP, 2 MM_APP and 2 MPEG_APP traffic generators were used, where FTP used TCP Reno and the others used UDP as the underlying transport agent. All the TCP agents were set to have a maximum congestion window size of 20 packets and maximum packet size of 1500 bytes. The UDP agents were also set to have maximum packet size of 1500 bytes. The MM_APP traffic generators used the transmission rates shown in Table 1, that is 300, 500, 700, 900, 1100 Kbps, for scale 0 to 4 transmission rates, respectively. The MPEG_APP traffic generators used the transmission policies and the trace file shown in Figure 2, which generated traffic rates of 352 Kbps to 1056 Kbps.

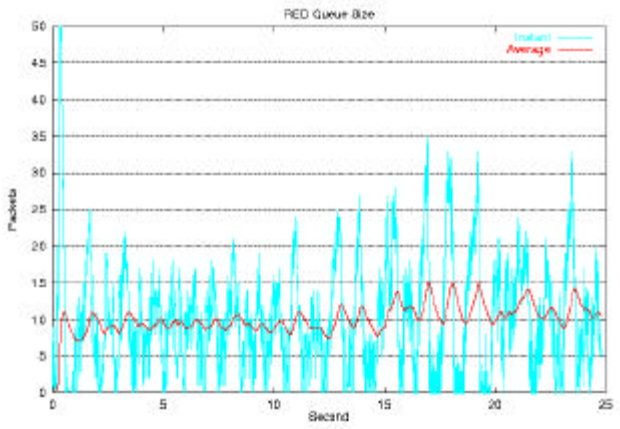
Both simulations started with five FTP applications, 1 MM_APP and 1 MPEG_APP, and after 15 seconds the remaining traffic sources joined. For the first 15 seconds, the available bandwidth share for each connection was about 857 Kbps, and for the next 10 seconds, the share went down to 600 Kbps. Figure 5 shows the results of SIM_MM. For the throughput measurements, we omitted the first 5 seconds to eliminate the startup effect of unstable TCP and RED behaviors on the fairness at the initial stage.

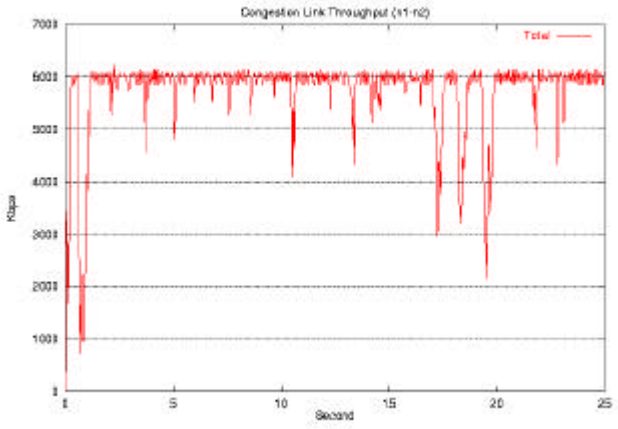
Figure 5. SIM_MM – Simulation Test of MM_APP and MPEG_APP

This simulation is set to have the FTP applications over TCP and the flow-controlled multimedia applications to fairly share the output bandwidth for the first 10 seconds (5 to 15 seconds). For the next 10 seconds when three more sources join decreasing the available bandwidth.



	5 – 15 (Sec)	15 – 25 (Sec)
FTP-TCP1	733.2	505.2
FTP-TCP2	888.0	464.4
FTP-TCP3	717.6	552.0
FTP-TCP4	958.8	412.8
FTP-TCP5	841.2	428.4
FTP-TCP6		367.2
MM-UDP1	868.8	669.6
MM-UDP2		727.8
MPEG-UDP1	809.6	826.8
MPEG-UDP2		647.2
Fair Share	857.1	600.0





In SIM_MM, one can see that all types of traffic receive a fair share of network bandwidth. Even during the second part of the simulation, seconds 15-25, the multimedia streams do become noticeably greedier, but still do not completely starve out the TCP flows. The existing unfairness is mainly caused by the in average reduction of TCP's congestion window size, and the widely fluctuating and higher RED's average queue length. When there is extreme congestion, RED drops packets from all flows, and the TCP flow's window-based mechanisms will only transmit packets on timeout, while the rate-based mechanism of MM_APP and MPEG_APP, will transmit more frequently.

SIM_MM shows that multimedia applications that use a rate-based flow control mechanism can be greedier than TCP agents that use window-based flow control mechanism under conditions of extreme congestion. Despite this drawback, there are very strong reasons that interactive multimedia applications today do not want to use TCP as underlying transport agent.

Figure 6 that shows the TCP and MM_APP jitter, as measured by the inter-packet arrival time. TCP's bursty transmission policy that transmits data packets up to the minimum of congestion and receiver side window at a time without using any particular transmission scheduling introduces high jitter compared to the continuous transmission policy. Furthermore, from time to time, TCP's timeouts add huge jitter peaks. Delay sensitive interactive multimedia application may not stand for the high jitter, since they need to play out the multimedia data at a specific time interval. They could use relatively large buffers to ameliorate the effect of jitter, however this gives extra delays that might not satisfy the users of the application. In addition, the fact that TCP does not separate flow control from loss recovery discourages multimedia applications from using it [BRS99], since this gives possibly unbounded transmission delays for a multimedia packet that is useless after a specific period due to the retransmission attempts for the previous packets.

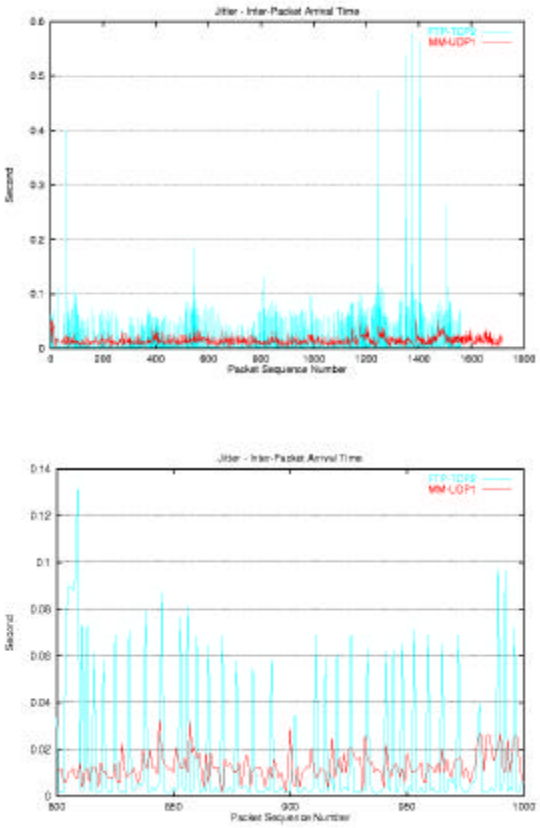


Figure 6. FTP-TCP and MM-UDP Jitter (Inter-Packet Arrival Time)

4. CONCLUSION

In this paper, we have presented the design and evaluation of flow-controlled multimedia over UDP. Our well-behaved applications use a rate-based flow control mechanism based on media scaling, but still result in fair bandwidth allocation, as does TCP. We also demonstrated that TCP is not the transport agent of choice for multimedia applications by showing its poor performance on jitter and delay that is due to its transmission policy.

There exist many possible areas for future work, including building NS applications that support other media encoding types, such as H.261 or MPEG2. Implementing MM_APP and MPEG_APP in code and experimenting on the real Internet may provide additional insight as to application behavior. User studies carefully measuring the impact of flow-controlled multimedia on top of UDP versus TCP may provide additional evidence as to the merits of our approach, as well as indicate additional means by which the network mechanism may be improved.

REFERENCES

[BRS99] Balakrishnan, H., Rahul, H. S. and Seshan, S., “An Integrated Congestion Management Architecture for Internet Hosts”, In *Proceedings of SIGCOMM '99*, Cambridge, MA, September 1999

[CR99] Mark Claypool and John Riedl, “End-to-End Quality in Multimedia Applications”, *Chapter 40 in*

Handbook on Multimedia Computing, CRC Press, Boca Raton, Florida, 1999

[CT99] Mark Claypool and Jonathan Tanner, "The Effects of Jitter on the Perceptual Quality of Video", *ACM Multimedia Conference*, Volume 2, Orlando, FL, October 30 - November 5, 1999

[DHH93] Delgrossi, L., Halstrick, C., Hehmann, D., Herrtwich, R. G., Krone, O., Sandvoss, J. and Vogt, C., "Media Scaling for Audiovisual Communication with the Heidelberg Transport System", In *Proceedings of the Conference on Multimedia '93*, Anaheim, CA, August 1993

[FF96] Floyd, S. and Fall, K., "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *Computer Communication Review*, V.26 N.3, July 1996, pp. 5-21

[FF97] Floyd, S. and Fall, K., "NS Simulator Tests for Random Early Detection (RED) Queue Management", *Lawrence Berkeley Laboratory*, One Cyclotron Road, Berkeley, CA 94704, April 29, 1997

[FF98] Floyd, S. and Fall, K., "Promoting the Use of End-to-End Congestion Control in the Internet", *IEEE/ACM Transactions on Networking*, February 1998

[FJ93] Floyd, S. and Jacobson, V., "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, August 1993

[Fl94] Floyd, S., "TCP and Explicit Congestion Notification", Computer Communication Review, October 1994

[MPFG97] Mitchell, J.L., Pennebaker, W.B., Fogg, C.E. and LeGall, D.J., "MPEG Video Compression Standard", In *Digital Multimedia Standards Series*, Chapman & Hall, New York, NY, 1997

[ns2] VINT, "Virtual InterNetwork Testbed, A Collaboration among USC/ISI, Xerox PARC, LBNL, and UCB", URL: http://netweb.usc.edu/vint/

[RK99] Raghavendra, A. M. and Kinicki, R. E., "A Simulation Performance Study of TCP Vegas and Random Early Detection", *IPCCC99*, February 1999

[two99] Traffic Workload Overview, caida, 1999 [Online] at http://www.caida.org/Learn/Flow/tcpudp.-html

[WKC97] Walpole, J., Koster R., Cen, S., Cowan, C., Maier, D., McNamee, D., Pu, C., Steere, D. and Yu, L., "A Player for Adaptive MPEG Video Streaming over the Internet", In *Proceedings 26th Applied Imagery Pattern Recognition Workshop AIPR-97*, SPIE, Washington DC, October 1997