Traffic Sensitive Active Queue Management

by

Abhishek Kumar

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

Dec 2003

APPROVED:

Professor Mark Claypool, Major Thesis Advisor

Professor Robert Kinicki, Major Thesis Advisor

Professor Craig Wills, Thesis Reader

Professor Michael Gennert, Head of Department

Abstract

Internet applications have varied Quality of Service (QoS) Requirements. Traditional applications such as FTP and email are throughput sensitive since their quality is primarily affected by the throughput they receive. There are delay sensitive applications such as streaming audio/video and IP telephony, whose quality is more affected by the delay. The current Internet however does not provide QoS support to the applications and treats the packets from all applications as primarily throughput sensitive. Delay sensitive applications can however sacrifice throughput for delay to obtain better quality. We present a Traffic Sensitive QoS controller (TSQ) which can be used in conjunction with many existing Active Queue Management (AQM) techniques at the router. The applications inform the TSQ enabled router about their delay sensitivity by embedding a delay hint in the packet header. The delay hint is a measure of an application's delay sensitivity. The TSQ router on receiving packets provides a lower queueing delay to packets from delay sensitive applications based on the delay hint. It also increases the drop probability of such applications thus decreasing their throughput and preventing any unfair advantage over throughput sensitive applications. We have also presented the quality metrics of some typical Internet applications in terms of delay and throughput. The applications are free to choose their delay hints based on the quality they receive. We evaluated TSQ in conjunction with the PI-controller AQM over the Network Simulator (NS-2). We have presented our results showing the improvement in QoS of applications due to the presence of TSQ.

Acknowledgements

I extend my sincere gratitude to my advisor Prof. Mark Claypool, who has helped me every step of the way in completion of this thesis work. His dedication has been a true inspiration for me. Thank You for all the guidance and for showing patience with me. I also thank my co-advisor Prof. Robert Kinicki for his guidance during my research work.

I thank Prof. Craig Wills for being the reader for my work and providing useful comments. I thank all my friends who have made my stay at WPI memorable, and hence helped me relax and make life easier for me.

I dedicate my work to my parents, whose life is an inspiration for me, and whose unconditional love and support is the reason why I could reach here and finish this work.

Contents

List of Figures

Chapter 1

Introduction

Applications on the Internet today have different Quality of Service(QoS) requirements in terms of bandwidth and delay. We broadly identify the following different QoS requirements for applications: a) Throughput sensitive applications which benefit from increased throughput and are willing to tolerate large amounts of delay, for example FTP and email; b) Delay sensitive applications which require low delays and are willing to sacrifice some throughput to get lower delays, for example streaming audio/video and IP telephony; c) In between applications, which have less stringent delay requirements than delay sensitive applications, but still benefit from increased throughput, for example Web browsing. Figure 1.1 shows a few applications and their throughput and delay sensitivity.

Unfortunately, the current Internet does not support per application QoS. Instead all applications are treated primarily as throughput sensitive and no attempt is made to provide a lower delay to applications that desire it. Every packet arriving at a router is enqueued at the tail, thus providing the same average delay to all applications. When there is persistent congestion, the router queue builds up and eventually packets have to be dropped. Thus, the router drops packets randomly

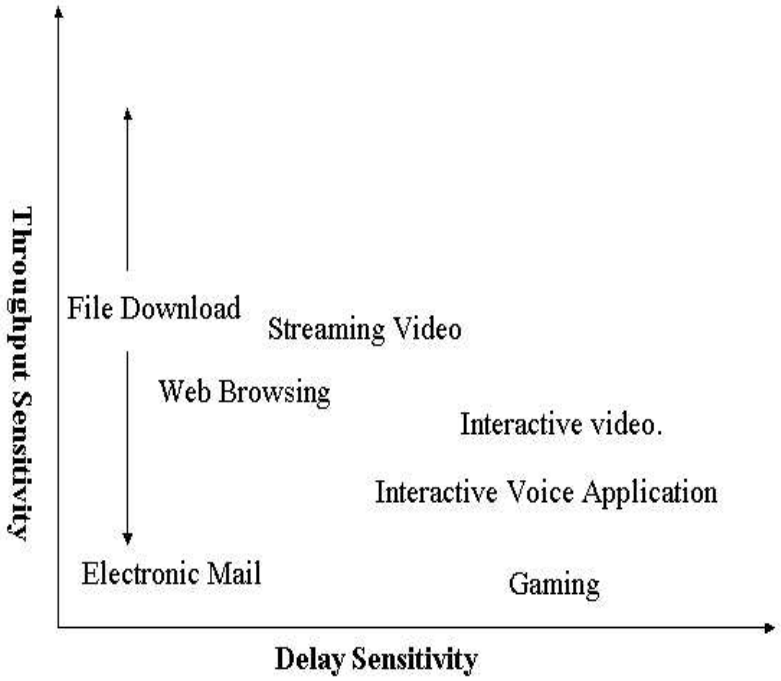

Figure 1.1: QoS Spectrum of Applications

without considering the throughput requirements of the applications sending the packets. Also the large queue build-up causes high queueing delays for all applications, regardless of their delay sensitivity.

However, if the router is capable of providing QoS support, then it could treat packets from delay-sensitive applications differently than those from throughput-sensitive applications. Since the delay-sensitive applications are loss-tolerant, the router can try to provide them with a lower delay and approximately decrease the throughput provided to them. The loss of throughput may not decrease the overall quality of the delay-sensitive applications very significantly, but the reduction in delay can cause a significant improvement in quality. The throughput gained can be allocated to the throughput-sensitive applications, thus providing them with higher

quality.

ABE [HKBT01] provides a queue management mechanism for low delay traffic. ABE allows delay-sensitive applications to sacrifice throughput for lower delays. ABE, however, rigidly classifies all applications as either delay-sensitive or throughput-sensitive. Thus applications are not able to choose relative degrees of sensitivity to throughput and delay. CBT [BFC93] is a class-based approach and provides bandwidth limit guarantees for different classes. However, these fixed and pre-determined classes are not sufficient to represent the varying QoS requirements of applications within one particular class. Similarly, DCBT with ChIPS [CC00], 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) [HBWW99] and Expedited Forward (EF) [JNP99], try to give differentiated service to traffic aggregates. However the DiffServ architectures are very complicated and require the presence of traffic monitors, markers, classifiers, traffic shapers and droppers to enable the components to work together. IntServ [SBC94] provides the best possible per flow QoS guarantees. However, it requires complex signaling and reservations via RSVP by all routers along a connection on a per-flow basis, making scalability difficult for global deployment.

This thesis presents a QoS controller called the Traffic Sensitive QoS Controller (TSQ), that provides per packet QoS support based on an application's delay sensitivity. TSQ works well in the current Internet and can be applied on top of most existing Active Queue Management (AQM) techniques. With TSQ, applications mark each packet with a *delay hint* which is a measure of the application's senstivity to delay. Applications which use low delay hints values degrade under network and queueing delay. On the other hand, applications which use high values of delay

hints depend more significantly on throughput and less on delay. The TSQ router will, on receipt of each packet, examine its delay hint and calculate an appropriate queue position where the packet is to be inserted. A packet from an application which has a low value of delay hint will be allowed to "cut-in-line" towards the front of the queue, while a packet from an application with a high value of delay hint will be inserted towards the end of the queue. To prevent the delay-sensitive applications from gaining an unfair advantage over the throughput-sensitive applications, TSQ proportionately increases the drop probability of the packets inserted into the queue. The more a packet cuts-in-line, the more the packet's drop probability is increased. Throughput-sensitive applications mark their packets with high values of delay hints, and hence they are not cut-in-line, so nor do they have their drop probability increased, thus providing them with good quality. TSQ requires no per-flow state information, no traffic monitoring and no edge policing or marking.

We have evaluated the performance of TSQ when used in conjunction with the PI-controller (Proportional Integral controller) AQM [HMTG01] with varying mixes of delay-sensitive and throughput-sensitive flows. We also evaluated the performance of TSQ in the presence of unresponsive flows (UDP flows in our case). The results obtained suggest that TSQ with PI provides better quality for all application simulation set-ups than PI by itself.

The remainder of the thesis is organized as follows: Chapter 2 describes the related work; Chapter 3 presents the Quality Metrics we have devised for some typical applications; Chapter 4 discusses the TSQ mechanism; Chapter 5 describes the various experiments we conducted to evaluate TSQ and the analysis of the results obtained and Chapters 6 and 7 summarise our work and discuss the possible future work.

Chapter 2

Related Work

In this chapter we discuss some of the work related to this thesis. In Section 2.1 we discuss different approaches that have been developed to provide QoS support to Internet applications. Some approaches are best effort and attempt to provide per-packet QoS, while some are class-based algorithms. We briefly discuss in Section 2.2 some of the mechanism that use source hints in the form of bits in the packet header. Section 2.3

2.1 QoS Support

Various suggestions have been proposed to make the Internet be able to provide varying degrees of QoS support to the applications using it. In this section we describe a few approaches towards providing QoS support on the Internet. In [PCK02] two algorithms viz. Red-Worcester and Red-Boston have been proposed. RED-Worcester is a simple extension to the ARED [FGS01] mechanism and tries to meet the average performance requirements of incoming packets in terms of throughput and delay, thus improving the overall QoS support at the router. It has a moving target queue size, which depends on the average QoS requirements of the incoming

traffic. The QoS requirement of the applications is specified by inserting a delay hint in the IP header of each packet. If incoming traffic is mostly throughput-sensitive RED-Worcester maintains a higher average queue to improve the overall throughput. On the other hand, when incoming traffic is mostly delay sensitive, RED-Worcester lowers the average queue size to reduce the average queueing delay. RED-Boston calculates the average drop probability based on the average queue size like ARED, but adjusts the per-packet drop probability p' based on the delay hint and the average delay as follows:

$$p' = p \times delay/delay\_hint$$

RED-Boston inserts packets in the queue sorted by weight, where the weight is calculated from the delay hint and the arrival time of each packet as follows:

$$weight = arrival\_time + delay\_hint$$

Thus packets with lower delay hints will have lower weights and will be inserted further into the queue. This will provide packets with lower delay hints to achieve lower queueing delays. Thus RED-Boston attempts to provide a per-packet QoS. It does not provide any guarantee of service but instead it is a best effort service.

Another approach proposed is Alternate Best Effort (ABE) [HMTG01]. ABE identifies two classes of network traffic; delay sensitive (green) and throughput sensitive (blue). It provides the packets from two classes with different treatment by maintaining two separate queues, a larger queue for blue traffic and a smaller queue for green traffic and uses a deadline based scheduler to serve the packets from these queues. This allows delay-sensitive applications to sacrifice throughput for lower de-

lays without affecting throughput-sensitive applications. However, as ABE provides just two classes of service, it restricts the applications to be either delay-sensitive or throughput-sensitive without allowing them to choose their own delay-throughput tradeoff.

The approach presented in [NT02] provides service differentiation to interactive TCP applications by prioritizing the packets from these applications. Some other approaches to provide QoS service are class based mechanisms such as CBT [Par01], CBQ [Flo], IntServ [SBC94], DiffServ [HBWW99] [JNP99] etc.

2.2 Source Hints

In both RED-Worcester and RED-Boston described in Section 2.1, applications send source hints (in form of delay hints) to inform the router about their QoS requirements. These source hints can be placed in the IP or TCP header. In the Context-Aware TCP/IP proposed in [Car02], context information is provided by marking the packet headers. This is another form of source hint. In [NT02] a mechanism for source marking is suggested, where the applications mark the headers with their priority in terms of delay (High, Med, Low). Similarly sometimes the hints are provided by the edge routers instead of the source. In CSFQ [SSZ98] edge routers compute per-flow rate estimates and label the packets passing through them by inserting these estimates in the header. Similarly in [GM01] edge routers mark individual packets as belonging to the class of short flows or the class of long flows.

2.3 Multimedia Quality

Multimedia applications have different QoS requirements from traditional throughput-sensitive Internet applications. We discuss some of the work that has been done to measure the quality of these applications. [IKK93] studies the variation in the perceived quality (Mean Opinion Scores also known as MOS) of Internet audio applications with round-trip delay. The study done in [DCJ93] measures the effect of one-way transmission delay on different aspects of communication such as difficulty, ease of interruption and general acceptability. [Zeb93] conducts tests on the effect of delay on audio and video telephony.

2.4 TCP friendliness

The throughput of a TCP flow is given by the TCP response function [PFTK98] as follows:

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)}$$

where s is packet size, R is round-trip time, p is the steady-state loss rate and t_{RTO} is the TCP retransmit timeout value. In [FF99] a TCP-friendly flow is defined as flow is defined as the flow whose arrival rate does not exceed the arrival rate of a conformant TCP connection in the same circumstances. The following equation places an upper bound on the througput of a TCP-friendly flow.

$$T \leq \frac{1.5\sqrt{1/3} \times B}{R \times \sqrt{p}}$$

where B is the bytes per packet and R is the round-trip delay including the queueing delay. It is important for applications to use TCP-friendly protocols to keep network

congestion in control. Applications using these protocols respond to packet drops by reducing their rate of transmission, thereby decreasing congestion. Protocols such as UDP do not respond to packet drops and are not TCP-friendly. Thus it is possible for UDP flows to gain more than their fair share of bandwidth. Hence, in order to maintain TCP-friendliness of TSQ, any unfair gain in bandwidth by a particular flow should be prevented.

2.5 AQM approaches

The problems faced by Drop-Tail routers in face of persistent congestion such as global synchronization, lockouts and full queues have prompted various Active Queue Management(AQM) algorithms. Random Early Detection (RED) [FJ93] maintains two thresholds for the current queue size, known as $\min_{th}$ and $\max_{th}$. If the queue size goes above $\min_{th}$, then the router assumes imminent congestion and starts dropping packets randomly with a drop probability of p. As the queue size builds up, the drop probability is increased even further, to reduce congestion. If the queue size exceeds $\max_{th}$, then the router drops all incoming packets, i.e. the drop probability is now 1.

Another AQM technique that uses a drop probability to control queue is the PI-controller [HMTG01], which we have used in our experiments. The PI-controller tries to maintain its average queue size close to a defined reference called the q_{ref}. The PI-controller also maintains a drop probability p, which is updated at fixed interval. The following set of equations govern the drop probability of the PI con-

troller:

$$
\begin{aligned}
p = &\quad a \times (q - q_{ref}) - b \times (q_{old} - q_{ref}) + p_{old} \\
p_{old} = &\qquad\qquad\qquad p \\
q_{old} = &\qquad\qquad\qquad q
\end{aligned}
$$

where a and b are parameters of PI. The PI controller prevents excess queue build up during congestion. Hence it can prevent very large queueing delays at the expense of higher loss rates. BLUE [FKSS01] is another AQM that uses a drop probability p for congestion control. However other AQM algorithms such as REM [ALLY01] and AVQ [KS01] etc. do not have an explicit drop probability p.

Chapter 3

Quality Metrics

In this chapter we describe our approach in developing quality metrics which are used to quantify the performance of three network applications: interactive audio, such as used in IP telephony, interactive video, such as used in a video conference and file transfer applications such as used in P2P or FTP. Based on information from previous work [Gan] [IKK93] [DCJ93] [Zeb93], we have devised quality functions for these three applications in terms of their network delay and the network throughput called the *delay quality* (Q_d) and *throughput quality* (Q_t) respectively. The overall quality of the application is the minimum of the two quality metrics.

$$Q(d, T) = min(Q_d(d), Q_t(T))$$

The quality measure lies between 0 and 1, where 1 represents the maximum quality that the application requires, and a quality of 0 is of no use to the application at all.

3.1 Interactive Audio Quality

In this section we discuss the quality functions that we have derived for an interactive audio application. The quality functions are of two types, the delay quality function and the throughput quality function. We have graphed the quality functions for the application versus one-way delay and throughput respectively.

3.1.1 Effect of Delay on Interactive Audio Quality

As discussed earlier, the quality of interactive audio applications over the network is very sensitive to delay. On the other hand these applications are less sensitive to reduction in throughput. [Gan] suggests that audio quality in terms of delay is essentially divided into 3 parts. A one-way delay of 150 ms or less means excellent quality for the audio application. As the delay increases the audio quality decreases, and a one-way delay of 150-400 ms means good quality. A one-way delay in excess of 400 ms is poor quality. Also [IKK93] has observed the variation of audio quality with delay in terms of Mean Opinion Scores (MOS scores). The graph in Figure 3.1 is from [IKK93] and it shows the variation of MOS scores for free conversation with round-trip delay.

Based on this previous work, we have produced the graph in Figure 3.2 plotting the *delay quality* of an interactive audio application. The best quality possible is 1, and it happens when there is a zero delay. A quality of 1 can be considered equivalent to a MOS score of 5 which means excellent quality. The audio application has an excellent quality if the one way delay is within 150 ms. As the delay starts increasing, the fall in quality is not very significant, and a delay of 150 ms provides the application with a quality of 0.98, an almost insignificant drop in quality. However, as the delay increases above 150 ms, the drop in quality now becomes significant. A

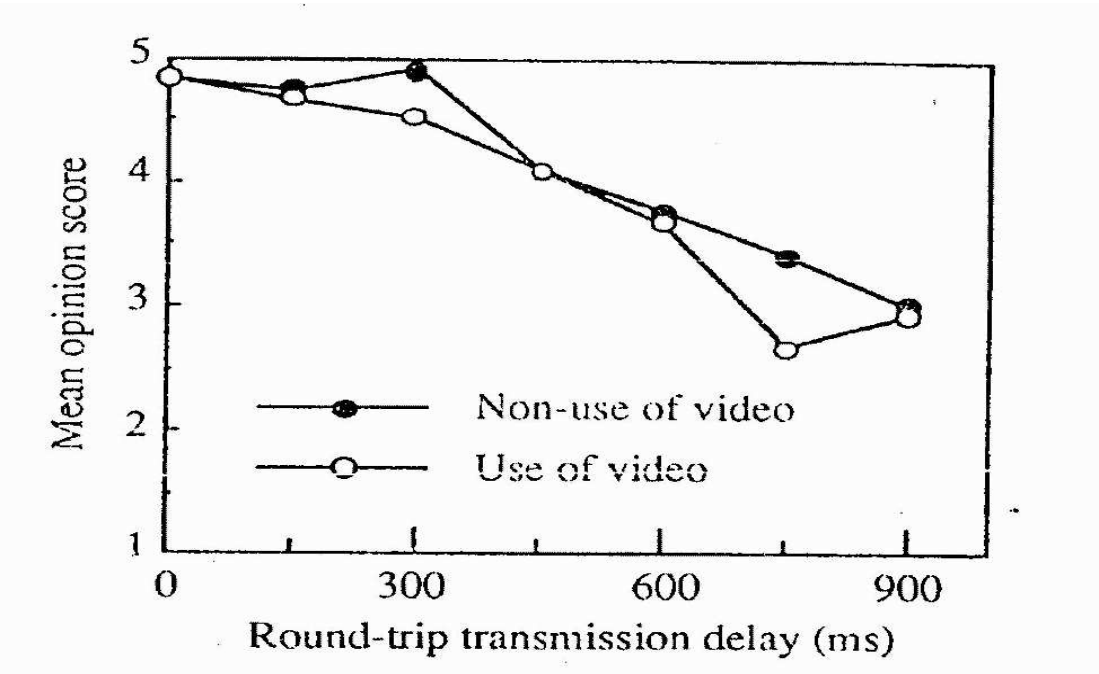

Figure 3.1: MOS scores versus round-trip delay

delay of 300 ms reduces the quality to 0.7 (equivalent to a MOS score of 3.5) and it reduces further to 0.5 (equivalent to a MOS score of 3) when delay is 400 ms. This is the minimum acceptable quality for interactive audio. As the delay increases higher than 400 ms, the quality of the application degrades significantly. We propose that the degradation after 400 ms is about twice the degradation in quality from 150 to 400 ms delay. Thus, from the graph we can see the three broad sections of quality described in [Gan] and also get the value of the quality for intermediate one-way delays. The set of equations governing the delay quality of an interactive audio

application is as follows:

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

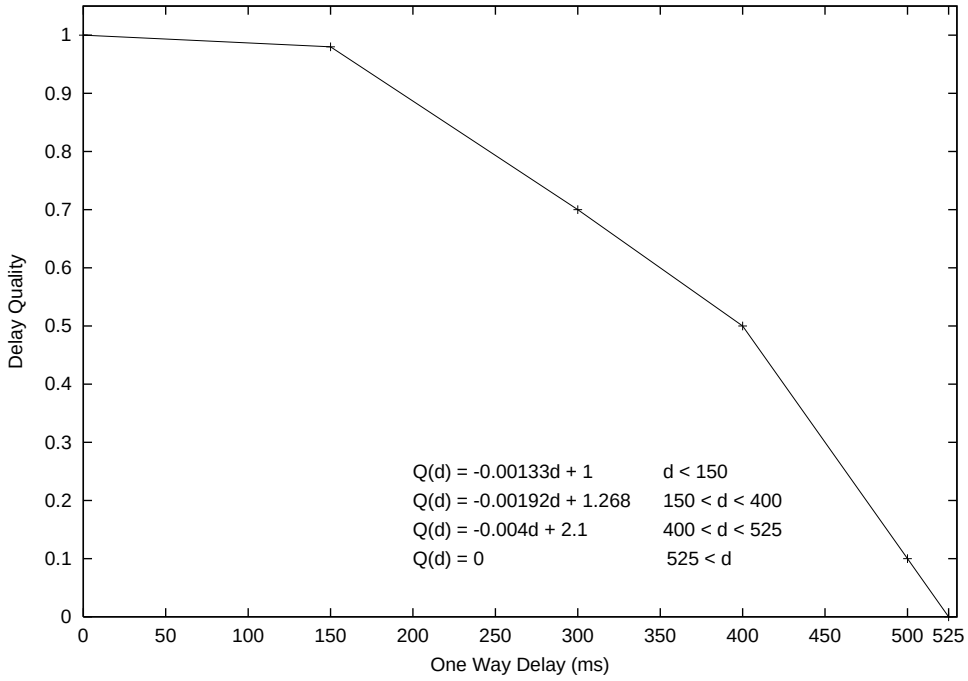

Figure 3.2: Delay Quality for Interactive Audio versus One Way Delay

3.1.2 Effect of Throughput on Interactive Audio Quality

Figure 3.3 shows the plot of quality for an interactive audio application versus the throughput that the application receives (*the throughput quality*). The application has a thoughput quality of 1 when the throughput is 128 Kbps, since at this bit-rate the quality of audio is of CD quality, which we assume is the best possible. The

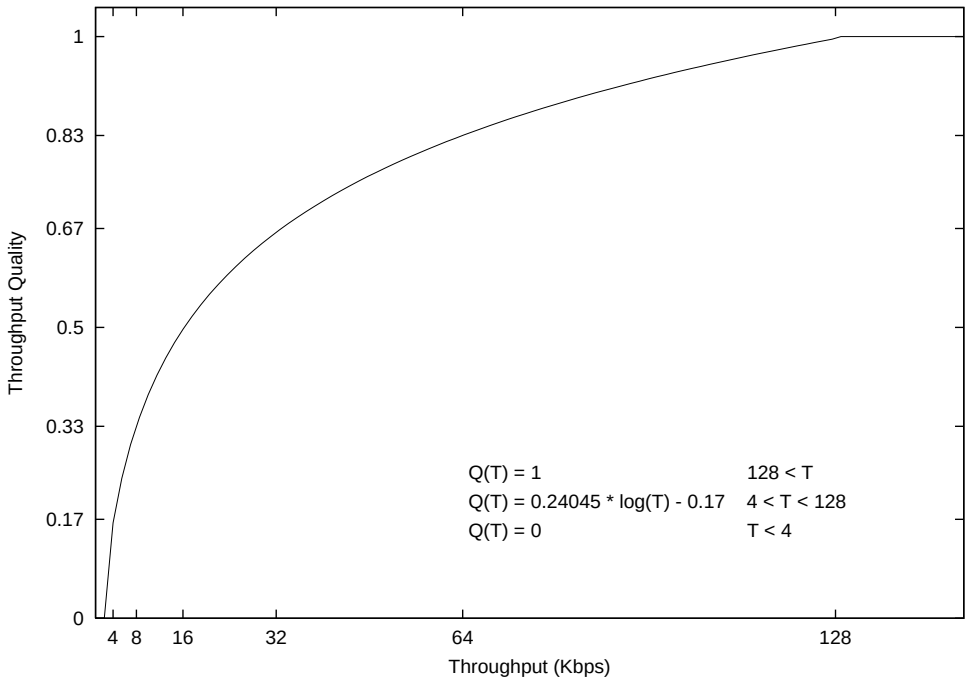

Figure 3.3: Throughput Quality for Interactive Audio versus Throughput

throughput quality decreases linearly as the throughput is halved. This is because, every time one fewer bit is used to encode the audio, the throughput of the audio codec is reduced by half. We assume that the quality of the audio application reduces linearly with the reduction in the number of encoding bits. Hence the variation of audio quality with throughput is a logarithmic curve, where a reduction in throughput above 64Kbps does not greatly reduce the quality of the application, 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 further to 0, when the throughput is 2 Kbps. This is because 4 Kbps is the lowest codec rate available for audio application [Cor98]. The set of equations for the throughput quality are

as follows:

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

3.2 Interactive Video Quality

As another representative delay sensitive application we derived quality metrics for an interactive video application, specifically a typical H.323 video-conference. We have developed quality functions in terms of both one-way delay and throughput.

3.2.1 Effect of Delay on Interactive Video Quality

Since the nature of the interactivity of a video conference is the same as that in an audio conference, the delay requirements of a video conference are similar to those of an interactive audio application described in Section 3.1.2. Hence the plot in Figure 3.2 showing the delay quality of interactive audio also represents the delay quality of a video conference. The formulae for interactive audio delay quality suggested in Section 3.1.2 also apply to delay quality of video conference.

3.2.2 Effect of Throughput on Interactive Video Quality

Typically, a H.323 video conference requires 384 Kbps of bandwidth for good quality [Cor00]. If the application receives this throughput we assign it a quality of 0.8, based on the MOS scale, where a score of 4 on a scale of 1-5 is considered good. As the throughput provided to the application increases the quality of the application increases, but in a smaller proportion. Thus, the quality increases to 0.85 when

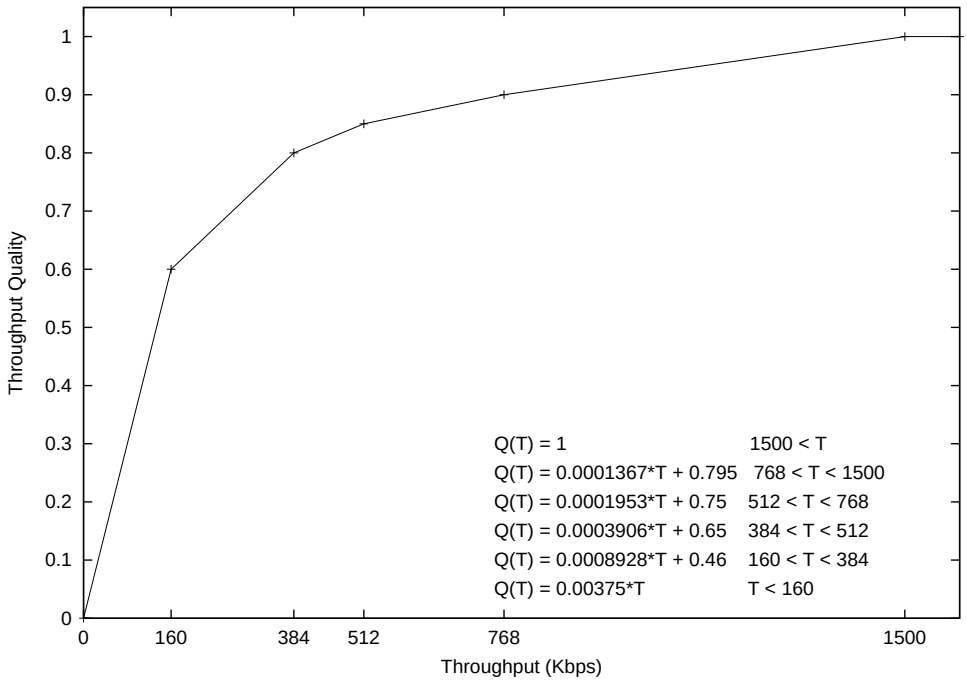

Figure 3.4: Video Quality versus Throughput

throughput is 512 Kbps, and to obtain a quality of 0.9 the throughput required is 768 Kbps. It gets its best quality of 1 when the throughput is 1.5 Mbps because a H.323 video conference operating at 1.5 Mbps is of excellent quality [Cor00]. Any subsequent increase in the throughput does not improve the quality. An H.323 video conference has average quality if it has a throughput of 160 Kbps. Thus, we assign this throughput a quality value of 0.6 corresponding to a MOS score of 3 which is considered as "fair" quality. Any further reduction in throughput will cause the quality to fall off sharply. We thus come up with the following set of equations which

determine the throughput quality for the video application, depicted in Figure 3.4.

$$
\begin{aligned}
Q_t(T) &= 1 && 1500 < T \\
Q_t(T) &= 0.0001367 \times T + 0.795 && 768 \leq T \leq 1500 \\
Q_t(T) &= 0.0001953 \times T + 0.75 && 512 \leq T \leq 768 \\
Q_t(T) &= 0.0003906 \times T + 0.65 && 384 \leq T \leq 512 \\
Q_t(T) &= 0.0008928 \times T + 0.46 && 160 \leq T \leq 384 \\
Q_t(T) &= 0.00375 \times T && T \leq 160
\end{aligned}
$$

3.3 File Transfer Quality

In this section we discuss the quality metrics we used to measure the quality of file transfer applications. File transfer applications, unlike the interactive audio and video conference applicacations, are not delay sensitive. Instead, the quality of these applications is almost entirely dependent on their throughput.

3.3.1 Effect of Delay on File Transfer Quality

The quality of the FTP application is not affected by an increase in the delay over typical Internet latencies. This is because the application is not delay-sensitive and hence an increase in the delay the order of a few 100 ms, will not affect its quality. The 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 queueing delay. Since in our experiments, the delay is generally on order of few 100 ms, we ignore the effect of delay on FTP quality beyond 1000 ms. If the delay is within this range, the delay quality of the FTP application will be the best possible. Figure 3.5 shows the graph

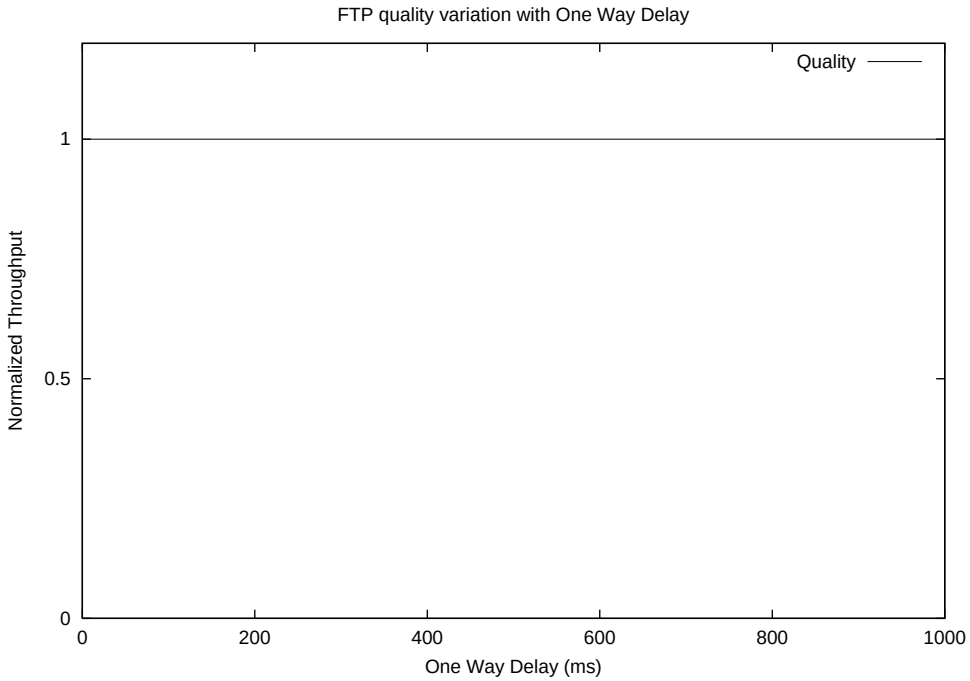

Figure 3.5: Delay Quality of File Transfer versus One Way Delay

and the equation is as follows:

$$Q_d(d) = \quad 1 \quad d \leq 1000$$

3.3.2 Effect of Throughput on File Transfer Quality

The quality of a file transfer application depends almost entirely on the throughput that it can get from the network. In this section, we quantify its quality in terms of its throughput. Since, the size of files being transfered can vary greatly, it is not appropriate to have a single curve represent the quality requirements of all file transfers. Hence in our quality metrics, the quality requirements of a file transfer is 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 an FTP application has the best quality if it can finish transferring a

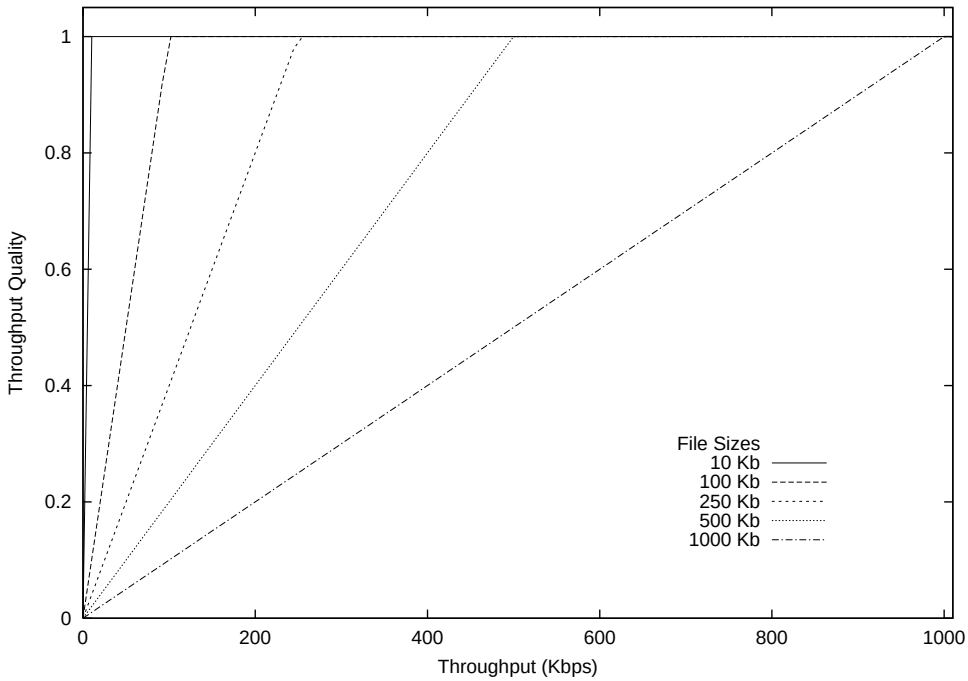

Figure 3.6: Throughput Quality of File Transfer Application versus Throughput

file in 1 second. Thus for 10 Mb file, the best quality of 1 is attained if it can get a throughput of 10 Mbps. If the throughput obtained is greater, the quality does not improve. However, the decrease in quality is directly proportional to a decrease in throughput. Similarly for a smaller file of 10 Kb, the required throughput for best quality will be only 10 Kbps. We derive the following equation for throughput quality of file transfer applications:

$$Q_t(T, S) = \quad T/S$$

where S is the size of the file. Figure 3.6 shows the quality plots of file transfer applications with various file sizes. As can be seen, the quality plot for a small file transfer is much steeper as compared to that of a large file.

Chapter 4

Mechanism

The Traffic Sensitive QoS controller (TSQ) is a Quality of Service (QoS) controller that can be in conjunction with many existing Active Queue Mechanisms (AQMs). TSQ accomodates delay sensitive applications, such as interactive multimedia, by providing a low queueing delay, while at the same time not penalizing the throughput of the traditional applications, such as file transfers. TSQ achieves this per-application QoS by providing a trade-off between queueing delays and drop probabilities. The applications inform TSQ about their delay sensitivity by providing a *delay hint*. A TSQ-enabled router provides flows with a low delay hint with a lower delay by using a "cut-in-line" mechanism. 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.

In Section 4.1, we first describe how applications notify the TSQ router about their delay sensitivity by using a *delay hint*. In Section 4.2, we describe the "cut-in-line" mechanism which is used to provide delay sensitive applications with lower queueing delays. Section 4.3 discusses the adjustment in drop probability that is made for the delay-sensitive flows so that they do not get unfair advantage over

throughput-sensitive flows. Section 4.4 concludes with a diagram and algorithm detailing TSQ.

4.1 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 throughput-sensitive and does not require a low delay for improvement in quality. Applications such as interactive multimedia will typically provide low delay hints. On the other hand, applications such as file transfer will typically provide higher delay hints.

Based on the discussion in [SZ99] there are 4 to 17 bits available in the IP header that can be used to carry hint information. In our current implementation of TSQ, the range of delay hints is from 1 to 16 requiring 4 bits in the packet header. Thus, an application which chooses the minimum delay hint of 1 will be extremely delay-sensitive, in contrast to an application which can tolerate some delay and hence will have the maximum delay hint of 16. If the number of bits used for the delay hints is increased, the applications will have more levels of delay-senstivity to choose from, hence more accurately representing their QoS requirements, but at the cost of increased overhead in each packet header. Similarly if the number of bits used to represent delay hints is reduced, the applications will have a smaller range of delay-sensitivity to choose from, but less overhead per packet. The optimal number of bits for delay hints is left as future work.

4.2 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 queueing delay. The queueing delay obtained by each packet depends upon the instantaneous queue length (q) and the outgoing link capacity. TSQ provides delay-sensitive packets with a lower queueing 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 queueing 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 this, we introduce an aging mechanism to prevent starvation. This cut-in-line mechanism is implemented by using a weighted insertion into the queue. At the arrival time (t_a) of a packet, we calculate the queueing delay that the packet would experience if it was inserted at the end of the queue; we call this queueing delay the *drain time* (t_d) of the queue. TSQ calculates its weight (w) according to its delay hint and time of arrival at the queue.

$$w = \frac{d \times t_d}{2^n} + t_a \tag{4.1}$$

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 the lower weight packets inserted towards the front of the queue and the higher weight packets inserted towards the end of the queue. This new position of the packet in the queue is called *q'*. Thus, a high delay-hint will cause a packet to have a higher weight and hence a higher value of *q'*. Newly arriving packets will have their weights slightly increased due to the effect of the time of arrival on their weight, thus preventing starvation of older packets.

This cut-in-line requires a weighted insertion that can be implemented using a probabilistic data structure such as skip lists [Pug90], giving complexity *O(log(l))*, where l is the number of packets in the queue.

4.3 Drop Probability

During congestion, many AQM techniques produce a 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 inserted in the queue. However, in the case of the TSQ, a uniform drop probability for all packets will potentially result in a higher throughput for the delay-sensitive applications, since it is providing 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 queueing 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 steady state throughput T of a TCP flow in which throughput is inversely proportional to the queueing delay and the square root of the loss rate [PFTK98]:

$$T = \frac{K}{r \times \sqrt{p}} \tag{4.2}$$

there, 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 queueing delay and the round-trip propagation delay. Since some packets can have a decreased queueing delay by cutting in line, we compensate by increasing the drop probability for those packets. Let the new queueing 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':

$$T' = \frac{K}{(l + q') \times \sqrt{p'}} \tag{4.3}$$

We want to prevent the new throughput T' from being greater than the throughput obtained without TSQ, ($T' \leq T$). Hence, we calculate the new drop probability p' as:

$$p' = \frac{(l + q)^2 \times p}{(l + q')^2} \tag{4.4}$$

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, we keep the value of l as a constant, but is typically between 40-100 ms

for many Internet links [CPS02]. Setting l to lower values in this range will result in a more aggressive increase in drop probability, while setting l to higher values of propagation delay will result in less aggressive increase in drop probability. For our experiments, we fixed the one way propagation delay constant for the router at 40 ms[1].

4.4 Summary

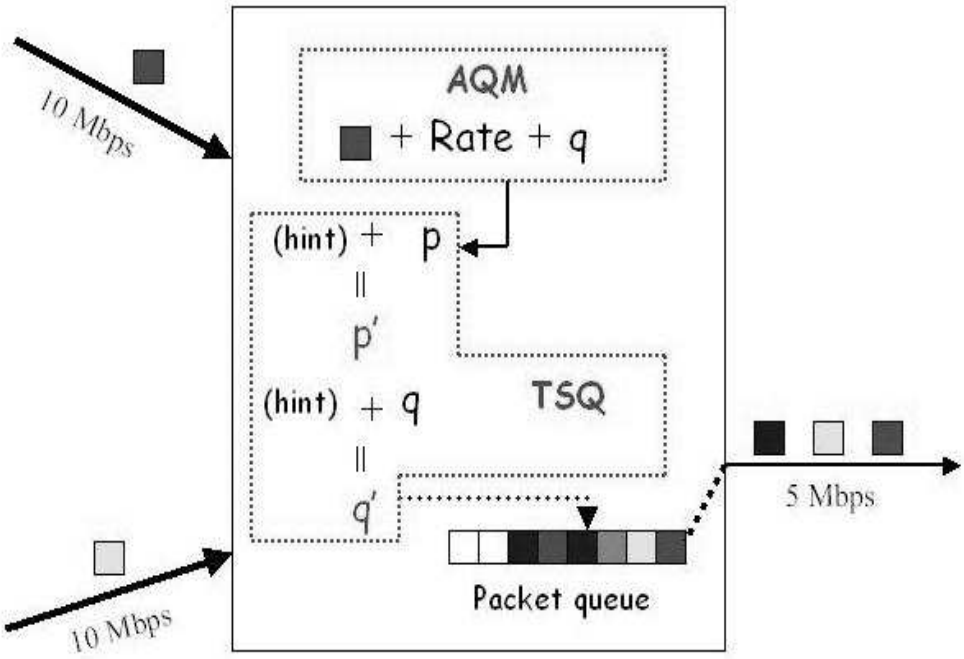

Figure 4.1: Traffic Sensitive QoS Controller along with an Active Queue Mechanism

Figure 4.1 shows TSQ in relation to the AQM. TSQ is essentially a QoS controller which will work in conjuction with an AQM. The congestion controller can be any AQM technique that provides an aggregate drop probability (PI in our case). The

[1]Note that this value is fixed for the TSQ router for all experiments although the experiments will be simulated on networks with different propagation delays.

congestion controller is responsible for controlling the flow of packets through the router. Most AQM techniques try to control the size of the router queue by dropping packets with a uniform drop probability. Packets that are not dropped are queued at the end in a FIFO queue. TSQ examines the delay hint of each packet and determines the position in the queue where the packet should be inserted. It also adjusts the drop probability of the packet based on the position on which it will be inserted. If, after adjusting the drop probability the packet is not dropped then it is inserted into the queue in the previously determined position. The QoS controller thus aims to provide per packet QoS.

Figure 4.2 summarizes the algorithm for TSQ.

```
//q - length of queue
//q' - position in the queue, where the packet will be inserted
//C - capacity of queue
//w - packet weight
//d - delay hint
//t_d - drain time
//n - number of bits used for delay hints
//t_a - time of arrival
//p - drop probability before TSQ
//p' - drop probability after TSQ
//l - network latency
on receiving packet pkt:

// Calculate its drain time
t_d = q/C

// Calculate packet weight
w = (d × t_d)/2^n + t_a

// Determine new position of packet in the queue
q' = weightedInsert(w,pkt)

// Calculate new drop probability
p' = ((l+q)^2 × p) / ((l+q')^2)

// Generate random number between 0 and 1
r = uniform[0,1]

if (r ≤ p') then
   drop(pkt)
else
   insertPacket(q', pkt)
```

Figure 4.2: TSQ Algorithm

Chapter 5

Experiments

This chapter describes the experiments which were carried out to evaluate the Traffic Sensitive Quality of Service Mechanism (TSQ) over an existing Active Queue Management (AQM) technique. We chose the PI-controller [HMTG01] as the AQM technique over which to evaluate TSQ. The PI-controller attempts to provide a steady queueing delay by keeping the queue size stable around a target queue length, by adjusting the drop probability applied to incoming packets. Thus, PI provides an explicit drop probability that enables TSQ to be easily placed on top of the PI controller AQM.

We conducted a variety of experiments to test the effect of TSQ on the quality of interactive audio, interactive video and file transfer flows. We also measured the variation in queueing delay and throughput for the audio and video flows. We also make a comparison between the performance of the applications with PI and TSQ to the same applications with only PI. Finally, we ran experiments to measure the effect of unresponsive flows when using TSQ in order to verify that non-responsive flows do not benefit from TSQ.

5.1 Experimental Setup

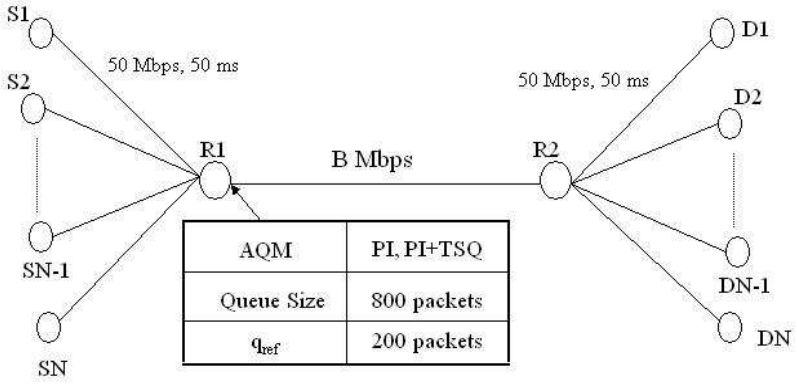

Figure 5.1: Network Topology

All the experiments were done over the Network Simulator (NS-2) [ref]. Figure 5.1 shows the generic network topology for all the experiments in the simulation. There are N sources S1..SN and N destinations D1..DN. These 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 node have a link capacity of 50 Mbps and 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. This bottleneck router runs the PI AQM [HMTG01] plus TSQ. PI is configured with the values recommended in [HMTG01], a = 0.00001822, b = 0.00001816, w = 170, q_{ref} = 200 packets and q_{max} = 800 packets. The average packet size is 1000 bytes.

5.2 Audio Quality Evaluation

In this experiment we evaluate the performance of a single interactive audio flow sharing the network with other TCP based FTP flows.

5.2.1 Experimental Setup

The network topology is same as the generic setup described in Section 5.1. The bottleneck link capacity B is 15 Mbps. The one-way propagation delay D of the bottleneck link is 50 ms. Hence the one-way propagation delay between each of the sources and their respective receivers is 150 ms. In this experiment N = 100, hence there are 100 sources and 100 destinations. Among the 100 flows present in the network there are 99 TCP based FTP bulk transfer flows that are not delay sensitive and so provide the maximum delay hint of 16, and the other flow is an interactive audio flow simulated as a TCP-friendly source sending data at a rate of 128 Kbps. The experiment is allowed to run for 100 seconds. At the end of each run, we change the delay hint of the audio flow. Hence, we evaluate the performance of the audio flow over several runs with different delay hints.

5.2.2 Analysis

We analyze the effect of different delay hints on the queueing delay and throughput of the audio flow.

Figure 5.2 plots the CDF of the queueing 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 gave the audio flow its optimal quality, and delay hint 16, which gives maximum delay. As we can see from the figure the

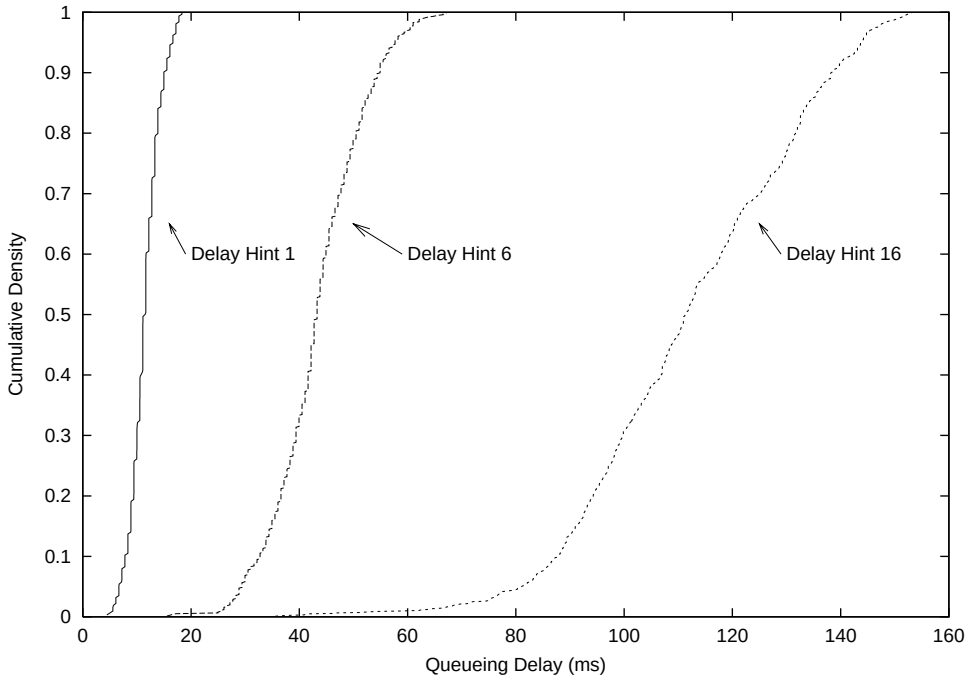

Figure 5.2: CDF of the queueing delay experienced by the audio flow with delay hints of 1,6 and 16.

median queueing delay is lower for the lower delay hints, also the CDF plots for lower hints are much steeper, which implies that there is less variation in the per-packet queueing delay at lower hints. Hence, for delay sensitive applications an AQM with TSQ can provide a lower average queueing delay with less variation than can an AQM alone.

Figure 5.3 shows a similar CDF plot for the throughput obtained by the audio flow for the delay hints of 1, 6 and 16. The throughput is calculated every RTT (300 ms in this experiment). 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 becomes lower.

We measured the throughput and total delay (queueing delay plus propagation

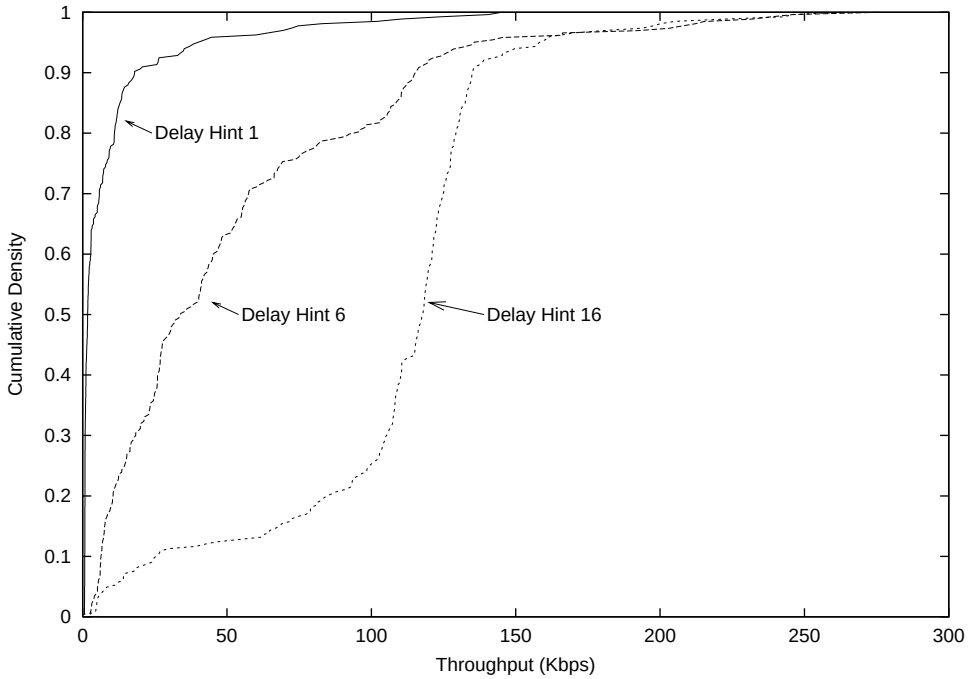

Figure 5.3: CDF of the throughput experienced by the audio flow with delay hints of 1, 6 and 16.

delay) experienced by the audio flow for the various delay hints. Using the quality model described in Chapter 3 we plot the quality of the audio flow as the delay hints vary. Figure 5.4 shows the variation of 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, it is "cutting" more in line, hence getting a lower average queueing delay which improves its delay quality. However, correspondingly the audio flow gets dropped with a higher probablity, hence acheiving a lower throughput and causing a drop in the throughput quality.

Figure 5.5 shows the overall quality of the application for various delay hints. As we have proposed in our quality model, the overall quality of an application is the minimum of the delay quality and the throughput quality. Thus the application

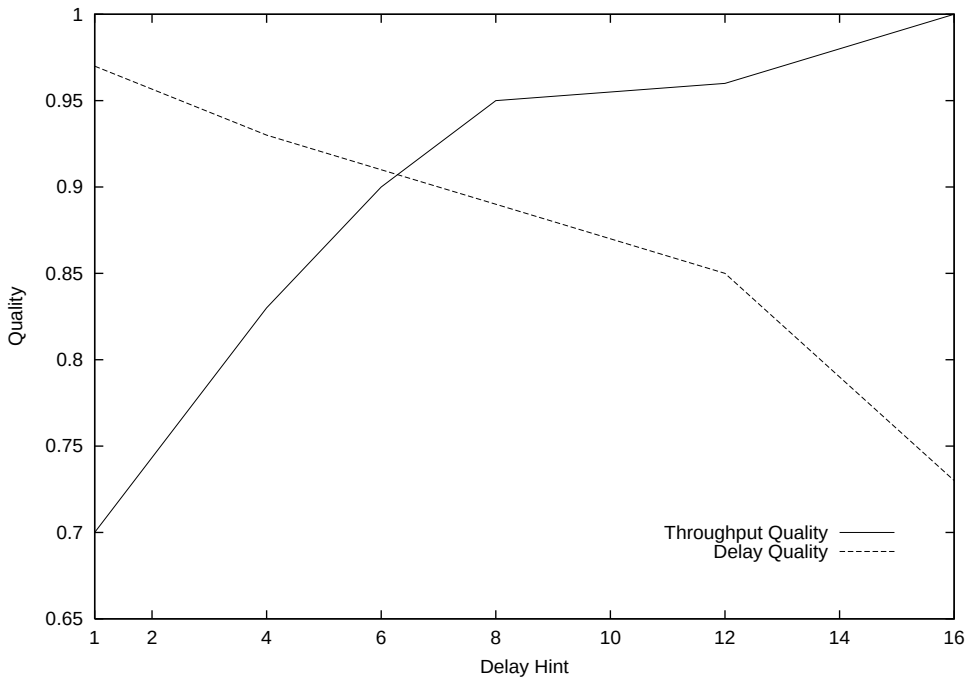

Figure 5.4: Throughput and Delay Quality for an audio flow versus Delay Hint

gets its best overall quality 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.

5.3 Video Quality Evaluation

The experiments conducted in the previous section and the analysis following it showed how TSQ can be used to improve the quality of interactive audio applications. We next present a series of experiments showing how TSQ can be useful for improving the quality of interactive video applications.

5.3.1 Experimental Setup

The topology for the video-conference is similar to the generic dumbell topology described in Section 5.1. In these experiments, we have 20 flows (N=20), and a

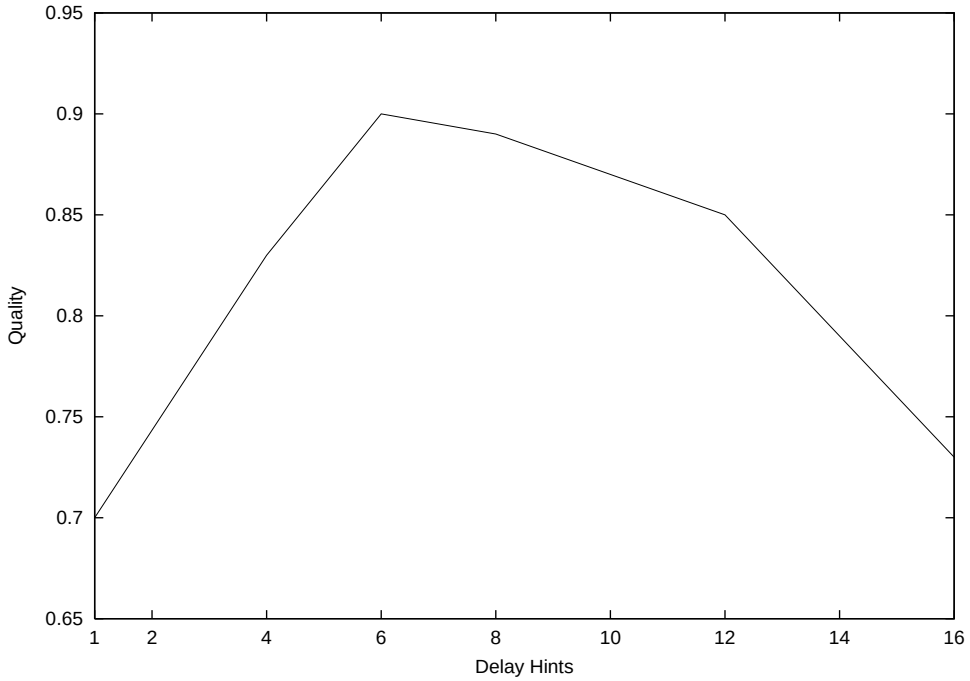

Figure 5.5: Quality for an audio flow versus Delay Hint

bottleneck link of 4 Mbps. Of the 20 flows, 19 are bulk file transfers, and one flow is a TCP-friendly CBR source sending data at a rate of 500 Kbps, typical of a H.323 video-conference [Cor00]. All other parameters remain the same as in the previous set of experiments. We run each simulation for 100 seconds and change the delay hint of the video flow after each run.

5.3.2 Analysis

Figure 5.6 shows the CDF of the queueing delay for the video flow for delay hints of 1, 6 and 16. As seen in Section 5.2.2 for the audio application, the median queueing delay for the video application is lower for the lower delay hints. Also, the CDF plots for lower hints are much steeper, which implies low variance in the queueing delay. Thus similar to the interactive audio, TSQ can provide a lower queueing delay with less variation to the interactive video application.

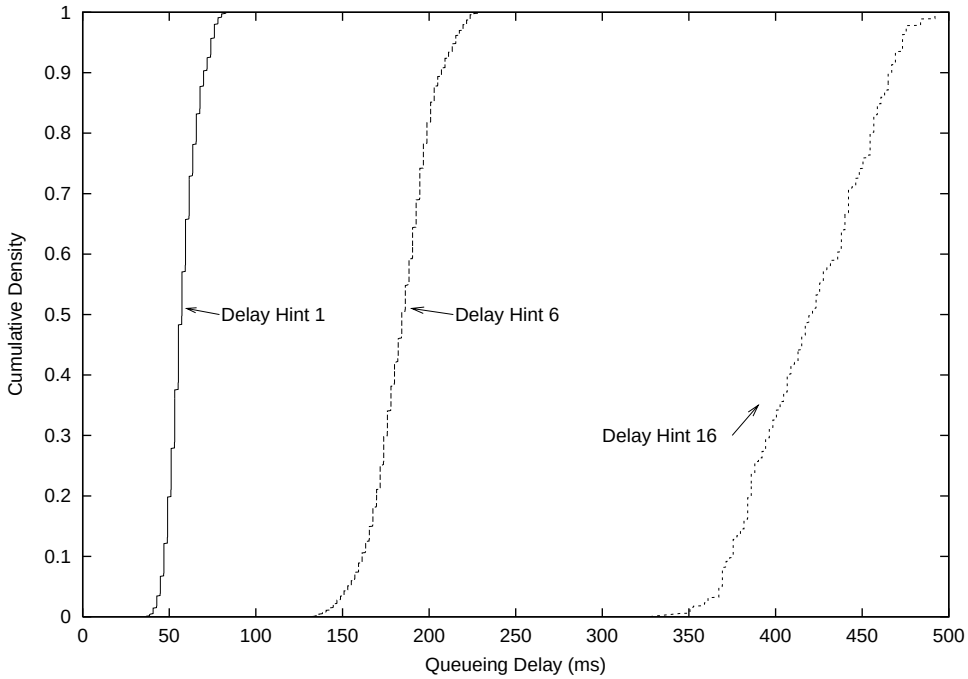

Figure 5.6: CDF of queueing delay of the video flow for delay hints of 1, 6 and 16

Figure 5.7 shows the CDF of the throughput obtained by the video flows for the same 3 delay hint values. The throughputs are calculated over 1 RTT (300 ms in our experiments). The three CDF plots are closer for video as compared to interactive audio, indicating that the decrease in throughput is not very significant when the delay hint is reduced.

The graph in Figure 5.8 shows how the quality of the video flow is affected by different delay hints. For lower delay hints, the average queueing delay and hence the average delay for the video flow decreases. This results in a significant gain in delay quality. However, the drop in throughput quality is not as significant for lower delay hints as it was in the audio experiments.

Figure 5.9 shows the overall quality of the video flow for different delay hints. The quality for the video conference is maximum when the delay hint is 6. Thus, by changing its delay hint, the application can get optimal quality.

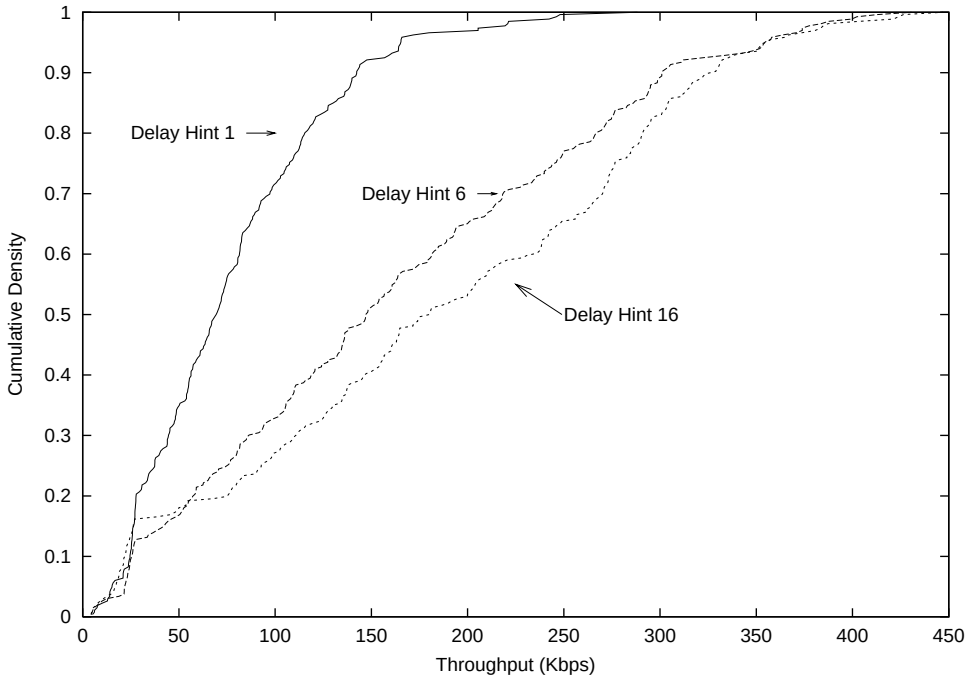

Figure 5.7: CDF of throughput of the video flow with delay hints of 1, 6 and 16.

5.4 Performance of TSQ over varying traffic mixes

The experiments conducted so far had one single delay sensitive flow (an audio flow in the first set of experiments and a video flow in the second set of experiments). We now evaluate the performance of TSQ when there is a varying mix of delay-sensitive and throughput-sensitive flows.

5.4.1 Experimental Setup

The experimental setup for this experiment is similar to the first set of experiments, i.e., we have a 15 Mbps bottle-neck link shared by 100 flows. Within these 100 flows, we changed the relative number of delay-sensitive (audio) flows with respect to throughput-sensitive (file transfer) flows. The delay-sensitive flows used a delay hint of 6 for all runs. The traffic mixes we ran include: 1 audio flow, 99 file transfer flows; 25 audio, 75 file transfer; 50 audio, 50 file transfer; and 75 audio, 25 file

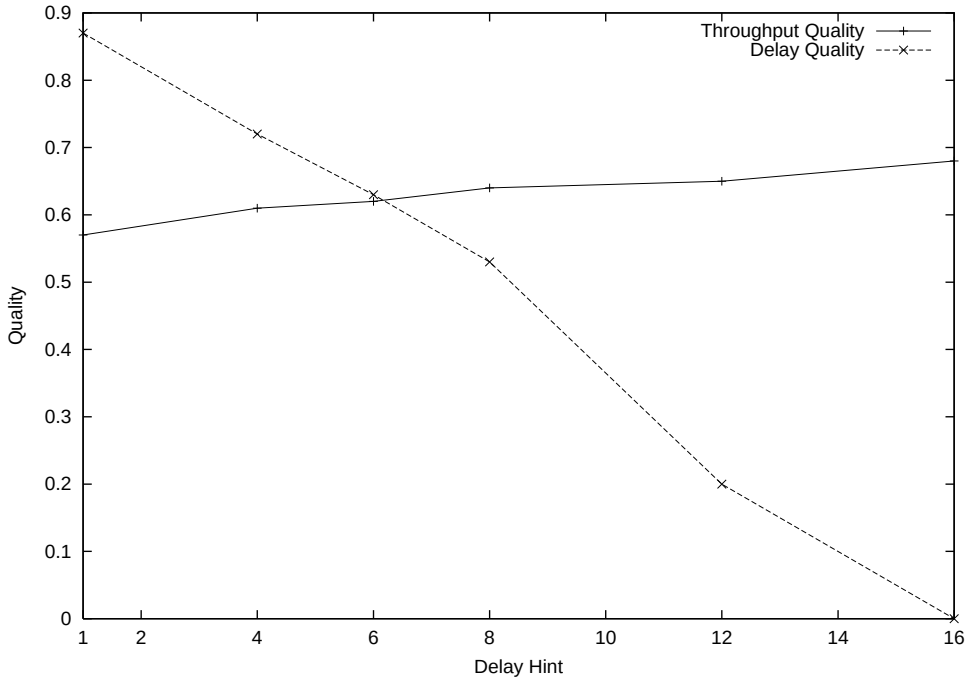

Figure 5.8: Delay and throughput quality of a single video flow versus delay hint

transfer[1]. The FTP flows are TCP flows using the maximum delay hint of 16.The audio flows are a TCP-friendly CBR sources sending data at a rate of 128 Kbps and using a delay hint of 6 (the optimum delay hint from Section 5.2).

5.4.2 Analysis

We calculated the average quality obtained by file transfer flows and audio flows for the various traffic configurations. This quality was then normalized against the quality that the application obtained when TSQ was switched off. In other words, the normalized quality of an application when TSQ is switched off is 1. If it gets better quality when TSQ is switched on, then its normalized quality is greater than 1. Conversely if the quality of the application is worse when TSQ is switched on,

[1]The extreme case of 99 audio flows and 1 file transfer flow was not evaluated, as this configuration did not cause sufficient queue build-up and hence was not useful for comparitive evaluation.

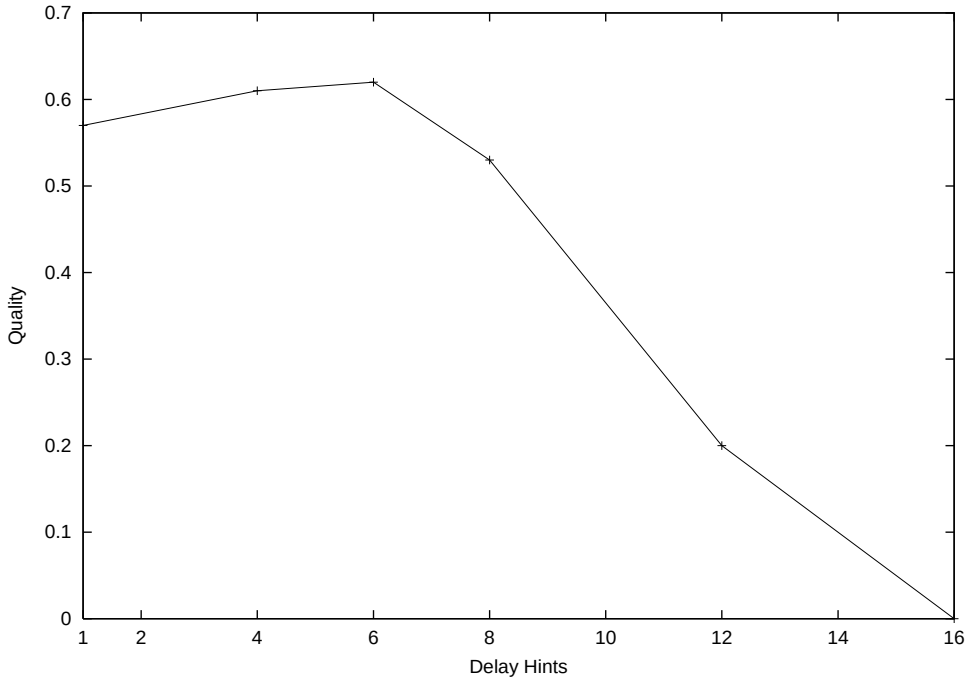

Figure 5.9: Quality of a single video flow versus delay hint

then normalized quality is less than 1.

Figure 5.10 summarizes the results obtained by this experiment. It can be seen that as the percentage of audio flows in the network increases, the average gain in quality of the audio application decreases. This is because as the number of delay-sensitive flows increase in the network, they will cut in line less than they would with fewer such flows. Hence the advantage gained in decreasing the queueing delay will be reduced. However, at all times the normalized quality is greater than 1. Hence, the quality of service obtained using TSQ is always higher than that obtained without TSQ even with increasing numbers of audio flows.

For the file transfer flows, the normalized quality increases initially with an increase in number of flows. However, as the number of audio flows increases beyond 25 percent, the normalized file transfer quality starts decreasing. However for all traffic mixes, the normalized file transfer quality is greater or equal to 1. Thus, TSQ

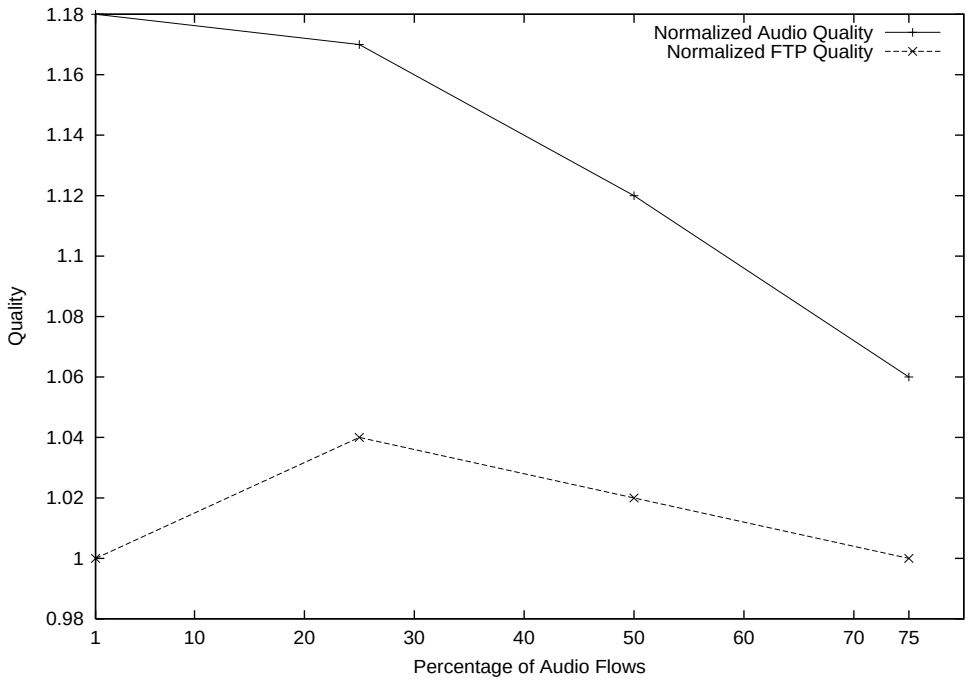

Figure 5.10: Normalized Quality of Audio flows and FTP flows with varying traffic mixes

provides better or equal quality for both audio and file transfer applications than does the underlying AQM (PI in our experiments) without TSQ.

5.5 Unresponsive Flows

In this section we evaluate the behavior of unresponsive flows when TSQ is used. A TCP based application will halve its congestion window if a packet is dropped. However an unresponsive UDP application will not reduce its rate of sending data in response to packet loss. Hence, we investigated whether unresponsive UDP flows can gain an unfair advantage due to the presence of TSQ. In the first set of experiments, we introduced a single unresponsive UDP flow in a network with only file transfer TCP flows. We observed the effect of the UDP flow on the average throughput of the TCP flows. We repeated the experiment with different values for the delay hints

for the UDP flow.

Current Internet has many interactive media flows using UDP. Hence, in the second set of UDP experiments we evaluated the effect on quality of UDP and TCP applications with varying mixes of UDP and TCP flows. The quality of these applications were normalized against the quality achieved under similar network conditions if TSQ was not used.

5.5.1 Experimental Setup 1

The network topology is similar to those in previous experiments. We have 99 bulk file transfers using TCP, and 1 audio flow over UDP sharing a 15 Mbps link. The audio flow is a CBR traffic source sending data at a rate of 600 Kbps, which is more than the flow's fair share of bandwidth of 150 Kbps. The file transfer flows are delay-insensitive and hence have the maximum delay hint of 16. The UDP flows used a different delay hint in each run. We ran each simulation for 100 seconds.

5.5.2 Analysis 1

We measured the average throughput for the 99 file transfer flows in each run. Figure 5.11 shows the average file transfer throughput when running with UDP flows with different delay hints. The throughput is normalized against the average file transfer throughput when the same experiment ran on PI without TSQ. As we can see from the graph, the average file transfer throughput remains almost constant in each of the runs and the file transfer throughput is actually a little less affected by the UDP flow under TSQ. Hence, we can say that by using a lower delay hint, the unresponsive flows will not gain any additional advantage due to TSQ. This makes AQM routers that use TSQ no more vulnerable to unresponsive flows than if they did not use TSQ.

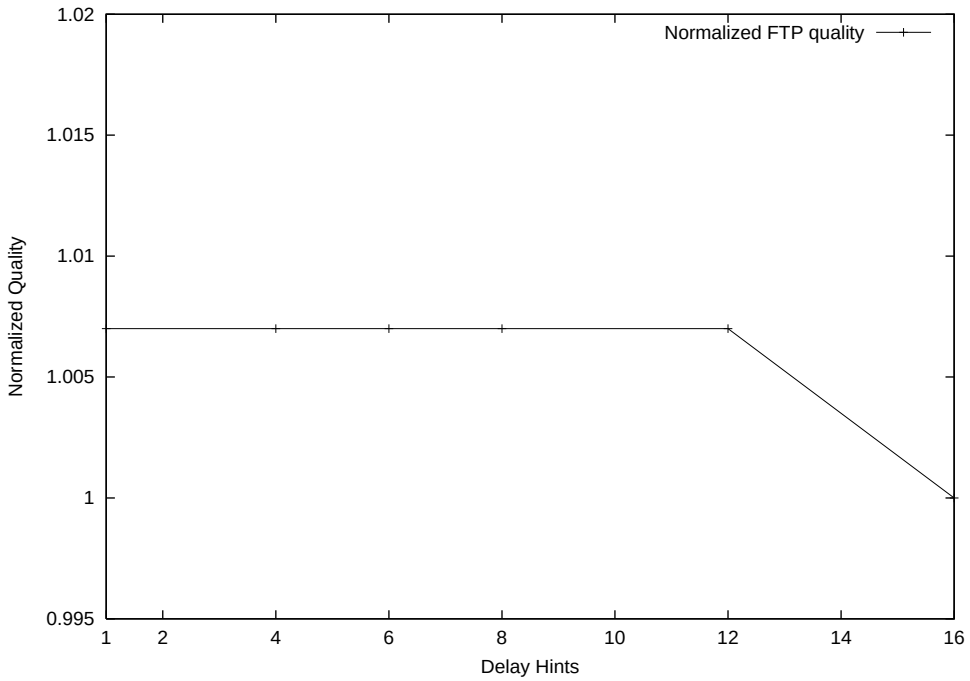

Figure 5.11: Normalized file transfer throughput versus delay hints

5.5.3 Experimental Setup 2

A total of 100 flows share a 15 Mbps link with a one-way propagation delay of 150 ms. The 100 flows comprise of a mix of unresponsive audio flows running over UDP and file transfers running over TCP. We vary the mix of UDP flows from 1 to 75 (1, 25, 50 and 75). The audio flows use a delay hint of 6 and the TCP flows use a delay hint of 16.

Analysis

Figure 5.12 shows a plot of the average audio and file transfer quality normalized against the average quality obtained without TSQ. As the number of UDP audio flows in the network increases the normalized quality decreases for both the UDP and file transfer applications. However, at all times the normalized quality is above 1 for both UDP audio and TCP file transfer. Hence, there is an improvement in

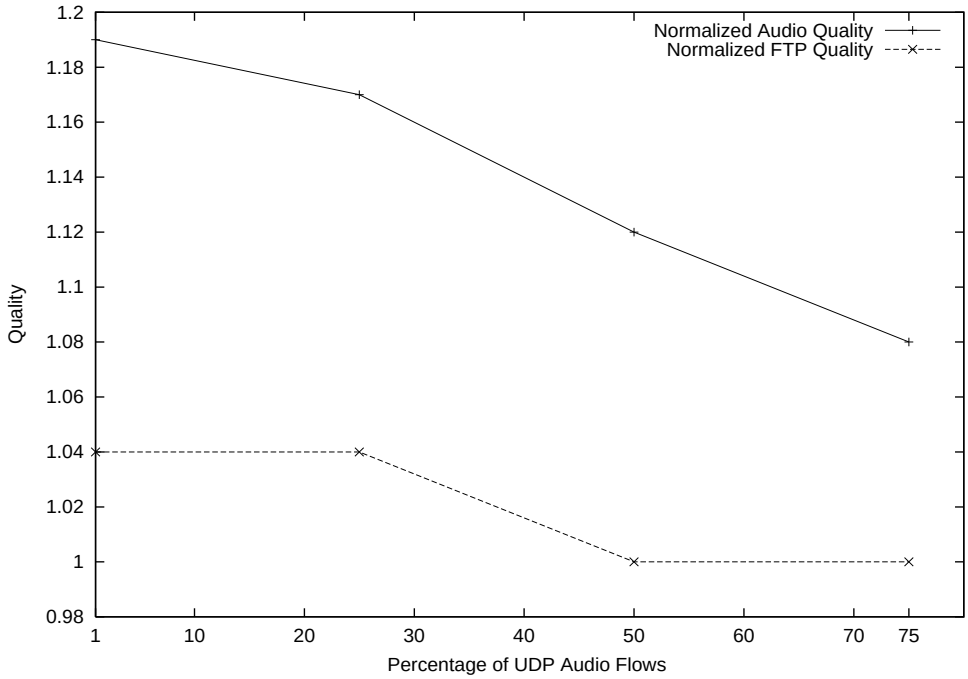

Figure 5.12: Quality Variation with UDP flows

the average quality of both the UDP audio and TCP file transfer applications due to the TSQ for a varying mixes of flows, suggesting TSQ will not more negatively impact network performance in the presence of unresponsive flows.

Chapter 6

Future Work

Our current implementation of TSQ uses 4 bits in the IP header to embed the delay hint. This allows applications to choose from 16 levels of delay senstivity. A larger range of delay hints will be available if more bits are used to embed the delay hint, but at the cost of more bits of overhead. Hence, further research is required to determine the number of bits needed to support a sufficient sensitivity without inducing unnecessary overhead.

Another area of potential future research is in developing quality metrics. We have made an attempt to provide quality metrics representative of three applications (interactive audio, interactive video and file transfer). However, if other applications need to take advantage of TSQ, they must have knowledge of their quality requirements. In addition, there may be other ways to quantify QoS, such as taking the average (or the sume) of the throughput and delay qualities. Hence, further investigation into the quality metrics and requirements of other applications on the Internet is required to support the full range of applications of today's Internet. This

will allow the evaluation of TSQ performance with a wider variety of applications.

Another possible extension would be to build applications that can take advantage of TSQ by dynamically changing their delay hints. These applications can evaluate the quality that they obtained by using their current delay hint and can adapt their delay hint if they are not satisfied with the QoS received. How rapidly an application adapts to change network QoS would be an open issue.

Chapter 7

Conclusions

The current Internet supports applications with varying Quality of Service (QoS) requirements. The QoS of the application primarily depends on two factors: 1) the delay, and 2) the throughput. Emerging applications, such as interactive multimedia, are delay-sensitive and hence their quality is affected more by changes in network delay than by changes in network throughput. Traditional applications, such as file transfer, are throughput-sensitive and their quality is almost solely dependent on the throughput that they obtain. Unfortunately, the current Internet however does not distinguish between the differences in application QoS requirements and instead provides uniform service to all applications. We assert that the Internet can instead provide QoS mechanisms while remaining best effort, raising the overall QoS for most applications, while preserving the robustness and scalabilty of the network.

In this thesis, we have presented a Traffic Sensitive QoS controller (TSQ). TSQ is sensitive to the varying QoS requirements of diverse Internet traffic, and thus provides different delay and throughput treatments to packets from different types of applications. TSQ can be used in conjunction with many current AQM techniques

allowing the full performance benefits to quality that the underlying AQM has to offer. Applications inform TSQ about their delay sensitivity by embedding within each packet a delay hint, an indicator of an application's delay-sensitivity. Based on the delay hint of each packet, TSQ makes a decision as to where the packet must be inserted in the queue (thus potentially decreasing its queueing delay) and how much the drop probability of the packet must be increased (thus potentially decreasing its throughput). This mechanism helps delay-sensitive applications attain better QoS, while at the same time avoids hurting the QoS of throughput-sensitive applications.

In order to quantify an application's QoS, we propose a QoS metric based on the minimum of an application's delay quality and throughput quality. We have also presented quality metrics for some typical Internet applications: interactive audio, interactive video and file transfer. Based on previous perceptual quality research, we have devised quality functions for these applications in terms of delay and throughput. Thus, with TSQ, applications such as these, with knowledge of their QoS requirements can dynamically choose their delay hints so as to maximize their Quality of Service.

We have implemented TSQ in ns-2, in conjuction with the PI-Controller [HMTG01]. We have tested TSQ to measure its effect on the quality of interactive audio, interactive video and file transfer applications. We have also evaluated TSQ by varying the proportion of delay-sensitive and throughput-sensitive flows in the traffic mix. Our evaluation of TSQ with varying traffic mixes shows TSQ can increase the average quality of all applications (8% to 18% for delay sensitive applications and up to 4% for throughput sensitive applications) over the quality obtained by using the AQM without TSQ. We have also evaluated the performance of TSQ when there are unresponsive flows in the traffic mix. The results obtained show that TSQ does

not allow unresponsive traffic to gain further advantage over responsive traffic than does the underlying AQM.

TSQ is still a best-effort service and thus requires no policing, traffic monitoring, or per-flow information. The RED-Boston algorithm presented in [PCK02] enhances the ARED [FGS01] by providing a per-packet QoS to Internet applications. It also uses delay hints and "cut-in-line" mechanism. However our approach extends the work done in [PCK02] as follows:

1. We developed QoS metrics based on delay and loss. The overall quality is defined as the minimum of the delay quality and throughput quality. We have defined the quality metrics for 3 applications: interactive audio, interactive video and file transfer.

2. We formally defined the relationship between the decrease in queueing delay and the corresponding increase in drop probability to maintain TCP "fairness". We have derived the necessary formulas based on TCP steady state throughput.

3. We decoupled the QoS controller from the underlying AQM. This allows TSQ to be applied in conjunction with many existing AQM techniques. In this work, we have implemented it on top of the PI-Controller.

Bibliography

[ALLY01] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. REM: Active Queue Management. *IEEE Network*, May 2001.

[BFC93] T. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT) an Architecture for Scalable Multicast Routing. In *Proceedings of ACM SIGCOMM Conference*, pages 85 – 95, September 1993.

[Car02] Carey Williamson and Qian Wu. A Case for Context-Aware TCP/IP. *ACM Performance Evaluation Review*, 29(4):11 – 23, March 2002.

[CC00] Jae Chung and Mark Claypool. Demonstration of Dynamic Class-Based Router Queue Management. In *Proceedings of the ACM Multimedia Conference*, November 2000.

[Cor98] Real Networks Corporation. Real Networks Guide for Audio Production, 1998.

[Cor00] RadVision Corporation. Multipoint Conferencing Specifications, 2000.

[CPS02] Andrew Corlett, D. I. Pullin, and Stephen Sargood. Statistics of One-Way Internet Packet Delays. In *Proceedings of 53rd Internet Engineering Task Force*, March 2002.

[DCJ93] Spiros Dimolitsas, Franklin L. Corcoran, and John G. Phipps Jr. Impact of Transmission Delay on ISDN Videotelephony. In *Proceedings of Globecom*, pages 376 – 398, November 1993.

[FF99] Sally Floyd and Kevin Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, February 1999.

[FGS01] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. Under submission, 2001.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[FKSS01] 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.

[Flo] Sally Floyd. References on CBQ (Class-Based Queueing). Internet document.
http://www-nrg. ee.lbl.gov/floyd/cbq.html.

[Gan] Implementation Architecture Specification for the Premium IP Service, 2002. Gn1(Gant) deliverable d9.7-addendum 1. http://archive.dante.net/geant/public-deliverables/GEA-02-079v2.pdf.

[GM01] Liang Guo and Ibrahim Matta. The War Between Mice and Elephants. In *9th IEEE International Conference on Network Protocols*, November 2001.

[HBWW99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. *IETF Request for Comments (RFC) 2597*, June 1999.

[HKBT01] P. Hurley, M. Kara, J. Le Boudec, and P. Thiran. ABE: Providing a Low Delay within Best Effort. *IEEE Network Magazine*, May/June 2001.

[HMTG01] C. V. Hollot, Vishal Misra, Donald F. Towsley, and Weibo Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. In *Proceedings of INFOCOM*, pages 1726–1734, 2001.

[IKK93] Satoru Iai, Takaaki Kurita, and Nobuhiko Kitawaki. Quality Requirements for Multimedia Communication Services and Terminals-interaction of Speech and Video Delays. In *Proceedings of Globecom*, pages 394 – 398, November 1993.

[JNP99] V. Jacobson, K. Nichols, and K. Poduri. Expedited Forwarding PHB Group. *IETF Request for Comments (RFC) 2598*, June 1999.

[KS01] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue. In *Proceedings of ACM SIGCOMM*, August 2001.

[NT02] Wael Noureddine and Fouad Tobagi. Improving the Performance of Interactive TCP Applications using Service Differentiation. In *Proceedings of IEEE Infocom*, June 2002.

[Par01] Mark Anthony Parris. Class-Based Thresholds: Lightweight Active Router-Queue Management for Multimedia Networking, 2001.

[PCK02] Vishal Phirke, Mark Claypool, and Robert Kinicki. Traffic Sensitive Active Queue Management for Improved Quality of Service. Technical Report WPI-CS-TR-02-21, CS Department, Worcester Polytechnic Institute, May 2002.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. In *Proceedings of ACM SIGCOMM*, 1998.

[Pug90] William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[SBC94] S. Shenker, R. Braden, and D. Clark. Integrated Services in the Internet Architecture: An Overview. *IETF Request for Comments (RFC) 1633*, June 1994.

[SSZ98] Ion Stoica, Scott Shenker, and Hui Zhang. *Core*-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM Conference*, September 1998.

[SZ99] Ion Stoica and Hui Zhang. Providing Guaranteed Services Without Per Flow Management. In *Proceedings of ACM SIGCOMM Conference*, September 1999.

[Zeb93] J.A. Zebarth. Let Me Be Me. In *Proceedings of Globecom*, pages 389 – 393, November 1993.