Traffic Sensitive Active Queue Management

Mark Claypool, Robert Kinicki, and Abhishek Kumar
{claypool|rek}@cs.wpi.edu
Computer Science Dept at Worcester Polytechnic Institute
100 Institute Road, Worcester, MA 01609, USA

Abstract— **Delay sensitive applications, such as voice over IP and network games, often sacrifice throughput for lower delay to obtain better quality. Unfortunately, the Internet does not allow an application to choose the amount of delay or throughput it receives and instead packets from all applications receive the same best-effort service. This paper presents a new QoS mechanism called the Traffic Sensitive Quality of Service controller (TSQ) that provides better delay performance for delay sensitive applications and higher throughput for throughput sensitive applications. Also contributed are quality metrics for some typical Internet applications that can be used by an application to adapt its delay hints and evaluate QoS based on current Internet traffic conditions. Experiments suggest TSQ's benefits to performance along with retention of the current best-effort Internet environment without complicated traffic monitoring or policing.**

I. Introduction

Emerging applications such as IP telephony, video conferencing and networked games, unlike traditional applications, have more stringent delay constraints than loss constraints. Moreover, with the use of repair techniques [1], [21], [17] packet losses can be partially or fully concealed, enabling multimedia applications to operate over a wide range of losses and leaving end-to-end delays as the major impediment to acceptable quality. Unfortunately, the current Internet does not support per application Quality of Service (QoS).

ABE [13] provides a queue management mechanism for low delay traffic. ABE allows delay-sensitive applications to sacrifice throughput for lower delays by rigidly classifying all applications as either delay-sensitive or throughput-sensitive. Thus applications cannot choose relative degrees of sensitivity to throughput and delay. Approaches such as CBT [20] and [18] provide class-based and bitrate guarantees for different classes. However, these fixed and pre-determined classes are not sufficient for representing the varying QoS requirements of applications within one particular class. Similarly, DCBT with ChIPS [2], which extends CBT by providing dynamic thresholds and lower jitter for multimedia traffic, still limits all multimedia traffic to the same QoS.

DiffServ approaches, such as Assured Forwarding (AF) [11] and Expedited Forward (EF) [15], give differentiated service to traffic aggregates. Unfortunately, the DiffServ architecture is very complicated and requires traffic monitors, markers, classifiers, traffic shapers and droppers to provide QoS. IntServ [23] provides per flow QoS guarantees, but requires complex signaling and reservations by all routers along a connection on a per-flow basis, making scalability difficult for global deployment. Liao and Campbell[16] propose a related approach using application utility functions and apriori adaptation scripts whereby wireless applications can adapt to network capacity changes reported via network layer probes. However, this scheme does not consider the delay-sensitivity demands that are important for the quality of interactive applications.

This paper introduces a new Internet QoS mechanism, the Traffic Sensitive QoS Controller (TSQ), that provides a congested Internet router with per packet QoS support based on an application's delay sensitivity. Unlike approaches that provide fixed service classes, with TSQ each application chooses a customized delay-throughput trade-off based on its own requirements and correspondingly marks each packet with a *delay hint* indicating the relative importance of delay versus throughput. On receipt of each packet, the TSQ router examines the delay hint and calculates an appropriate queue position where the packet is to be inserted. A packet with a low delay hint is allowed to "cut-in-line" towards the front of the queue, while a packet from an application with a high delay hint is inserted towards the end of the queue. To prevent delay-sensitive applications from gaining an unfair advantage over throughput-sensitive applications, TSQ proportionately increases the drop probability of the packets inserted into the queue. The more a packet attempts to cut-in-line, the more the packet's drop probability is increased. Since throughput-sensitive applications mark their packets with high delay hints, they are not cut-in-line and so do they not have their drop probability increased. TSQ still allows Internet service to be best-effort in that it requires no per-flow state information, additional policing mechanisms, charging mechanisms or usage control.

TSQ can be used in conjunction with most AQMs that provide an aggregate drop probability, for example RED [8], Blue [7], PI [12], and SFC [10]. Performance of TSQ used in conjunction with the Proportional Integral (PI) controller AQM [12] has been evaluated with varying mixes of delay-sensitive and throughput-sensitive flows. To quantify an application's QoS, a QoS metric is proposed based on the minimum of an application's delay quality and throughput quality. Another contribution of this work is quality metrics, based on recommended application performance requirements, that cover a range of QoS and throughput sensitivities: interactive audio, interactive video and file transfer. Using TSQ, applications apply knowledge of their QoS requirements to dynamically choose their delay hints to maximize their Quality of Service. Evaluation results suggest that TSQ with PI provides better quality for all applications than PI by itself.

Section II presents quality metrics devised for fundamental Internet applications; Section III discusses the TSQ mechanism; Section IV describes experiments and analyzes the performance of TSQ; and Section V summarizes our work

II. Application Quality Metrics

Utilizing previous work [6], [9], [14], [25], we have devised quality functions for several[1] applications in terms of their net-

[1] Because of space constraints, quality functions for video conferences are omitted here, but can be found in [3].

work delay and the network throughput called the *delay quality* (Q_d) and *throughput quality* (Q_t), respectively. Overall application quality is defined as the minimum of these two metrics:

$$Q(d,T) = min(Q_d(d), Q_t(T)) \tag{1}$$

$Q(d,T)$ is a normalized metric such that a value of 1 represents the maximum quality that the application can receive and a quality of 0 represents performance that is of no use to the application.

A. Audio Conference Quality

Audio conferences are relatively sensitive to increased delays but less sensitive to reduced throughput. [9] suggests that audio conference quality is effectively impacted by three ranges of delay: one-way delays of 150 ms or less means excellent quality; one-way delays of 150-400 ms means good quality; and one-way delays in excess of 400 ms means poor quality. Mean Opinion Score (MOS) testing of audio conference conversations in [14] suggests one-way delays above 525 ms means unacceptable quality.

Figure 1 (left) utilizes these results to graph the delay quality of an audio conference versus one-way delays. The highest quality of 1 (equivalent to a MOS of 5) occurs when there is no delay. The audio application has excellent quality when the one way delay is 150 ms or smaller. As delay increases, the initial decrease in quality is not significant, and a delay of 150 ms provides the application with a quality of 0.98. However, as the delay increases above 150 ms, the drop in quality is steep, with a delay of 300 ms reducing quality to 0.7 (equivalent to a MOS score of 3.5) and to 0.5 (equivalent to a MOS score of 3) when the delay is 400 ms. When the delay is higher than 400 ms, we propose that the degradation slope is about twice the degradation slope in quality from 150 to 400 ms delay. Thus, the graph visually represents the three broad quality ranges described in [9] and corresponding to MOS scores in [14], while also providing quantitative quality metrics for intermediate one-way delays. The set of equations governing the delay quality of an audio conference is as follows:

$$\begin{array}{rcl} Q_d(d) = & -0.00133 \times d + 1 & d \leq 150 \\ Q_d(d) = & -0.00192 \times d + 1.268 & 150 \leq d \leq 400 \\ Q_d(d) = & -0.004 \times d + 2.1 & 400 \leq d \leq 525 \\ Q_d(d) = & 0 & 525 \leq d \end{array}$$

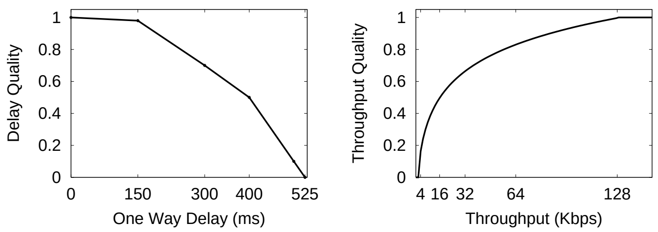

Fig. 1. Delay Quality (Left) and Throughput Quality (Right) for an Audio Conference

Figure 1 (right) shows a quality curve for an audio conference as a function of the throughput that it receives (the *throughput quality*). The audio conference throughput is given a quality of 1 when the throughput is 128 Kbps since this data rate can provide CD quality audio. The throughput quality decreases linearly as the throughput is halved since every time one fewer bit is used for audio encoding, the throughput of the audio codec is reduced by half. Hence audio quality versus throughput follows a logarithmic curve, where a reduction in throughput above 64 Kbps does not greatly reduce quality, while a reduction in throughput below 64 Kbps does. The throughput quality is 1 for 128 Kbps throughput, decreases to 0.83 for 64 Kbps and falls to 0 when the throughput is 2 Kbps. This is appropriate because 4 Kbps is the lowest codec rate available for typical audio applications [5]. The set of equations for the throughput quality of an audio conference is as follows:

$$\begin{array}{rcl} Q_t(T) = & 1 & 128 \leq T \\ Q_t(T) = & 0.24045 \times log(T) - 0.17 & 4 \leq T \leq 128 \\ Q_t(T) = & 0 & T \leq 4 \end{array}$$

B. File Transfer Quality

File transfer applications, unlike the audio and video conferences, are not delay sensitive (relative to router queuing delays). Instead, the quality of these applications is almost entirely dependent on their throughput. A file transfer application's quality will degrade only if the delay increases on the order of tens of seconds, which is well beyond the scope of router queuing delays. Thus, the delay quality of a file transfer application is as follows:

$$Q_d(d) = \quad 1 \quad d \leq 10000$$

Since, in our experiments, delays are all less than 1000 ms, the quality of file transfer application is unaffected by router queuing delays.

The quality of a file transfer application depends almost entirely on the throughput that it can get from the network. In our quality metrics, the quality requirements of a file transfer are dependent upon the size of the file that it is transferring. A small file will require a lower throughput to attain good quality as compared to a very large file. We propose that a file transfer application has maximum quality if it can finish transferring a file in 1 second. Thus for a 10 Mb file, a quality of 1 is attained from a throughput of 10 Mbps. If the throughput obtained is greater than 10 Mbps, the quality does not improve, while a decrease in quality is directly proportional to a decrease in throughput. For a smaller file of 10 Kb, the required throughput for a quality of 1 is only 10 Kbps. We derive the following equation for throughput quality of file transfer applications:

$$Q_t(T,S) = \frac{T}{S}$$

where S is the size of the file transferred.

III. Traffic Sensitive QoS Controller

The Traffic Sensitive QoS controller (TSQ) provides Quality of Service when used in conjunction with most existing Active Queue Management (AQM) mechanisms. TSQ accommodates delay-sensitive applications, such as interactive multimedia, by providing a low queuing delay, while at the same time not penalizing the throughput of traditional throughput-sensitive applications, such as file transfers. TSQ achieves this per-application

QoS by providing a trade-off between queuing delays and drop probabilities. Applications inform TSQ about their delay sensitivity by providing a *delay hint* (see Section III-A). A TSQ-enabled router then provides packets with a low delay hint a lower delay by using a "cut-in-line" mechanism (see Section III-B). In order to avoid penalizing throughput-sensitive applications, TSQ adjusts the drop probability of a delay-sensitive packets based on the reduction in delay it provides to the packet (Section III-C). Figure 2 summarizes the TSQ algorithm at the end of this section.

A. Delay Hints

Applications wanting to use the benefits of TSQ need to provide the router with a measure of their sensitivity to delay. This is done by providing a *delay hint* (d) in the header of each IP packet, where a low delay hint means that the application requires a low network delay for good quality and a high delay hint means that the application is more throughput-sensitive and does not require a low delay for good quality. Applications such as interactive multimedia and network games will typically provide low delay hints, while applications such as file transfer will typically provide the highest delay hints.

Based on the discussion in [24], there are 4 to 17 bits available in the IP header that can be used to carry delay hints. In our current implementation of TSQ, we use a 4 bit delay hint,[2] providing a range of delay hints from 1 to 16. Thus, an application which is extremely delay-sensitive will choose a delay hint of 1, in contrast to an application which can tolerate some delay that chooses the maximum delay hint of 16.

B. Cut-in-Line

Typically routers use a FIFO queue to hold packets. Since all packets are enqueued at the end of the queue, all packets, and therefore all applications, receive the same queuing delay. The queuing delay obtained by each packet depends upon the current queue length (q) and the outgoing link capacity. TSQ provides delay-sensitive packets with a lower queuing delay by "cutting" packets in line according to their delay hints. A packet from a delay-sensitive application with a low delay hint will generally be queued towards the front of the queue leading to a lower queuing delay for that packet. A packet from a throughput-sensitive application having a high delay hint will generally be enqueued towards the end of the queue. However queue insertion based solely on delay-hints may cause starvation of packets with high delay hints. For example, a packet with a high delay-hint at the end of the queue can be starved in the face of a large number of low delay-hint packets cutting in line at (or above) the link capacity in front of this packet.

To avoid starvation, the TSQ cut-in-line mechanism is implemented by using a weighted insertion into the queue. At the arrival time of a packet (t_a), TSQ calculates the queuing delay that the packet would experience if it was inserted at the end of the queue; we call this queuing delay the *drain time* (t_d) of the queue. TSQ calculates the packet weight (w) according to its delay hint, drain time and time of arrival at the queue:

$$w = \frac{d \times t_d}{2^n} + t_a \tag{2}$$

where n is the number of bits used to represent the delay hint (4 in our current implementation). The packets in the queue are inserted in order sorted by their weights, with lower weight packets inserted towards the front of the queue and higher weight packets inserted towards the end of the queue. The new position of the packet in the queue is referred to as q'. Thus, a high delay-hint will cause a packet to have a higher weight and hence a higher value of q' (a delay hint of 16 will cause a packet to have a $q' = q$).[3] Newly arriving packets have their weights slightly increased due to the effect of the time of arrival on their weight, thus preventing starvation of older packets. Note, too, that since the weight of the current packet includes the drain time of the queue, packets arriving after the current packet that have the same weight will always be placed behind the current packet.

This cut-in-line requires a weighted insertion that can be implemented using a probabilistic data structure such as skip lists [22], giving complexity $O(log(q))$, where q is the number of packets in the queue.

C. Drop Probability

During congestion, many AQM techniques produce an aggregate drop probability (p)) which is applied to packets arriving at the router. All arriving packets are subject to the same drop probability, with packets that are randomly dropped not being inserted in the queue. However, in the case of TSQ, a uniform drop probability for all packets will potentially result in a higher throughput for the delay-sensitive applications, since TSQ provides a lower delay to its packets. Hence, TSQ increases the drop probability for packets with delay hints lower than the maximum (2^n, or 16 in our implementation). The increase in drop probability is related to the reduction in queuing delay that the packet would otherwise experience if it were inserted in the queue in the position calculated by the cut-in-line mechanism. Thus, for a packet from a throughput-sensitive application which would otherwise be inserted at the end of the queue, the drop probability from the AQM technique is not increased, hence the application benefits from any throughput advantage provided by the underlying AQM.

To determine the appropriate drop probability of packets that have cut in line, TSQ starts with the approximate steady state throughput T of a TCP flow in which throughput is inversely proportional to the queuing delay and the square root of the loss rate [19]:

$$T = \frac{K}{r \times \sqrt{p}} \tag{3}$$

where r is the round-trip time, p is the loss rate and K is a constant for all flows based on the network conditions. The round trip delay r is the sum of the queuing delay and the round-trip propagation delay. Since some packets can have a decreased queuing delay by cutting in line, TSQ compensates by increasing the drop probability for those packets. Let the new queuing

[2]The optimal number of bits that should be used for delay hints is left as future work.

[3]To support legacy applications and incremental deployment, TSQ assumes any packet that does not provide a delay hint implicitly requests the maximum delay hint, 16.

delay after TSQ be q', the new drop probability be p', and the round-trip propagation delay be l. The throughput obtained by the flow will now be T':

$$T' = \frac{K}{(l + q') \times \sqrt{p'}} \tag{4}$$

To prevent the new throughput T' from being greater than the throughput obtained without TSQ, ($T' \leq T$), the new drop probability p' is calculated as:

$$p' = \frac{(l + q)^2 \times p}{(l + q')^2} \tag{5}$$

The value of p' depends on the new queue position value q' and the queue position q if TSQ were not present (in other words, the instantaneous queue length when the packet arrived). p' also depends on the one-way propagation delay l of the network. Since it is difficult for the router to determine the one-way propagation delay of every flow, the value of l is kept as a constant, but is typically between 40-100 ms for many Internet links [4]. Network administrators setting l to lower values in this range will result in a more aggressive increase in drop probability, while setting l to higher values in this range will result in less aggressive increase in drop probability. For our experiments, the one-way propagation delay constant is fixed for the router at 40 ms.[4]

Constants:
C - capacity of outgoing link, l - network latency
n - number of bits used for delay hints
Variables:
d - packet delay hint, p - AQM drop probability
p' - drop probability after TSQ, q - current length of queue
q' - position to inserted packet, t_a - packet arrival time
t_d - packet drain time, w - packet weight

on receiving packet pkt:

$t_d = \frac{q}{C}$ *// Calculate queue drain time*

$w = \frac{d \times t_d}{2^n} + t_a$ *// Calculate packet weight*

q' = weightedInsert(w,pkt) *// Calculate position in queue*

$p' = \frac{(l+q)^2 \times p}{(l+q')^2}$ *// Calculate drop probability*

if (uniform[0,1] $\leq p'$) **then** drop(pkt)
else insertPacket(q', pkt)

Fig. 2. TSQ Algorithm

IV. Experiments

We implemented TSQ over an existing Active Queue Management (AQM) technique, the PI-controller [12] (or just PI for short). PI attempts to provide a steady queuing delay by keeping the queue size stable around a target queue length, adjusting the drop probability in response to the rate of incoming packets in order to meet that target. Like many AQMs, PI provides an explicit aggregate drop probability required for TSQ.

All implementation and experiments were done in the Network Simulator (NS-2).[5] Figure 3 shows the generic network

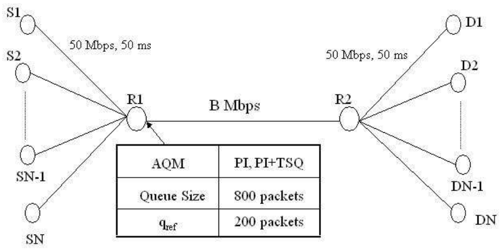

Fig. 3. Network Topology.

topology for all experiments. There are N sources S1...SN and N destinations D1...DN. The N flows are connected to a single common link giving rise to a bottleneck at router R1. Each of the connections between the sources and the bottleneck router have a link capacity of 50 Mbps and a propagation delay of 50 ms. Similar connections exist between the egress router (R2) and the destinations. The bottleneck link capacity is B Mbps. The one-way propagation delay of the network is D ms. The bottleneck router runs PI [12] plus our implementation of the TSQ algorithm in Figure 2. PI is configured with the values recommended in [12]: $\alpha = 0.00001822$, $\beta = 0.00001816$, $\omega = 170$, q_{ref} = 200 packets and q_{max} = 800 packets. The packet size is 1000 bytes.

A. Audio Quality Evaluation

In this experiment the performance of a single interactive audio flow sharing the network with other TCP based bulk file transfer flows is evaluated. Details on the performance of a videoconference flow, an application that is both delay and throughput sensitive, are omitted here due to lack of space, but can be found in [3].

The bottleneck link capacity $B = 15$ Mbps and the one-way propagation delay $D = 50$ ms providing one-way propagation delays between each of the sources and their respective destinations at 150 ms. The number of flows $N = 100$, with 99 TCP based FTP bulk transfer flows that are not delay sensitive and so provide the maximum delay hint of 16, and 1 audio conference flow simulated as a TCP-friendly[6] source sending data at a rate of 128 Kbps. The experiment is run for 100 seconds of simulation time, whereupon the delay hint of the audio flow is changed for the next run in order to evaluate the performance of the audio flow over a range of delay hints.

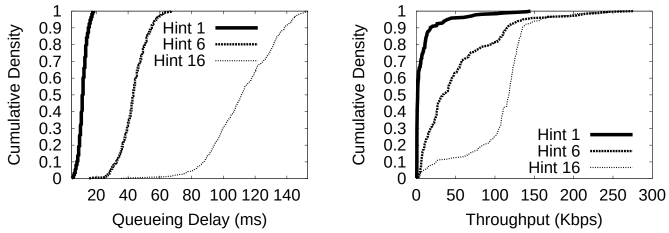

Fig. 4. CDF of Queuing Delay (left) and Throughput (right) Audio Conference Flow with Delay Hints of 1, 6 and 16.

Figure 4 (left) depicts a cumulative density function (CDF)

[4]Note that this value is fixed for the TSQ router for all experiments although the flows may have different propagation delays.
[5]http://www.isi.edu/nsnam/ns/

[6]A flow is *TCP-Friendly* if its data rate does not exceed the maximum data rate from a conformant TCP connection under equivalent network conditions.

of the queuing delay experienced by the audio flow for 3 different delay hints. The CDF is plotted for a delay hint 1, which gives the minimum delay, a delay hint 6, which gives the audio flow its optimal quality, and a delay hint 16, which gives the maximum delay. The median queuing delay is lower for the lower delay hints, and the CDF curves for hints 1 and 6 are steeper than for hint 16, which implies that there is less variation in the per-packet queuing delay with lower hints. Hence, for delay sensitive applications an AQM with TSQ can provide a lower average queuing delay with less variation than can an AQM alone.

Figure 4 (right) shows a CDF plot for the throughput (calculated every 300 ms, about the round-trip time) obtained by the audio flow for delay hints of 1, 6 and 16. The throughput distributions of the file transfer flows are similar to the distributions obtained with delay hints of 16. If TSQ were not used then the throughput distribution would be similar to that of a flow with delay hint 16. As is evident from the figure, the median throughput decreases as the delay hint decreases.

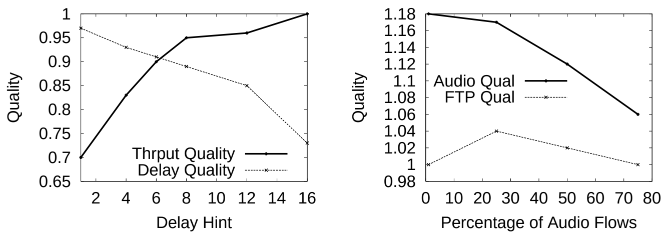

Fig. 5. (Left) Throughput and Delay Quality for Audio Conference Flow versus Delay Hint. (Right) Normalized Quality of Audio Conference Flows and File Transfer Flows for Varying Traffic Mixes.

Using the quality model described in Section II and the throughput and total delay (queuing delay plus propagation delay), the quality of the audio flow for different delay hints is computed. Figure 5 (left) shows the delay quality and throughput quality of the audio flow with different delay hints. The delay quality of the audio application improves with a decrease in delay hint, while its throughput quality decreases. In other words, as the application indicates its preference for lower delay, TSQ enables it to "cut" in line more, hence getting a lower average queuing delay which improves its delay quality. However, correspondingly the audio flow gets dropped with a higher probability, hence achieving a lower throughput and causing a decrease in the throughput quality. The overall quality of an application is the minimum of the delay quality and the throughput quality. Thus, the audio conference gets its best overall quality (a 0.90) at a delay hint of 6. When TSQ is not used, the delay obtained by all applications is similar to that obtained by an application with delay hint 16, and the audio conference gets an overall quality of 0.73.

B. Mixed Traffic Evaluation

The setup for this experiment is similar to the first set of experiments (B=15, D=50, N=100). Within the 100 flows, the relative number of delay-sensitive (audio) flows is changed with respect to the number of throughput-sensitive (file transfer) flows. The traffic mixes ran include: 1 audio conference flow, 99 file transfer flows; 25 audio, 75 file transfer; 50 audio, 50 file transfer; and 75 audio, 25 file transfer.[7] The audio conference flows were a TCP-friendly sources sending data at a constant bitrate of 128 Kbps and using a delay hint of 6 (the optimum delay hint from Section IV-A), while the file transfer flows used the maximum delay hint of 16.

The average quality obtained by the file transfer flows and the audio conference flows for the various traffic configurations is calculated. This quality is then normalized against the quality that the application obtained when TSQ was not enabled (the bottleneck router ran only PI without TSQ). In other words, the normalized quality of an application when TSQ is switched off is 1. If an application receives better QoS when TSQ is enabled, then its normalized quality is greater than 1. Conversely, if an application receives lower quality when TSQ is enabled, then its normalized quality is less than 1.

Figure 5 (right) shows that as the percentage of audio conference flows in the network increases, the average gain in quality of the audio conference decreases. As the number of delay sensitive flows increases in the network, the delay sensitive flows will "cut" in line less than they would when there are more throughput sensitive flows, reducing the quality gains. However, notice at all times the normalized quality is greater than 1, hence, the QoS obtained using TSQ is always higher than the QoS without TSQ even with an increasing proportion of audio conference flows.

For the file transfer flows, the normalized quality increases initially with an increase in the number of audio conference flows. However, as the number of audio conference flows increases beyond 25 percent, the normalized file transfer quality starts gradually decreasing. Again, for all traffic mixes, the normalized file transfer quality is greater or equal to 1. Thus, TSQ provides better or equal quality for both audio conference and file transfer applications than does the underlying AQM (PI, in our experiments) without TSQ.

V. Summary and Future Work

This paper presents TSQ, a Traffic Sensitive QoS controller, that responds to varying QoS requirements from diverse Internet applications without employing complicated policing, pricing or per-flow accounting mechanisms. Our evaluation of TSQ using novel QoS metrics demonstrates that TSQ can significantly improve the average quality of all applications over the quality obtained by using an AQM without TSQ. One potential research area is developing quality metrics for more Internet applications and exploring alternative QoS metrics such as taking the average, sum or the product of the throughput and delay qualities. Future work could also include building applications to take advantage of TSQ by dynamically changing their delay hints.

[7] The extreme case of 99 audio conference flows and 1 file transfer flow was not evaluated, as this configuration did not cause sufficient congestion or queuing delay build-up and hence was not useful for comparative evaluation.

REFERENCES

[1] J.-C. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-Based Error Control for Internet Telephony. In *Proceedings of IEEE INFOCOM*, New York, NY, Mar. 1999.

[2] J. Chung and M. Claypool. Dynamic-CBT and ChIPS - Router Support for Improved Multimedia Performance on the Internet. In *Proceedings of the ACM Multimedia Conference*, Nov. 2000.

[3] M. Claypool, R. Kinicki, and A. Kumar. Traffic Sensitive Active Queue Management. Technical Report WPI-CS-TR-04-10, CS Department, Worcester Polytechnic Institute, Apr. 04.

[4] A. Corlett, D. I. Pullin, and S. Sargood. Statistics of One-Way Internet Packet Delays. In *Proceedings of 53rd Internet Engineering Task Force*, Mar. 2002.

[5] R. N. Corporation. Real Networks Guide for Audio Production, 1998.

[6] S. Dimolitsas, F. L. Corcoran, and J. G. P. Jr. Impact of Transmission Delay on ISDN Videotelephony. In *Proceedings of Globecom '93 – IEEE Telecommunications Conference*, pages 376 – 379, Houston, TX, Nov. 1993.

[7] W. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: An Alternative Approach To Active Queue Management. In *Proceedings of the Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2001.

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

[9] Implementation Architecture Specification for the Premium IP Service, 2002. http://archive.dante.net/geant/public-deliverables/GEA-02-079v2.pdf.

[10] Y. Gao and J. Hou. A State Feedback Control Approach to Stabilizing Queues for ECN-Enabled TCP Connections. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, Apr. 2003.

[11] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. *IETF Request for Comments (RFC) 2597*, June 1999.

[12] C. Hollot, V. Misra, D. Towsley, and W. Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. In *Proceedings of IEEE Infocom*, Anchorage, AK, USA, Apr. 2001.

[13] P. Hurley, M. Kara, J. L. Boudec, and P. Thiran. ABE: Providing a Low Delay within Best Effort. *IEEE Network Magazine*, May/June 2001.

[14] S. Iai, T. Kurita, and N. Kitawaki. Quality Requirements for Multimedia Communcation Services and Terminals – Interaction of Speech and Video Delays. In *Proceedings of Globecom – IEEE Telecommunications Conference*, pages 394 – 398, Houston, TX, Nov. 1993.

[15] V. Jacobson, K. Nichols, and K. Poduri. Expedited Forwarding PHB Group. *IETF Request for Comments (RFC) 2598*, June 1999.

[16] R. Liao and A. Campbell. A Utility-Based Approach for Quantitative Adaptation in Wireless Packet Networks. *ACM Journal on Wireless Networks (WINET)*, 7(5), Sept. 2001.

[17] Y. Liu and M. Claypool. Using Redundancy to Repair Video Damaged by Network Data Loss. In *Proceedings of IS&T/SPIE/ACM Multimedia Computing and Networking (MMCN)*, Jan. 2000.

[18] W. Noureddine and F. Tobagi. Improving the Performance of Interactive TCP Applications using Service Differentiation. In *Proceedings of IEEE Infocom*, New York, NY, June 2002.

[19] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. In *Proceedings of ACM SIGCOMM*, Vancouver, Brisish Columbia, Candada, 1998.

[20] M. Parris, K. Jeffay, and F. Smith. Lightweight Active Router-Queue Management for Multimedia Networking. In *Proceedings of Multimedia Computing and Networking (MMCN), SPIE Proceedings Series*, Jan. 1999.

[21] C. Perkins, O. Hodson, and V. Hardman. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. *IEEE Network Magazine*, Sep/Oct 1998.

[22] W. Pugh. Skip Lists: A Probabilistic Alternative to Balalnced Trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[23] S. Shenker, R. Braden, and D. Clark. Integrated Services in the Internet Architecture: An Overview. *IETF Request for Comments (RFC) 1633*, June 1994.

[24] I. Stoica and H. Zhang. Providing Guaranteed Service Without Per Flow Management. In *ACM SIGCOMM Computer Communication Review , Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, Aug. 1999.

[25] J. Zebarth. Let Me Be Me. In *Proceedings of Globecom – IEEE Telecommunications Conference*, pages 389 – 393, Houston, TX, Nov. 1993.