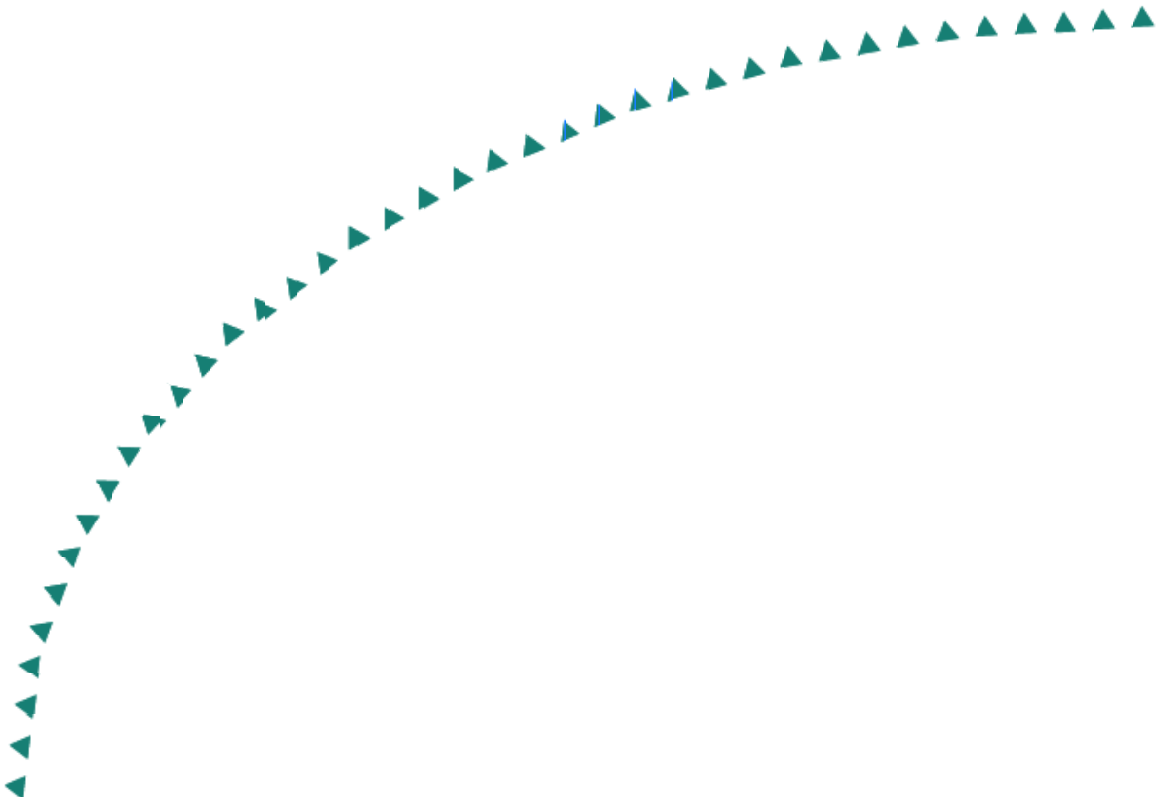
2004-34

Final Report

Implementation of Quality of Service (QoS) with Dynamic Resource Allocation



Research

Technical Report Documentation Page

1. Report No. MN/RC-2004-34	2.	3. Recipients Accession No.	
4. Title and Subtitle Implementation of Quality of Service (QoS) with Dynamic Resource Allocation		5. Report Date March 2004	
		6.	
7. Author(s) Vladimir Cherkassky, Harry Rostovtsev		8. Performing Organization Report No.	
9. Performing Organization Name and Address University of Minnesota Department of Electrical and Computer Engineering 200 Union Street SE Minneapolis, Minnesota 55455		10. Project/Task/Work Unit No.	
		11. Contract (C) or Grant (G) No. (c)81655 (wo)38	
12. Sponsoring Organization Name and Address Minnesota Department of Transportation Research Services Section 395 John Ireland Boulevard Mail Stop 330 St. Paul, Minnesota 55155		13. Type of Report and Period Covered Final Report	
		14. Sponsoring Agency Code	
15. Supplementary Notes www.lrrb.org/PDF/200434.pdf			
16. Abstract (Limit: 200 words) As the need to monitor traffic conditions on highway systems increases, the ability to get a clear video picture decreases due to network congestion. There are many Quality of Service (QoS) implementations available on the market for high traffic networks but none implement dynamic priority assignment changes to different network traffic. This is very useful for highway video surveillance using camera systems attached to a limited bandwidth network. This project uses freely available software and low cost hardware to provide such a system. A working prototype of the system has been developed and a detailed performance analysis has been performed.			
17. Document Analysis/Descriptors Quality of Service QoS Dynamic bandwidth allocation		18. Availability Statement No restrictions. Document available from: National Technical Information Services, Springfield, Virginia 22161	
19. Security Class (this report) Unclassified	20. Security Class (this page) Unclassified	21. No. of Pages 68	22. Price

Implementation of Quality of Service (QoS) with Dynamic Resource Allocation

Final Report

Prepared by:

Vladimir Cherkassky
Harry Rostovtsev
Department of Electrical Engineering
University of Minnesota

March 2004

Published by:

Minnesota Department of Transportation
Research Service Section
Mail Stop 330
395 John Ireland Boulevard
St. Paul, Minnesota 55155-1899

Acknowledgements

I would like to thank Professor Vladimir Cherkassky for his help and time on this project. This project was possible only with his guidance.

I would sincerely thank Ted Morris, the Intelligent Transportation System laboratory (ITS) manager, for providing equipment and space in ITS lab, not to mention his immense knowledge base.

I would also like to thank Prakash Balasubramanium and Akash Malhotra, my predecessors on this project for doing the initial research and setup.

Finally, I would like to thank Ray Starr at Minnesota Department of Transportation (Mn/DOT). His feedback about the project has been the motivator for many additions as well as subtractions from the project.

Table of Contents

List of Figures

List of Tables

Executive Summary

In today's high information rate age, network usage is becoming more dominant. More and more applications are using wireless technology or other limited bandwidth network connections. Digital video transmission and recording from surveillance applications over Internet Protocol (IP) are becoming increasingly popular. One such example of digital video transmission is using wireless IP network to transfer live video of traffic from highways. With an increasing number of video cameras on freeways/highways and each camera demanding a high portion of bandwidth, speed and quality of transmission are becoming important issues in such networks. Network congestion results in delays and loss of data packets. This report is the result of an investigation of a possible solution to network congestion using Quality of Service (QoS).

Traffic prioritization is one of the best ways to provide QoS. This technique is flexible because one can use different prioritization schemes depending on the application. It is possible to allocate a fraction of available bandwidth for the use by different kinds of applications. In order to achieve traffic prioritization, the ALTQ open source software package was used. This software package, in conjunction with the FreeBSD operating system, converts a normal computer into a router. This system is flexible, cost-effective, and open sourced. Since the system is dynamic, it is scalable and is able to support a high number of different network streams. It is limited only by the total network bandwidth and the hardware on which it runs. The system is freely available for download even for commercial uses. The system is also under constant development. If there is a bug or a desired feature missing, it will be fixed or added at a later date or an experienced programmer can fix the problem if necessary.

The goal of this project is the implementation of the traffic prioritization and bandwidth allocation (implemented in software using ALTQ) in the prototype system at Intelligent Transportation System (ITS) Lab. The network needs to be able to support several streams of data, a camera control channel, and other low priority traffic. When the need arises, for example viewing traffic on the highway where an accident has occurred, the user should be able to give higher priority and bandwidth to the corresponding video stream even if it means borrowing network resources from the other traffic.

The Minnesota Department of Transportation (Mn/DOT) networks were studied and some bottlenecks were identified. Most of the bottlenecks involved wireless links to remote sites containing several video surveillance cameras. Other bottlenecks can be observed at points where one type of network connects to another type of network. These points would most benefit from network traffic management through QoS. These bottlenecks were modeled by a simple network prototype setup in the ITS Lab.

The prototype of this QoS system was set up using two computers. The first computer was used for packet marking or coloring. The second computer was used for the filtering of the packets based on the markings from the first computer. This system was later refined to fit on one computer, which in turn allowed the extension of the system to bidirectional QoS system. The bidirectional system provides a higher level of control over the network data, allowing for better traffic filtering and bandwidth resource allocation.

The two computers were setup with the FreeBSD 4.6 operating system with a patched kernel to allow the use of ALTQ software. Each computer has two network interfaces which are configured to be used with the ALTQ software. The computers are connected in serial network configuration, forming a 10 Megabit per second (Mb) bottleneck between the two systems. Each computer is on a separate subnet of the network and the packets are routed using Network Address Translation (NAT) and packet forwarding. The network is set up in a way that allows direct IP connection to each of the cameras by a mapped IP address.

Priority assignment and bandwidth allocation are done in a two – step process. The first step is to assign the priority to the incoming network packets on the first network interface on the computer. The second step is to filter the packets by their priority based on a predefined set of filters and to send the packets out in the new order on the second network interface. The filters are statically assigned. This requires knowledge of all possible types of traffic and the desired range of bandwidth for each type of traffic.

Three different network setups were constructed for this model. The first setup, named QOS0, used no priority scheme and relied only on the network card driver to handle network data queuing. The second setup, named QOS1, gave priority only to the traffic coming from the video cameras but controlled no traffic in the opposite direction. The third and final setup, named QOS2, setup priority to traffic coming from the video cameras and traffic going back to the video cameras. These different network setups were tested with three video streams from the cameras (high priority network data) and extra traffic generators (low priority network data). The cameras at Mn/DOT use Pan/Tilt/Zoom (PTZ) controllers which receive signals from the camera operators located at the Mn/DOT office locations. To accommodate for this camera control, the prototype was modified to include a camera control channel. This was the main motivation for bidirectional QoS setup.

Analysis of the model, conducted using different network setups, showed how the system handles high traffic loads and priority/bandwidth allocation. A detailed study of the data showed that the prototype gives a definite advantage over uncontrolled network data. In the worst case scenario, a comparison between QOS0 and QOS2 setups reveals that the video frame rate is 4.5 times faster and the delay for the camera control channel is 6 times less when using QoS. In all comparisons, the setups with QoS produced a higher quality video in terms of frame rate and lower delay for packets on the camera control channel than the setup without QoS.

The QoS system developed at the ITS Lab can be ported to the Mn/DOT network without any major difficulties. While the system is not transparent due to the addition of subnets, it would not interfere with any existing hardware. Since Mn/DOT uses a system designed by ADDCO to view the videos from their remote sites, there would need to be modifications to incorporate priority assignment into the ADDCO system. This is not necessary since the priority assignment can be done separately without any modifications to the ADDCO system already in place.

The hardware being used for the prototype at the ITS Lab are bulky PIII 700MHz computers. These machines are large, not weather proof, and use high amounts of power. If the QoS is to be used out in the field, especially at remote rural sites, a different packaging system would have to be designed. One option is to use weather-proofed Micro-ITX or Mini-ITX cases with less powerful processors and no hard disks. The FreeBSD kernel and configuration files for ALTQ

would reside in flash memory. This would decrease both the size and the power requirements for the QoS field units.

The QoS field units should be placed at the remote end of any slow network link or other bottlenecks of the Mn/DOT network. At the office site, one large server type QoS unit with several network connections can interface with the several field units at the same time. This would decrease the amount of space needed for the QoS system in the server room as well as ease the configuration and control of the QoS system.

Overall, the author believes that Mn/DOT would greatly benefit from the use of QoS in its network, especially at bottlenecked remote sites. The price of increasing the network bandwidth to remote rural areas is much higher than the cost of implementing the QoS system. The QoS system is scalable and is only limited by the hardware that it runs on and the total bandwidth. QoS would not only relieve the network congestion for the time being, it can be used after the bandwidth has been increased in order to accommodate more total traffic.

Chapter 1. Introduction

The rise of Internet Protocol (IP) as a foundation for a universal network raises several issues for large enterprise networks, not the least of which is how to guarantee that applications will receive the service levels they require to perform adequately across the network. For example, network administrators might need to define a low level of latency and packet loss to ensure that a large file or a business-critical traffic flow gets to its destination on time and without delay. Or, they may need to ensure that a real-time session such as voice or video over IP doesn't look choppy or out of sequence.

The problem with IP is that it does not guarantee bandwidth. Specifically, the protocol will not, in itself, differentiate network traffic based on the type of flow to ensure that the proper amounts of bandwidth and prioritization levels are defined for a particular type of application.

Because IP does not inherently support the preferential treatment of data traffic, it's up to network administrators to make their network components aware of applications and their various performance requirements.

Quality of Service (QoS) encompasses bandwidth allocation, prioritization, and control over network latency for network applications. There are several ways to ensure QoS. The easiest way is simply to throw bandwidth at the problem until service quality becomes acceptable. This costly approach might involve upgrading the backbone to a high-speed technology. If you have fairly light traffic, more bandwidth may be all you need to ensure that applications receive the high priority and low latency they require. However, this strategy collapses if the network is even moderately busy. In a complex environment - one that has a lot of data packets moving in many paths throughout the network, or that has a mixture of data and real-time applications - you could run into bottlenecks and congestion.

Also, simply adding bandwidth does not address the need to distinguish high-priority traffic flows from lower-priority ones. All traffic is treated the same. Some applications such as video over IP would still be affected by the lack of prioritization due to the large amount of data being transferred. Additional bandwidth can solve some of the short-term problems, but it's not a viable long-term solution, particularly if there is already enough bandwidth to accommodate all but the most highly sensitive network applications.

This report is an investigation into the viability of QoS for the Minnesota Department of Transportation (Mn/DOT) networks for improving the quality and speed of video transmissions from remote sites. Chapter 1 (current chapter) covers the background of QoS and the organization of the rest of the report. Chapter 2 discusses network configurations used for the prototype and how they relate to the network bottlenecks observed at Mn/DOT networks. Chapter 3 covers installation, configuration, and implementation of the QoS system using the ALTQ software package. It also provides the step-by-step instructions for installing the ALTQ software package and modifying the FreeBSD kernel to utilize it. Chapter 4 provides an in-depth coverage and analysis of ALTQ QoS performance based on experimental data gathered from the prototype at the ITS Lab. Chapter 5 provides recommendations for porting the system to Mn/DOT networks and the project conclusions. Chapter 6 is the conclusion to the document. The last document this chapter provides is a README file for running the existing prototype

system. In conjunction with the rest of the provided documents, Mn/DOT should be able to fully recreate the existing prototype system that is in place at the Intelligent Transportation System (ITS) Lab and build upon that.

Chapter 2. Network Configurations

QoS issues gain importance in cases of wireless network links and other Local Area Networks (LANs) where the available bandwidth is less than the required bandwidth or the network is generally busy. At present, the Mn/DOT networks have sufficient bandwidth to carry most of its real-time multimedia data. However, a bandwidth constraint might arise when there is a need to increase the number of cameras. More cameras mean more video data and more bandwidth. Even if not all the bandwidth is being utilized, the addition of extra traffic to the network will increase the time it takes for the data to make it across the network, causing loss of quality of streamed video. One possible solution is to increase the bandwidth throughout the whole network to accommodate the extra traffic. This approach, however, is usually fairly expensive, depending on the size of the network and the distance to remote sites. Some portions of Mn/DOT's network cover fairly large distances in order to reach remote rural areas. Increasing the bandwidth across such a large distance is quite costly. Some of such bottlenecks have been identified in the Mn/DOT network. Figure 2.1 illustrates a possible scenario where QoS implementations would be useful.

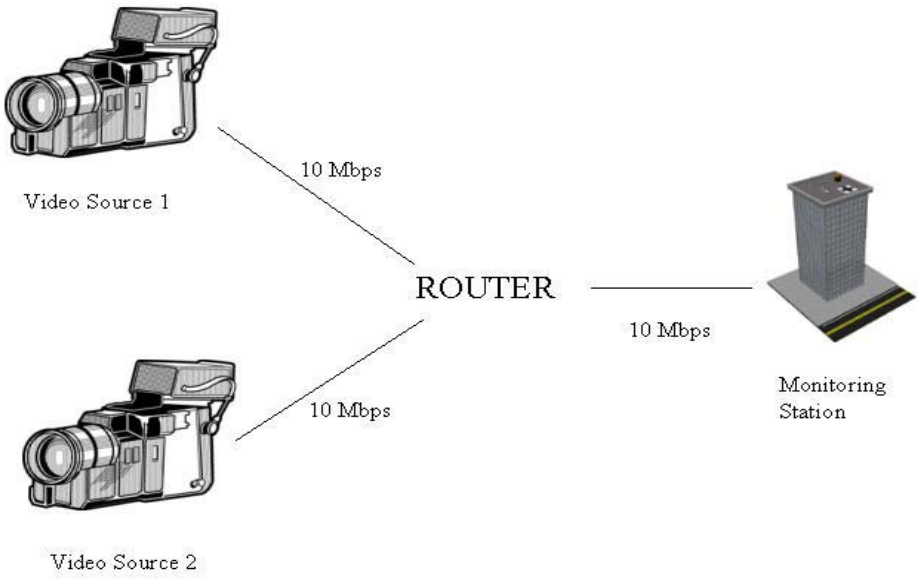


Figure 2.1 - Possible Bottleneck in the Network

Figure 2.1 illustrates a scenario which has multiple data sources operating on 10Mbps links. Both sources are routed to a single outgoing line of bandwidth 10Mbps. In such a scenario the input data sources can easily exceed the bandwidth of the outgoing link. Here, the router can be used to prioritize the traffic, allocate bandwidth, and control the delay of different applications in times of congestion. This simple example can be extended to include more video sources as

needed as well as more monitoring stations. The basic concept remains the same: there is more data trying to traverse the network than capacity allows for. Figure 2.2 below shows a portion of the Duluth TOCC network diagram.

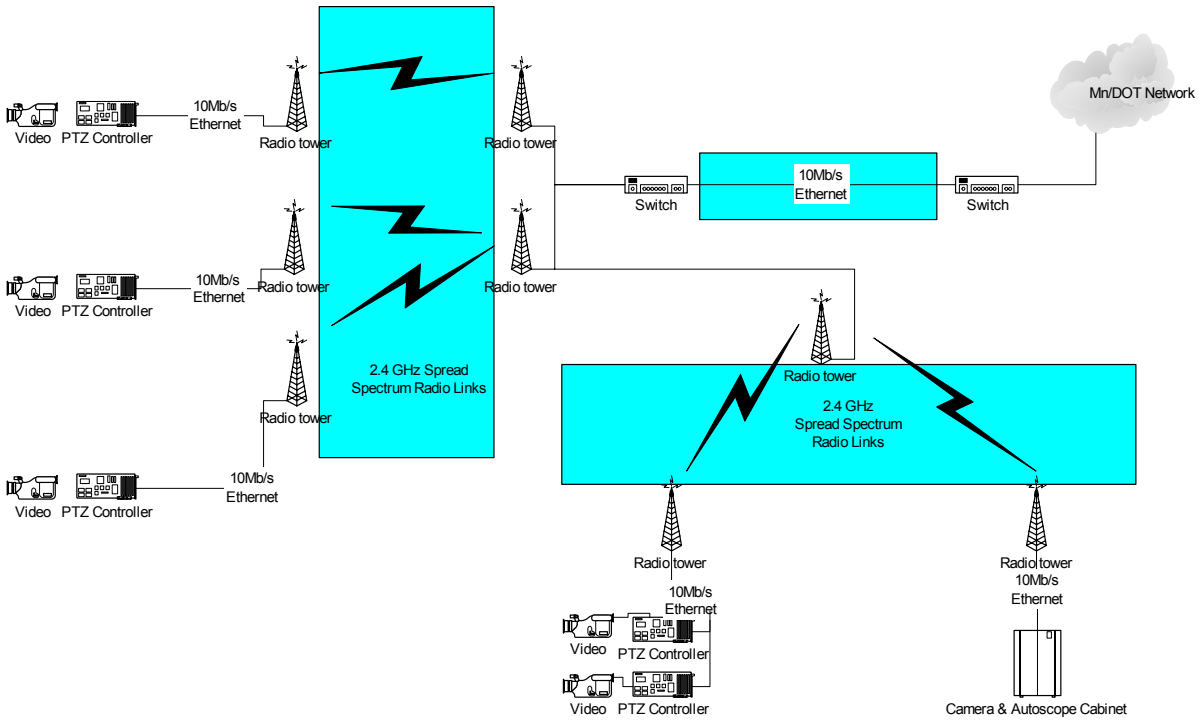


Figure 2.2 - Possible Bottlenecks in TOCC Duluth Network

The above figure shows one example of a network bottleneck where even a light load might lead to loss of video quality and a large increase in packet delay. In Figure 2.2, the shaded portions on the left and bottom right have a collective bandwidth of 50Mbps and the shaded portion on the top right only has the capacity to handle 10Mbps. Even if the cumulative bandwidth required by the shaded portion in the left is less than 10Mbps, there will be noticeable delay in transfer speeds (packet delay). As the network connection on the left approaches approximately 70%, applications start to compete for bandwidth. The results of this competitive behavior are very erratic. If one of the applications is using a lot more bandwidth than other applications, a live video stream for example, the "greediest" application will suffer worse than others due to the IP protocol's fairness. If the bandwidth use in the shaded portion on the left continues to increase, the situation in the shaded portion on the right deteriorates very rapidly at an un-linear rate. As more applications compete for the available bandwidth, each one receives less than the amount required to be useful. An example of this can be seen in Figure 2.3 below.

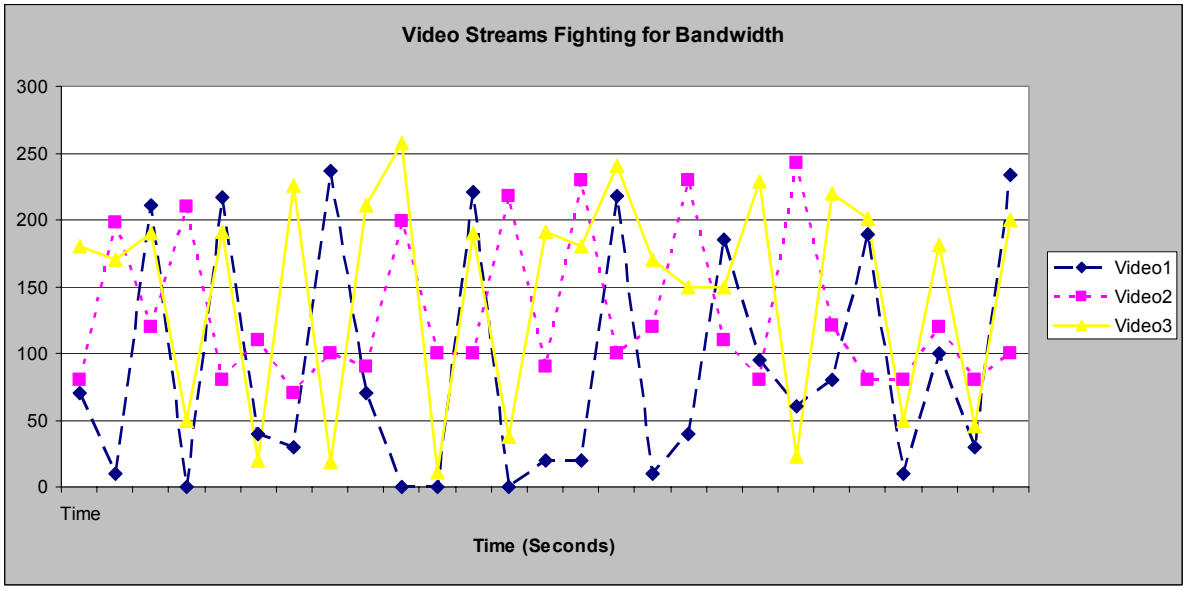


Figure 2.3 - Competitive Behavior in Network Video Streams

In Figure 2.3 shown above, three "greedy" applications are competing for 10Mbps of network bandwidth. Each stream is approximately 2.5Mpbs but the 10Mbps pipe is completely maxed out at 7.5Mbps. This is due to a 10Mbps being the theoretical limit of the network link. The actual throughput depends on many factors, such as the quality of the network switch being used, grade of cabling, distance between network switches, etc. If the above network connection was under control of QoS, the user could raise the priority of the important stream, while depriving less important data streams of bandwidth. This would raise the quality of the important data and lower the quality of the less important network traffic. Also, it would prevent the competitive behavior demonstrated earlier in Figure 2.3. The resulting network load (shown in Figure 3.4) does not exhibit this fight for bandwidth and when the network connection does get maxed out, it does so with more efficiency.

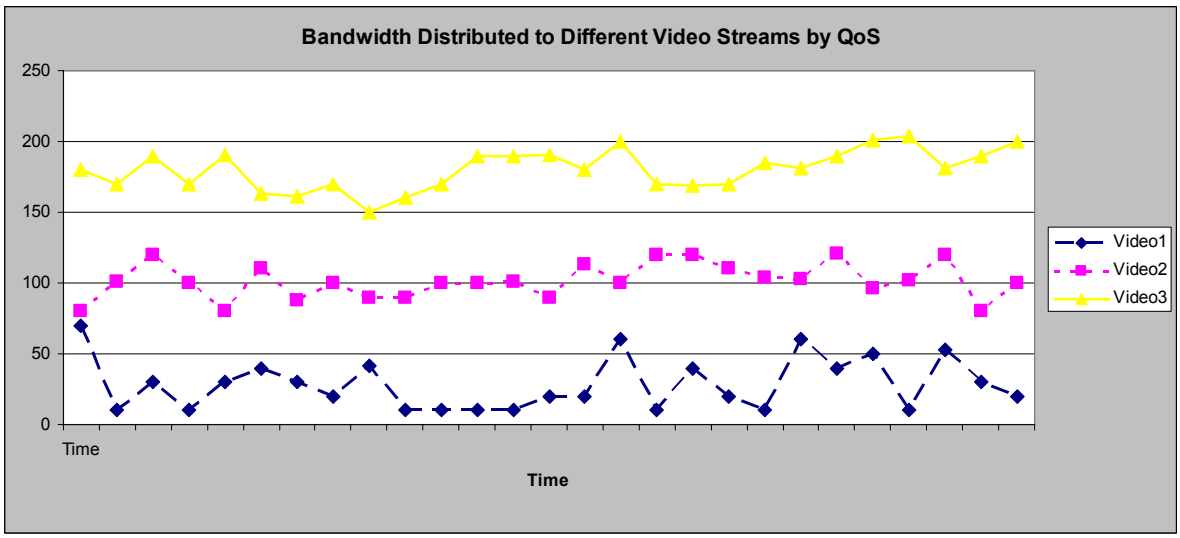


Figure 2.4 - No Competitive Behavior

As can be seen in Figure 2.4, the Video1 has the high priority so it gets its required amount of bandwidth with no other applications encroaching on it. The other two video streams have low priority and are allowed much less bandwidth than the high priority stream. This results in improvement of video quality for the high priority video stream and lowered quality of video in the other two streams. This simple example can translate directly to Mn/DOT networks. Where there is heavy network usage (exceeding approximately 70% of the total bandwidth), there will be a higher delay for data transmission and a lower overall throughput.

The QoS prototype is a model of these bottlenecks since it is here that the network is most likely to suffer from network congestion. Note that not all bottlenecks were found in the Mn/DOT networks. This would be impractical since the only way to do so accurately is to know the usage of the particular network segment without any measurements. It is up to Mn/DOT network administrators to identify any congestion points as they would be most familiar with the network.

Chapter 3. Implementation, Installation, and Configuration of ALTQ

There are several commercial products available on the market that implement QoS for various applications. A close examination of commercial products like Packet Shaper from Packeteer Incorporation and QoSworks of Sitara Networks reveals that these products are based on time scheduling. For example, between 8 a.m. and 12 p.m. they will give priority to live traffic from certain cameras over others, or between 9 p.m. till 7 a.m. archived video from all cameras will be given equal priority in order to archive traffic data for research. Traffic situations on highways change frequently, and no commercial software provides a dynamic allocation of bandwidth to different network streams. Any changes in the prioritization policy of these QoS systems required a restart of the system, resulting in loss of packets. Accidents can occur at any time of the day, and a traffic surveillance system should be such that one can view the accident site video on the fly. Lack of dynamic bandwidth allocation in commercial QoS products was a big motivation for this project. Also, in commercial products, prioritization of traffic is based on source IP address only. For example, traffic from camera 1 will be given higher priority. This would allow anyone who views the video from camera 1 to have a good quality picture. None of the products described dealt with client/host side priority. An example of that would be if video 1 had high priority due to an accident or some emergency, we would want to be able to give certain people higher priority than others while watching that video. So State Patrol Officers would have higher priority than an employee checking the status of the highway traffic before they go home.

Thus, a solution was proposed in which the user has the ability to dynamically change priority to any cameras on the highway without any loss of packets. Apart from dynamic allocation of bandwidth to specific cameras, our solution also provides two – way prioritization of traffic. This means that the user has the ability to provide prioritization to traffic from specific cameras or to a specific viewer. As an example of prioritization from a specific camera, all users in a LAN can watch video from specific cameras without any delay in the video. On the other hand our system also provides prioritization to traffic for a client/host. For example, if the ITS Lab Manager wants to view certain applications, priority will be given to him or her over other employees.

In order to provide proof-of-concept, a prototype was developed in the ITS Lab. This prototype uses ALTQ software to implement QoS over a network that resembles the bottlenecks in the Mn/DOT network discussed earlier.

Section 3-1. ALTQ

ALTQ is a package that provides Alternative Queuing for BSD UNIX developed at the Sony computer science laboratories in Japan by Kenjiro Cho. At present, BSD UNIX implements only simple tails drop, which is basically a "first in first out" (FIFO) queuing scheme at the output buffers of network interface cards (NICs). The ALTQ package aims to provide a better queuing scheme, in order to realize resource sharing and the associated quality of service.

Subsection 3-1.1. Altqd process

Altqd is a process which configures the ALTQ box for user specified configuration. User configurations are stored in a file altq.conf. This is the default file name used by altqd. In our system, different filenames are used, and these are described in Appendix B. There are many configurations possible but this report will focus only on the ones used in the project. This project uses two configurations: Packet Conditioner and CBQ. These components are described in Subsection 3-1.2 through Subsection 3-1.4.

Subsection 3-1.2. Packet Conditioner

Packet conditioning or coloring is done by changing the Type of Service (TOS) bits of IP packet headers. There are some reserved values which are listed in Table 3-1 but the rest can be used for our purposes. However, after some trial-and-error and some research into the subject, the TOS bits can only be set to certain values. The most significant nibble can be set to a range of 0 to F (in hexadecimal) but the least significant value can only take on 0, 4, 8, and C.

Table 3-1 - Usable IP TOS bits

Usable IP TOS bits							
	0x10	0x20	0x30	0x40	0x50	0x60	0x70
0x04	0x14	0x24	0x34	0x44	0x54	0x64	0x74
0x08	0x18	0x28	0x38	0x48	0x58	0x68	0x78
0x0C	0x1C	0x2C	0x3C	0x4C	0x5C	0x6C	0x7C
0x80	0x90	0xA0	0xB0	0xC0	0xD0	0xE0	0xF0
0x84	0x94	0xA4	0xB4	0xC4	0xD4	0xE4	0xF4
0x88	0x98	0xA8	0xB8	0xC8	0xD8	0xE8	0xF8
0x8C	0x9C	0xAC	0xBC	0xCC	0xDC	0xEC	0xFC

Table 3-2 shows the reserved TOS bits. Masks 0x00, 0x01, 0x02 cannot be set but 0x04, 0x08, and 0x10 can.

Table 3-2 - Reserved IP TOS Bits

TOS	mask	Suggested Use
Minimum Delay	0x10	ftp, telnet, ssh
Maximum Throughput	0x08	ftp-data, www
Maximum Reliability	0x04	SNMP, dns
Minimum Cost	0x02	nntp, SMTP

The actual number of TOS bit combinations that this project can use is even less than that. Since some are too common, like 0x08 or 0x10, they cannot be used either. This leaves a total of 60 combinations that can be used to mark packets. This also means that there can be no more than 60 IP addresses on each ALTQ interface being marked.

Doing the actual marking of the packets is relatively simple with ALTQ. A sample configuration file for Packet Conditioner (CNDR) is shown in Figure 3.1.

```
#/etc/altq.conf

 #define the interface, ingress for packet conditioning
interface xl0

#define the action, mark with 0x40
conditioner xl0 marker <mark 0x40>


#define which packets to mark, in this case all
#packets from IP address 172.16.31.2
filter xl0 marker 0 0 172.16.31.2 0 0
```

Figure 3.1 - Sample Packet Conditioner

Subsection 3-1.3. Class Based Queuing (CBQ)

Of all the queuing schemes, CBQ is the most developed and comprehensive. This was the queuing discipline that was chosen for the project. Another reason it was chosen is because it is simple to use from user/administrator point of view. CBQ provides both partitioning and sharing of link bandwidth using hierarchically structured classes.

A. Classes

There are many parameters that can be used to specify a class, the only required ones being the class name, the Network Interface Card (NIC) it belongs to, and the name of the parent class. Optional parameters are specify the burst characteristics, packet size, type of dropping algorithm to use, priority, and whether to borrow bandwidth from parent. Under each interface, there must be a Root class. Under the Root class, there has to be a Default class and a Control class. Each

class has its own queue and is assigned a portion of the total link bandwidth. By default, a child class can borrow any spare bandwidth from its parent. Under each interface, multiple classes (queues) can be established. Outgoing traffic is then placed into a particular class and will receive a level of service associated with that class. For example, in our case, video traffic was categorized under the "video class" and was given 65% of the total bandwidth. Each class can have various child classes. So our video class is subdivided into subclasses video1, video2 and video3. Each subgroup can be allocated percentage of the bandwidth. Each class has its own queue. A child class can borrow any spare excess bandwidth from it parent class unless otherwise specified. Packets are assigned to different classes using filtering rules. The structure of this assignment used in this project can be seen in Figure 3.2.

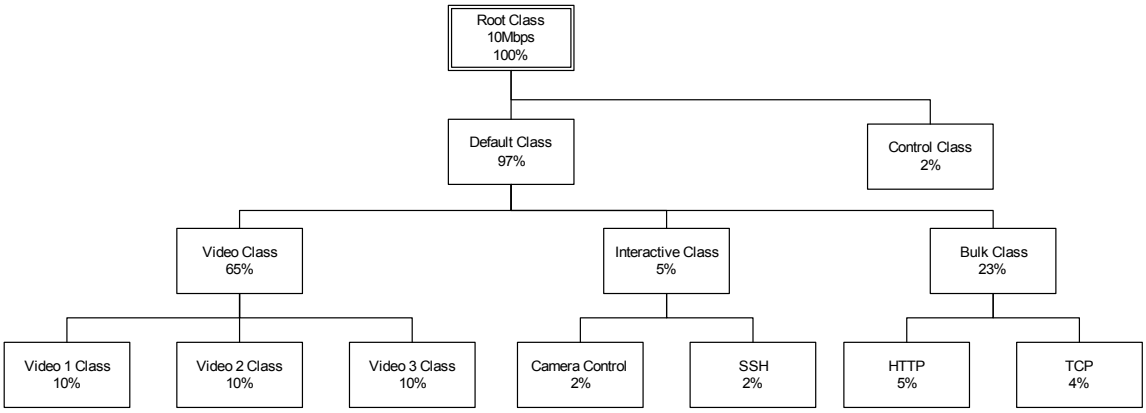


Figure 3.2 - Class Based Queuing Hierarchical structure (CBQ with classes)

Each CBQ class can be configured to run the RED or RIO algorithms on any of its classes. These algorithms are responsible for statistical packet drop prediction. RIO and RED will be discussed in Subsection 3-1.4. Only one queuing discipline can be configured per network interface.

CBQ consists of three major components:

- Classifier – this extracts flow information from the IP packet headers and uses this to try to assign the packets to one of the defined classes. If there is no corresponding class then the default class is used. The flow information can be used to extract the destination and/or source IP address, port numbers, and Type of Service (TOS) bits in the IP header. The filter explanation given in Subsection 3-1.4 explains how the classification works in detail.

- Estimator – this measures the throughput of each class and determines if the class is over, or under-limit, and marks it as such. Classes that are over limit may be allowed to borrow any spare bandwidth from their parent class (who in turn can borrow from a higher parent). For classes that have no bandwidth left to lend to subclasses, packet loss will occur. A simple tail-drop for full buffers can be used or the RED or RIO (see Subsection 3-1.4) schemes can be employed by each class.

- Packet Scheduler - takes packets from the various classes and places them on the outgoing interface. Classes are assigned a priority value and this is used to schedule packets. When

classes are assigned the same priority, a Weighted Round Robin (WRR) scheduler is used. In comparison to the Simple Round Robin scheduler, the WRR scheduler takes into account the packet sizes used by each class. A WRR scheduler keeps track of the actual number of bits transmitted by each class. If the packet size is over the allotted transmission size, then that class skips a turn next round and a class transmitting smaller packets (using less than the allotted bandwidth) gets to transmit two or more packets in one turn.

Subsection 3-1.4. ALTQ Components Common to CNDR and CBQ

A. Filters

The filters are used to classify incoming packets in to the desired class. The filter command uses as its matching parameters: ***dst_add, dst_port, src_add, src_port***, and protocol. The sample configuration file in Figure 3.7 will tell altqd to mark any packet coming from IP address 172.16.31.2 with TOS bits 0x40. There can be multiple rules like the one shown and they can be made much more complex. The filtering can happen not only by source IP address, but also by port, type of protocol, destination IP address. The filtering can also be combined to create "intersection-filtering." The CBQ classifier performs filter-matching for every packet. The classifier goes through the filters from the first entry in the configuration file, which means the most generic filters should be first in order. For example, it is possible to filter only TCP packets from IP Address 172.16.31.2 by the filters shown in Figure 3.3. For more information about filters, see Appendix B at the end of the document.

```
filter xl0 marker 0 0 172.16.31.2 0 0
filter xl0 marker 0 0  0              0 6
```

Figure 3.3 - Example of Intersection Filtering

The example in Figure 3.3 will capture only TCP traffic (designated by protocol number 6 on the second line) from IP address 172.16.31.2. This allows the system to be very flexible while still very powerful and concise. The results of the marking can be verified by running tcpdump on the NIC of the packet conditioner. Tcpdump will report the TOS bits in its output.

B. Interfaces

This is the NIC on the computer which is being used as the QoS router. Prioritization scheme queue is implemented at this NIC by a close coupling with the system driver for the network card. Only one queuing discipline can run on each network interface. The amount of network interfaces is only limited by the architecture of the computer being used.

C. RED

This is an acronym for Random Early Detection (RED). Rather than simply dropping all new packets attempting to join a full buffer, RED exercises packet dropping (or marking) stochastically according to the average queue length. The RED calculates the average queue size using a low-pass filter with an exponential weighted moving average. This calculated queue

length is then compared to two set thresholds: a minimum and a maximum threshold. When the queue size is less than the minimum threshold, no packets are dropped. When the queue length is greater than the maximum threshold, all incoming packets are dropped. If all marked packets are dropped, the average queue length is between the two thresholds and each arriving packet is dropped with a probability Pa, where Pa is a function of the average queue size. The probability that a packet from a certain connection is dropped is roughly proportional to that connection's share of the bandwidth in the queue.

D. RIO

RIO stands for RED with IN and OUT bits (RIO). RIO is two RED algorithms run at the same time on a queue. Initially, a policing function calculates if a packet from a certain source is of the "OUT" or "IN" profile. This can be determined by many parameters; for instance, maximum average throughput. Two RED algorithms are run; one for the "OUT" profile traffic and one for the "IN" profile traffic. When congestion occurs, it is the "OUT" profile of traffic that is subjected to dropping (using a single RED algorithm) first. If the congestion is severe, the "IN" profile will exceed the maximum RED threshold and be subjected to dropping. RIO was used on the queues of the CBQ classes in this project.

E. Altqstat

The altqstat program interrogates the ALTQ kernel and prints out many statistics for each defined class; this application continually updates the statistics every few seconds. Most of the altqstat statistics are read straight from kernel variables. A graphical monitoring program included with ALTQ is cbqmonitor. It prints out a graph of the information from altqstat.

Section 3-2. Network Setup

As a way to test the viability of ALTQ QoS in a mission critical corporate environment, it is necessary to set up a proper test bench for the prototype. This prototype needs to be simple enough to conduct valid experiments while still resembling a full network of a production environment on a smaller scale. From the specifications given by Mn/DOT, the test network needs to be able to support:

- Several streams of video data – these are the critical streams that need dynamic priority. These streams will get the most bandwidth by default and that bandwidth will always be reserved for them and not borrowed out to other network flows.

- A camera control channel – this stream has low bandwidth requirements but it needs low latency to ensure that the video cameras at the remote sites have a quick response to the commands sent by the camera operators.

- Other miscellaneous traffic – this is the "general" queue. It will be used to route any miscellaneous traffic as well as low priority video streams. This queue will have a high packet loss and high latency. This is the tradeoff for having high quality video streams in the video queues and low latency camera control queue.

Subsection 3-2.1. Proposed network

As can be seen in Figure 3.4, the network is composed of three subnets. The subnets are routed to each other through the use of NAT and port forwarding (PF). The configuration files for NAT and PF are provided in Appendix B.

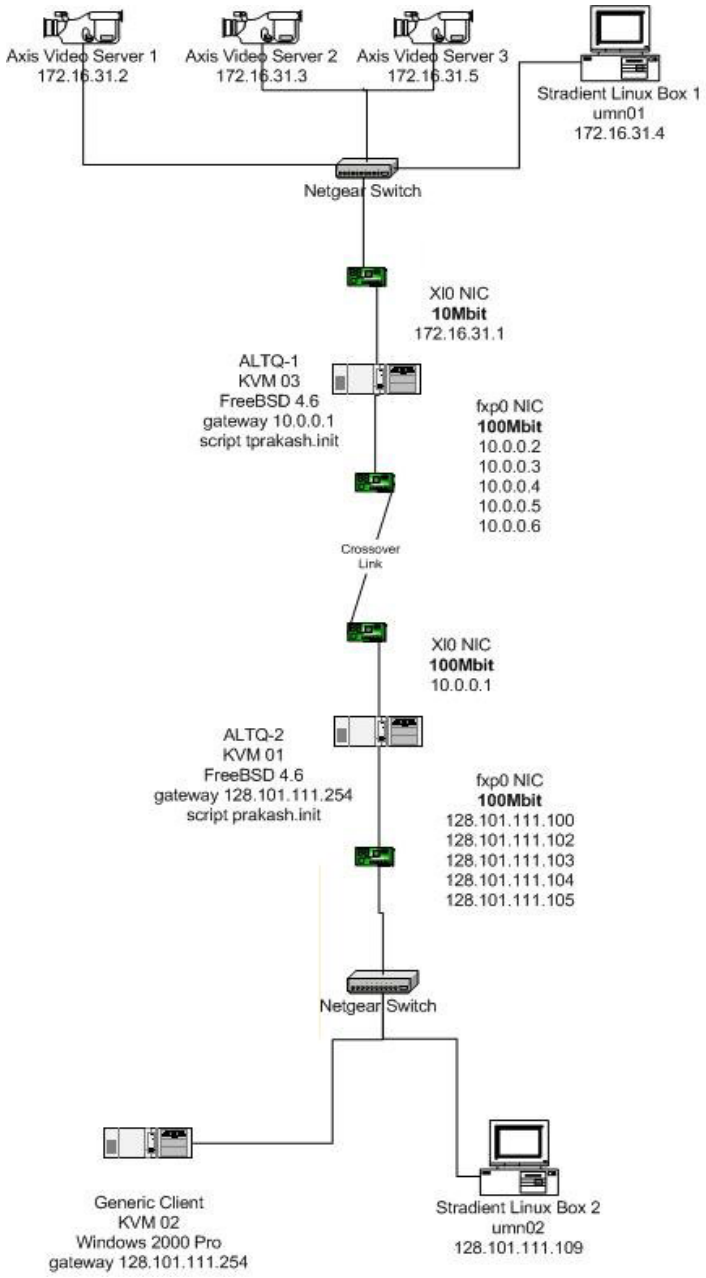


Figure 3.4 - Actual Network Layout - The crossover link between ALTQ1 and ALTQ2 is the bandwidth limited link over which ALTQ is used to control network traffic. Though the NIC on ALTQ1 box is the slower one, ALTQ is implemented over the crossover link and limits the data transfer rate to 10Mbit. The 10Mbit link ensures that the video servers are able to send no more than 10 Mbits of video data. No more than 10Mbits traverse the network in either direction.

Subsection 3-2.2. The 172 Subnet

The 172.16.31 subnet is where the cameras, the Axis Video servers, and a traffic generator computer reside. All components on the 172.16.31 subnet need to have the gateway setting set to 172.16.31.1 and need to be connected to a network switch which is in turn connected to the NIC of the ALTQ1 box. The network IP address of this NIC is 172.16.31.1. The ALTQ1 box maps the 172.16.31 subnet to the 10.0.0 subnet.

Subsection 3-2.3. The 10 Subnet

The 10.0.0 subnet is the intermediary subnet used only for routing and is mostly invisible to all other subnets. In this project, only the two ALTQ boxes can see the 10.0.0 subnet. This is the section where all the routing, mapping, and priority queuing is done. Both ALTQ boxes have two NICs and each NIC is connected to a different subnet and are connected to each other in series. See Figure 3.4 for details. The ALTQ1 box routes all the 172.16.31 subnet requests to the 10.0.0 addresses which are aliased to the second NIC (dubbed fxp0) of ALTQ1. It also implements queuing on the fxp0 NIC connected to ALTQ2 box. The packets are being colored using the Packet Conditioner on the first NIC (dubbed xl0). This sets up one directional QoS using ALTQ. This will filter and control bandwidth assignment on all network traffic coming from the Axis video servers to the next subnet but cannot control the traffic going in the opposite direction. Though the link between ALTQ1 and the video servers is 10Mbit while the rest of the links are 100Mbit, the two ALTQ boxes are used to limit the bandwidth of the link between them to 10Mbit to set the total network speed to the video servers. While it is possible to try and send up to 100Mbits of data to the video servers, the packets will be dropped by the ALTQ2 box and only 10Mbits of data will get through to the video servers. The tests using the extra traffic generators model this behavior (see Subsection 4-4.2 CASE 5).

Subsection 3-2.4. The 128 Subnet

The 128.101.111 subnet is the subnet used at the ITS Lab. This subnet is visible to the “world” and this is where the camera “operator” monitors and sends control signals to the cameras. This subnet actually consists of two parts: the second ALTQ router and the monitoring terminals. The ALTQ router, ALTQ2, is also on two subnets. One of its NICs is connected to the 10.0.0 subnet and it routes all the requests to the 128.101.111 subnet using the same technique as ALTQ1. ALTQ2 sets up QoS in the opposite direction of ALTQ1. It colors the packets using the Packet Conditioner on the NIC that is connected to the 128.101.111 subnet and queues the packets on the NIC connected to the 10.0.0 subnet. The addition of ALTQ2 router to the network is what makes a camera control channel possible; since the signals to control the cameras are coming from the opposite direction of the video streams, the first ALTQ router cannot control them. ALTQ2, together with ALTQ1, sets up a system that allows full control of the network traffic in both directions on the network.

Subsection 3-2.5. Other Prototype Features

The prototype also includes traffic generators at both end points of the network. These are in place to simulate a real network with a medium to heavy throughput. The traffic generators are on separate computers to ensure that the NIC is not a bottleneck but the network is. The idea is to give the network enough traffic to slow it down. This is done from both sides of the network to allow for traffic in the opposite direction since this shows the need for bidirectional QoS.

Though the system is not completely transparent, it can be almost invisible to the other systems. The prototype is set up to have a complete point to point mapping of IP addresses. Table 3-3 shows the direct mappings as they appear in the prototype network.

Table 3-3 - IP Address Mappings

Device	IP Address of	ALTQ1		ALTQ2		Mapped to Device
					128.101.111.100	Used to access ALTQ2
			10.0.0.2		128.101.111.101	Used to access ALTQ1
Axis Video Server 1	172.16.31.2		10.0.0.3		128.101.111.102	Used to access Axis Video Server 1
Axis Video Server 2	172.16.31.3		10.0.0.4	Gateway 10.0.0.1	128.101.111.103	Used to access Axis Video Server 2
Traffic Generator 1 (Stradient)	172.16.31.4	Gateway 172.16.31.1	10.0.0.5		128.101.111.104	Used to access Traffic Generator 1 (Stradient)
Axis Video Server 3	172.16.31.5		10.0.0.6		128.101.111.105	Used to access Axis Video Server 3

As can be seen from Table 3-3, both ALTQ routers have several IP addresses on one of the NICs. These extra addresses are aliased to that NIC and are mapped to the actual IP addresses of the devices on the network. This provides a direct connection to each device by its mapped IP address. This QoS system is not completely transparent from the rest of the network. One drawback of this is the need to have different subnets on the outer ends of the QoS system. There are ways around this but they were not investigated in this project.

Chapter 4. ALTQ Performance Analysis

Cursory pre-testing of the ALTQ QoS system has revealed that the ALTQ QoS system is very stable compared to other "software only" dynamic QoS implementations, like Envoda, and provides an improvement in video quality and network speed. Once the ALTQ QoS system was modified to support a camera control channel, the performance testing phase was broken up into several parts:

- Test environment
- Test scenarios
- Test analysis and results

The performance testing focused on the system's ability to handle large amounts of traffic, queue management, throughput, video frame rate, and packet delay of the interactive camera control channel. The tests also measured performance degradation when handling large amounts of extra network traffic. The performance analysis revealed that, in all cases tested, the performance was quite acceptable:

- Large numbers of simultaneous live video transfers (high priority) were handled with minimum loss in performance.
- Multiple extra network traffic (low priority) was filtered out properly allowing for better performance for high priority traffic.
- Switching priority and bandwidth allocation between different live video streams (high priority) does not cause any packet loss.
- Bidirectional QoS does provide slightly better results but may not be worth it due to added complexity and increased hardware cost (highly application dependent).

This chapter consists of seven sections. Section 4-1 describes the test bench environment. It gives a brief overview of the camera control channel and the extra network load imposed on the system. Section 4-2 gives a detailed description of experiment set up. Section 4-3 lists the tools and programs used to obtain the measurements. Section 4-4 gives the raw results obtained from measurements and explanation of test cases. Finally, Section 4-5 lists the conclusions of the performance analysis.

Section 4-1. Test Environment

All experiments were tested against a "control" system where no priority is being assigned to the packets and all network services are being done on a "first come, first serve" fashion. This configuration will be referred to as QOS0. See Figure 4.1.

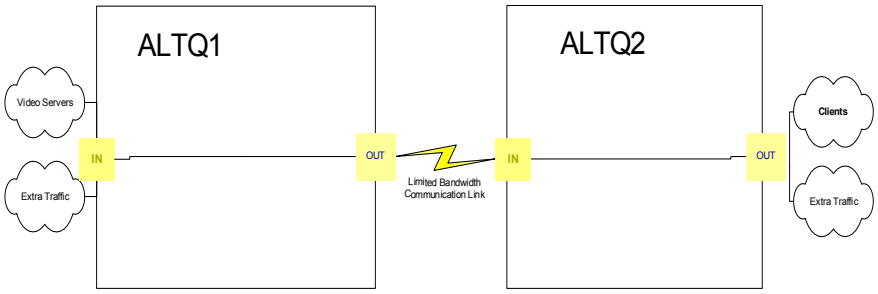


Figure 4.1 - QOS0

The top of Figure 4.2 shows the single router configuration. This setup, referred to as QOS1, provides QoS in a single direction only. The network packets are queued from the servers (left side – denoted as "Server Side") to the clients (right side – denoted as "Client Side") but the return network packets are "free flowing" and are left to fight against incoming traffic. The lower setup of Figure 4.2 illustrates a two router configuration, referred to as QOS2, which provides bidirectional QoS. This system queues the heavy traffic from the servers as well as the return packets from the clients. This allows for full control of the network traffic in both directions and better bandwidth allocation.

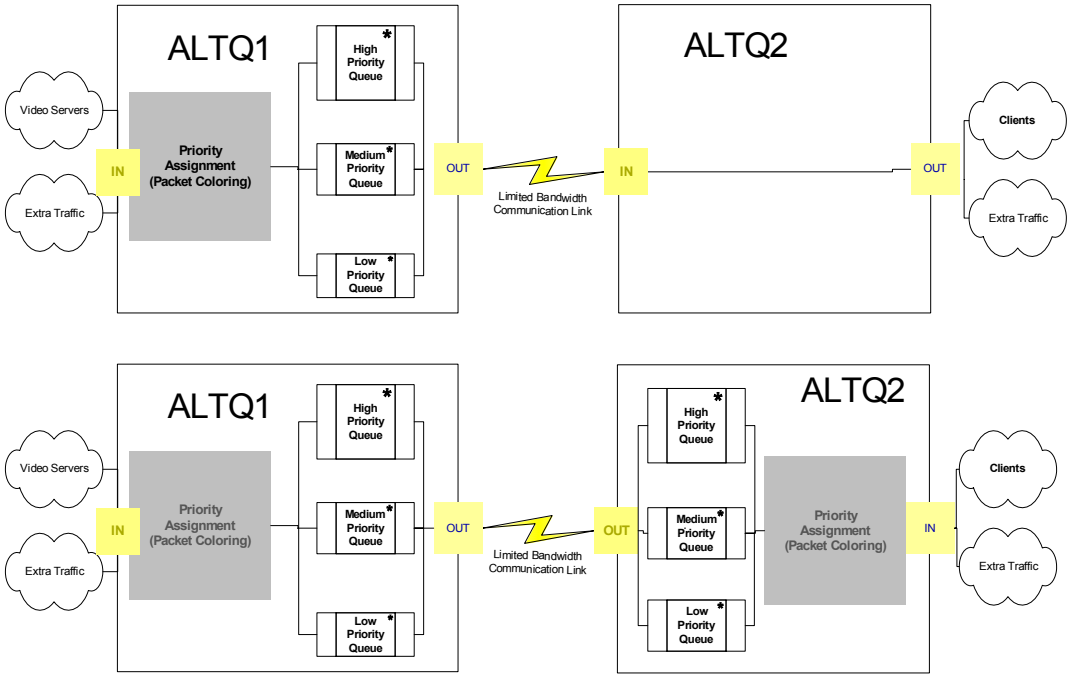


Figure 4.2 - Top – QOS1; Bottom – QOS2

The experiments for these configurations concentrated on frame-rate, amount of video data downloaded and uploaded from Axis video servers, and the amount of data downloaded and uploaded on the camera control channel. The camera control is described in Subsection 4-1.1.

Subsection 4-1.1. Camera Control Channel

The camera control network data was simulated using the ping program (using Internet Control Message Protocol (ICMP)) with a data size of 2300 bytes per transmission. The TOS (see Section 3-1.2) bits on the ping packets were modified to avoid priority being given to the ICMP packets from the ping program. By default, TOS bits of ICMP is 0x00, while normal TCP traffic uses TOS bits 0x08. By the design of the TCP stack, some newer routers give higher priority to ICMP packets over regular TCP/IP packets. By changing the TOS bits, the TCP stack treats the ICMP packets with the same priority as TCP/IP packets. This was done to remove some bias from the results. The data size of the ICMP packets was raised from the default value to 2300 bytes, which means that the average throughput rate was about 20Kbits per second. This is used to model the actual throughput of the camera controls used at Mn/DOT in order to provide "worst-case-scenario," since information on the exact throughput of the camera controls at Mn/DOT is unavailable. Most interactive network channels (ssh, telnet, etc.) use no more than a few kilobits per second so there is an assumption being made that the camera control channel is not much different. Another assumption being made is that the camera control is "pulsed," meaning that the data is not being constantly transmitted. The experiment is set up to use a "constant" data flow, which gives a higher confidence that the data throughput will be usable on a "pulsed" system if the "constant" system is unaffected by extra traffic imposed on the network.

Subsection 4-1.2. Extra Network Traffic

Extra network traffic was simulated using a program called "Netperf." This program consists of a client and server. When the client connects to the server, it sends a predetermined amount of data to the server. From Mn/DOT perspective, this extra traffic can represent unimportant video streams, non – time critical data streams, or traveler information. The traffic produced by Netperf is very bandwidth intensive and thus adds to network congestion. The direction of the majority of traffic is from client to server. There is some minor traffic in the forms of ACKs in the opposite direction.

Section 4-2. Description of Experimental Setup

There were a total of five tests run on the three systems mentioned in Section 4-1. The rationale behind all these experiments is to model the ALTQ QoS system in a realistic environment in such a way that any systematic errors will bias the results towards the worse case scenario. This provides a much more realistic environment for the use of the system. Each experiment was run for 5 minutes with measurements being taken for 1 minute at 5 second intervals. There are 3 AXIS 2400 video servers all receiving the same video stream from a VCR. The same video source was used for ease of comparison. Below is a detailed description of each test.

1. **CASE 1: CAM_CTRL.**

This experiment uses the camera control channel along with the three video streams and measures the throughput of the data on that channel. On QOS0, no traffic was given priority. On QOS1 and QOS2, however, the three video streams from the video servers were present but no priority was given to them. Camera control traffic did have allocated bandwidth, however. This experiment sets up a "control" system against which all other experiments will be compared. This "control" system shows how the system performs without any extra network traffic and without QoS, as well as the effects of heavy network load on the CAM_CTRL and the video streams.

2. **CASE 2: CAM_CTRL & Video 1 & 2.**

This experiment is the same for all three test configurations. Priority was given to Video 1 & 2 and camera control. Video 3 was present but did not have allocated bandwidth. In QOS0, all three video streams were present along with camera control. In this experiment, the effects of QoS are easily seen when comparing the quality of Video 1 & 2 to Video 3. Another observation that can be made here is the effect of only the videos (with no extra network traffic) on the camera control channel.

3. **CASE 3: CAM_CTRL & Video 1 & 2 & 3.**

This experiment is just like experiment 2 but Video 3 also gets priority for QOS1 and QOS2 configurations. QOS0 test conditions do not change. This experiment shows ALTQ's ability to handle the full load of the network and compares the results of that to the "control system" mentioned earlier. A new set of tests with new terminology are introduced to measure delay of the CAM_CTRL channel. "QOS0 no netperf," "QOS1 no netperf," and "QOS2 no netperf" refer to QOS0, QOS1, and QOS2 respectively in CASE 3.

4. **CASE 4: CAM_CTRL & Video 1 & 2 & 3 with Netperf (incoming).**

This experiment extends experiment 3. In addition to the previously described factors, it adds 2 Netperf clients on the "Server Side" connecting to 2 Netperf Servers on the "Client Side" (see Section 4-2) of the network. This adds heavy traffic flowing from the "Server Side" to the "Client Side". The packet delay of CAM_CTRL is still being measured by the technique introduced in CASE 3. QOS0_netperf1, QOS1_netperf1, and QOS2_netperf1 refer to QOS0, QOS1, and QOS2 with two instances of Netperf from "Server Side" to "Client Side." This experiment shows off the ability of ALTQ to filter out unwanted network traffic without the loss of any important network data.

5. **CASE 5: CAM_CTRL & Video 1 & 2 & 3 with Netperf (bidirectional).**

This experiment extends experiment 4 by adding two additional Netperf streams in the opposite direction of the existing Netperf streams. There are now a total of four Netperf streams; two streams from "Server Side" to "Client Side" and two streams from "Client Side" to "Server Side." This provides an overflow of network traffic in both directions and turns the network link into a serious bottleneck. The extra test to measure the delay of CAM_CTLR, introduced in CASE 3, is still being applied here. QOS0_netperf2, QOS1_netperf2, and QOS2_netperf2 refer

to QOS0, QOS1, and QOS2 respectively in CASE 5. This experiment is done to show how bidirectional ALTQ is able to handle this type of network stress but unidirectional ALTQ starts to slow down.

Section 4-3. Measurement Tools Used

There were several programs and utilities used to acquire the results for this analysis. The commercial tools used included Ethereal. Ethereal is a packet capture system; it captures all packets on the accessible network interface. It can then be used in conjunction with filters to visually show traffic destination and/or origin. The tool does have some limitations (see Measurement Accuracy and Approximations in Subsection 4-3.1 for details) and provides only graphical results. Numerical results can be extrapolated from the graphical data with some error in accuracy. This program was run on the "Client Side" where the video streams were being viewed.

Another utility that was used was "altqstat" which is part of the ALTQ distribution being used to implement QoS for this project. This tool is run on the router computer which is running ALTQ. Altqstat computes various statistics of all the queues in use. It outputs the information to standard out or to a file. However, due to the large amount of unorganized data that altqstat prints out, it is difficult to use it for any statistical purpose.

For this reason, a custom tool was developed by Harry Rostovtsev, "altqparse." This program parses the "altqstat" output and gives an output that can be imported into spreadsheet programs such as MS Excel. In addition, a macro was written for MS Excel to import the data from "altqparse" and plot various graphs, and compute statistics.

Subsection 4-3.1. Measurement Accuracy and Approximations

Measurements from Ethereal are derived from graphs and are approximate. These are denoted by "~" in front of the measurement result. When compared to results obtained from altqstat, which are assumed very accurate, the error is less than 10% in all cases, and less than 6% in most cases. Also, these results are over a period of time and are therefore averaged. The most errors in results occurred when testing configuration QOS0. In this case, altqstat cannot be used because ALTQ is not being run. All measurements are taken from Ethereal.

Measurements from altqstat are taken over ten iterations of the program and the average presented on the results table is the average of those 10 results. One caveat is the inability to start "altqstat" while ALTQ is running. Since the altqstat was started before ALTQ was, the first measurement is completely inaccurate due to empty queues. Due to this program constraint, the first result is off by several standard deviations and is ignored as a systematic error. This was done to improve the accuracy of the resulting averages.

Due to design constraints of ALTQ, any reported packet loss is not due to interruption of ALTQ. Though this does result in packet loss according to the developer of ALTQ, this loss can not be measured without seriously altering the whole code base of ALTQ and the FreeBSD kernel. Great care was taken to insure that the ALTQ was uninterrupted during the experiments to prevent packet loss to eliminate this systematic error. Any packet loss reported in the tests is from ALTQ purposefully dropping packets. All information listed about ALTQ was verified by the main developer of ALTQ, Kenjiro Cho.

Delay measurements are difficult to make on network data. The video streams did not need these tests but there were some rough measurements taken of the camera control traffic using the output of the ping program. This data has a strong dependence on the size of the packets being sent and, as mentioned earlier, represents the worst case scenario. It is difficult to predict how the measured delay will reflect a more realistic model due to the simplifications made in the model. However, due to extremely large packet size used for the tests, for no case will delay be worse than the data measured.

Section 4-4. Experimental Results

The results are presented in two subsections. Subsection 4-4.1 lists raw results obtained from the experiments. Subsection 4-4.2 provides a detailed explanation and motivation for each set of tests and experiments.

Subsection 4-4.1. Raw Results obtained from experiments

This section shows the results of the experiments performed. Table 4-1 shows the legend and some explanations for Table 4-2, Table 4-3, and Table 4-4. The results are displayed in Table 4-2, Table 4-3, and Table 4-4. Each table represents measurements taken from the point of view of the different Video stream. For example, Table 4-2 shows all the measurements for Video 1 and all the associated data with Video 1. Table 4-3 shows no measurements for Video 2. Each table does not contain the same data. The three tables (Table 4-2, Table 4-3, and Table 4-4) represent a 3D matrix, the number of tables (in this case 3 dimensions) being the depth of this matrix, represents which video measurements are being presented. Each table includes measurements taken from the different test iterations while keeping the conditions the same. This was done in order to not abstract any of the results into a single number and to isolate the results from each video into a separate category. The columns of the tables denote which QoS implementation was used. QOS0 column uses no QoS, QOS1 uses one-directional QoS, and QOS2 uses bidirectional QoS. The rows of the graphs denote which network streams were given priority. An exception to this scheme is the column QOS0. Due to lack of QoS in the QOS0 configuration, only the rows with netperf added should show any statistical difference. The rest of the rows should be in theory identical for the same table. Since the tests were run separately, each table does not necessarily show the same results for the same rows due to statistical noise in the measurements. Any differences in tables for the QOS0 column are relatively low.

Table 4-1 - Legend for tables 4-2, 4-3, and 4-4

FPS	Frame rate in frames per second. This is a measurement of the video quality. The maximum theoretical frame rate is 30 frames per second though only 29 fps was achieved in best case scenario.
Bit UP	Data uploaded back to the Axis video server. This number tends to be very low (KB) due to the data consisting of ACKs (TCP acknowledgement signals).
Bits Down	Data downloaded from the Axis video server. This number tends to be fairly large (MB) if the quality of the video is high.
Cam Up	Data uploaded to the simulated camera Pan/Tilt/Zoom (PTZ) controller. This data represents the control signals sent to control the position of the camera by the operator.
Cam Down	Data coming back from the simulated camera PTZ controller. This data represents the response from the camera PTZ controller in form of ACKs. The amount of data here should equal the amount of data uploaded due to the protocol being used. In a more realistic application, this data should be quite a bit smaller than the uploaded data.
CNDR	This is an acronym for packet Conditioner. This is the priority assignment done by packet coloring in ALTQ.
CBQ	This is an acronym for Class Based Queuing. This is the flavor of QoS that was chosen due to its maturity. The packets are queued based on the "packet color" assigned by CNDR. In the context of this document it means that QoS is being implemented.

Table 4-2 - Video 1

Point of Measurement from Video 1						
	No QoS (QOS0)		CBQ incoming & CNDR incoming (QOS1)		CBQ bidirectional & CNDR bidirectional(QOS2)	
CASE 1 CAM_CTRL	Test #	1	Test #	6	Test #	11
	FPS:	26 (24-27)	FPS:	2	FPS:	3
	Bits UP	~42Kb	Bits UP	~4.8Kb	Bits UP	~4.3Kb
	Bits Down	~2.32Mb	Bits Down	~209.67Kb	Bits Down	~200Kb
	Bits Drop	0.0Kb	Bits Drop	168.329KB	Bits Drop	~200KB
	Cam Up	~18.4Kb	Cam Up	~18.58Kb	Cam Up	18.97Kb
	Cam Down	~18.4Kb	Cam Down	18.62Kb	Cam Down	18.96Kb
CASE 2 CAM_CTRL & Video 1 & 2	Test #	2	Test #	7	Test #	12
	FPS:	27 (24-28)	FPS:	29	FPS:	29 (ave)
	Bits UP	~42Kb	Bits UP	~44.0Kb	Bits UP	43.562Kb
	Bits Down	~2.33Mb	Bits Down	2.31Mb	Bits Down	2.25Mb
	Bits Drop	0.0Kb	Bits Drop	0.0Kb	Bits Drop	0.0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.985Kb
	Cam Down	~18.4Kb	Cam Down	18.97Kb	Cam Down	18.970Kb
CASE 3 CAM_CTRL & Video 1 & 2 & 3	Test #	3	Test #	8	Test #	13
	FPS:	26 (24-28)	FPS:	26	FPS:	26 (ave)
	Bits UP	~42Kb	Bits UP	~24.0Kb	Bits UP	36.896Kb
	Bits Down	~2.32Mb	Bits Down	2.00Mb	Bits Down	1.98Mb
	Bits Drop	0.0Kb	Bits Drop	0.0Kb	Bits Drop	0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.623Kb
	Cam Down	~18.4Kb	Cam Down	18.62Kb	Cam Down	18.280Kb
CASE 4 CAM_CTRL & Video 1 & 2 & 3 with Netperf incoming direction	Test #	4	Test #	9	Test #	14
	FPS:	16(10-24)	FPS:	24(21-27)	FPS:	26 (ave)
	Bits UP	~32Kb	Bits UP	~24.0Kb	Bits UP	35.75Kb
	Bits Down	~1.58Mb	Bits Down	1.96Mb	Bits Down	1.83Mb
	Bits Drop	0.0Kb	Bits Drop	0.0Kb	Bits Drop	0.0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.67Kb
	Cam Down	~18.4Kb	Cam Down	18.96Kb	Cam Down	18.62Kb
CASE 5 CAM_CTRL & Video 1 & 2 & 3 with Netperf bidirectional	Test #	5	Test #	10	Test #	15
	FPS:	7(3-10)	FPS:	20(19-23)	FPS:	23 (ave)
	Bits UP	~6.3Kb	Bits UP	~20.0Kb	Bits UP	32.2Kb
	Bits Down	~640Kb	Bits Down	1.74Mb	Bits Down	1.77Mb
	Bits Drop	unknown	Bits Drop	0.0Kb	Bits Drop	0.0 Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.97Kb
	Cam Down	~18.4Kb	Cam Down	18.96Kb	Cam Down	18.96Kb

Table 4-3 - Video 2

Point of Measurement from Video 2						
	No QoS (QOS0)		CBQ incoming & CNDR incoming (QOS1)		CBQ bidirectional & CNDR bidirectional(QOS2)	
CASE 1 CAM_CTRL	Test #	1	Test #	6	Test #	11
	FPS:	24 (22-25)	FPS:	2	FPS:	3
	Bits UP	~42Kb	Bits UP	~4.8Kb	Bits UP	~4.3Kb
	Bits Down	~2.31Mb	Bits Down	~209.67Kb	Bits Down	~200Kb
	Bits Drop	0.0Kb	Bits Drop	168.329KB	Bits Drop	~200KB
	Cam Up	~18.4Kb	Cam Up	~18.58Kb	Cam Up	18.97Kb
	Cam Down	~18.4Kb	Cam Down	18.62Kb	Cam Down	18.96Kb
CASE 2 CAM_CTRL & Video 1 & 2	Test #	2	Test #	7	Test #	12
	FPS:	27 (24-28)	FPS:	29	FPS:	28 (ave)
	Bits UP	~42Kb	Bits UP	~44.0Kb	Bits UP	45.87Kb
	Bits Down	~2.32Mb	Bits Down	2.30Mb	Bits Down	2.29Mb
	Bits Drop	0.0Kb	Bits Drop	0.0Kb	Bits Drop	0.0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.985Kb
	Cam Down	~18.4Kb	Cam Down	18.97Kb	Cam Down	18.970Kb
CASE 3 CAM_CTRL & Video 1 & 2 & 3	Test #	3	Test #	8	Test #	13
	FPS:	26 (25-28)	FPS:	27	FPS:	25 (ave)
	Bits UP	~42Kb	Bits UP	~24.0Kb	Bits UP	37.42Kb
	Bits Down	~2.33Mb	Bits Down	2.00Mb	Bits Down	1.80Mb
	Bits Drop	0.0Kb	Bits Drop	0.0Kb	Bits Drop	0.0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.623Kb
	Cam Down	~18.4Kb	Cam Down	18.62Kb	Cam Down	18.280Kb
CASE 4 CAM_CTRL & Video 1 & 2 & 3 with Netperf incoming direction	Test #	4	Test #	9	Test #	14
	FPS:	17(10-25)	FPS:	25(23-27)	FPS:	27 (ave)
	Bits UP	~32Kb	Bits UP	~24.0Kb	Bits UP	35.24Kb
	Bits Down	~1.9Mb	Bits Down	1.99Mb	Bits Down	1.85Mb
	Bits Drop	0.0Kb	Bits Drop	0.0Kb	Bits Drop	0.0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.67Kb
	Cam Down	~18.4Kb	Cam Down	18.96Kb	Cam Down	18.62Kb
CASE 5 CAM_CTRL & Video 1 & 2 & 3 with Netperf bidirectional	Test #	5	Test #	10	Test #	15
	FPS:	6(4-10)	FPS:	18(13-24)	FPS:	27 (ave)
	Bits UP	~6.3Kb	Bits UP	~20.0Kb	Bits UP	31.18Kb
	Bits Down	~640Kb	Bits Down	1.68Mb	Bits Down	1.81Mb
	Bits Drop	unknown	Bits Drop	0.0Kb	Bits Drop	0.0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.97Kb
	Cam Down	~18.4Kb	Cam Down	18.96Kb	Cam Down	18.96Kb

Table 4-4 - Video 3

Point of Measurement from Video 3						
	No QoS (QOS0)		CBQ incoming & CNDR incoming (QOS1)		CBQ bidirectional & CNDR bidirectional(QOS2)	
CASE 1 CAM_CTRL	Test #	1	Test #	6	Test #	11
	FPS:	27 (24-29)	FPS:	2	FPS:	4
	Bits UP	~42Kb	Bits UP	~4.8Kb	Bits UP	~4.3Kb
	Bits Down	~2.32Mb	Bits Down	~209.67Kb	Bits Down	~200Kb
	Bits Drop	0.0Kb	Bits Drop	168.329KB	Bits Drop	~200KB
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.97Kb
	Cam Down	~18.4Kb	Cam Down	18.62Kb	Cam Down	18.96Kb
CASE 2 CAM_CTRL & Video 1 & 2	Test #	2	Test #	7	Test #	12
	FPS:	27 (24-28)	FPS:	7	FPS:	6(ave)
	Bits UP	~42Kb	Bits UP	~14.4Kb	Bits UP	~14.532Kb
	Bits Down	~2.32Mb	Bits Down	705.03Kb	Bits Down	~695Kb
	Bits Drop	0.0Kb	Bits Drop	~202K	Bits Drop	~103K
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.985Kb
	Cam Down	~18.4Kb	Cam Down	18.97Kb	Cam Down	18.970Kb
CASE 3 CAM_CTRL & Video 1 & 2 & 3	Test #	3	Test #	8	Test #	13
	FPS:	26 (23-28)	FPS:	26(23-28)	FPS:	26 (ave)
	Bits UP	~42Kb	Bits UP	~21.6Kb	Bits UP	36.99Kb
	Bits Down	~2.32Mb	Bits Down	2.02Mb	Bits Down	1.98Mb
	Bits Drop	0.0Kb	Bits Drop	0.0Kb	Bits Drop	0.0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.623Kb
	Cam Down	~18.4Kb	Cam Down	18.62Kb	Cam Down	18.280Kb
CASE 4 CAM_CTRL & Video 1 & 2 & 3 with Netperf incoming direction	Test #	4	Test #	9	Test #	14
	FPS:	15(9-26)	FPS:	25(23-27)	FPS:	26 (ave)
	Bits UP	~32Kb	Bits UP	~24.0Kb	Bits UP	36.0Kb
	Bits Down	~1.2Mb	Bits Down	2.02Mb	Bits Down	1.84Mb
	Bits Drop	0.0Kb	Bits Drop	0.0Kb	Bits Drop	0.0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.67Kb
	Cam Down	~18.4Kb	Cam Down	18.96Kb	Cam Down	18.62Kb
CASE 5 CAM_CTRL & Video 1 & 2 & 3 with Netperf bidirectional	Test #	5	Test #	10	Test #	15
	FPS:	6(3-11)	FPS:	22(14-26)	FPS:	23 (ave)
	Bits UP	~6.3Kb	Bits UP	~20.0Kb	Bits UP	31.94Kb
	Bits Down	~640Kb	Bits Down	1.62Mb	Bits Down	1.70Mb
	Bits Drop	unknown	Bits Drop	0.0Kb	Bits Drop	0.0Kb
	Cam Up	~18.4Kb	Cam Up	~18.62Kb	Cam Up	18.97Kb
	Cam Down	~18.4Kb	Cam Down	18.96Kb	Cam Down	18.96Kb

Subsection 4-4.2. Detailed Test Case Analysis

The analysis of the tests is as follows. Comparisons are made between each test configuration from the point of view of the three video streams. This sets up a common ground between all the tests and ensures that the comparisons are made between similar categories. Each table (Table 4-2 through Table 4-4) shows the measurements taken from the point of view of different cameras (see Section 4-2). Below is a detailed performance comparison on each setup.

1. **CASE 1: "CAM_CTRL": Tests 1, 6, & 11.**

In this test, QOS0 is a clear winner with all three video streams averaging around 26 fps, while QOS1 and QOS2 video streams are in the lower single digits. This because QOS0 setup uses no QoS scheme, the video streams are free to use all the available bandwidth. In QOS1 and QOS2, the video streams are not being assigned any priority and are being pushed through the general traffic queue which is only about 1/10 of the total bandwidth of the link. This is an intentional bottleneck. The CAM_CTRL channel throughput remains at approximately 18.4Kb/sec in both directions across all these tests. A more thorough analysis using packet return time is shown later.

2. **CASE 2: "CAM_CTRL & Video 1 & 2": Tests 2, 7, & 12.**

In this test scenario, the conditions remain the same under QOS0 but change quite radically for QOS1 and QOS2. Videos 1 and 2 now have priority assigned to them and have bandwidth allocated in the high priority queue while Video 3 remains in the "general traffic" low priority queue. The frame rates for videos 1, 2, and 3 are 29 fps, 29fps, and 7 fps respectively under QOS1 and 29 fps, 28 fps, and 6 fps respectively under QOS2. There is a dramatic improvement for videos 1 and 2 under both QOS1 and QOS2. Once again, the CAM_CTRL channel keeps its throughput of approximately 18.4 Kb.

3. **CASE 3: "CAM_CTRL & Video 1 & 2 & 3": Tests 3, 8, & 13.**

In this test, priority is assigned to all three video streams. The frame rate is approximately equal across QOS0, QOS1, and QOS2. This test actively demonstrates QoS's ability to keep up with a network that is not being fully utilized since QoS overhead can in some cases have negative impact on the network performance if the network is not at full capacity. Note that the outgoing and incoming data for the CAM_CTRL stream are both still approximately 2300 bytes/sec * 8 bits = 18.4Kbits/sec. Since the bandwidth used by the CAM_CTRL channel is relatively small compared to the video channels (~20Kb vs. ~2Mb) there will not be any considerable drop in the throughput of the system. The effects that can be noticed are the delay of CAM_CTRL packets. The delay of the packets during this test was an average of 30msec with a max delay of 100msec and a min delay of less then 10msec. The performance is only mildly degraded when using QOS1 or QOS2 scheme with no extra traffic due to the QoS overhead. Under QOS1, the average delay is 31ms with a max delay of 110ms, and under QOS2, the average delay is 28ms with a max delay of 120ms. See Table 4-5 and Figure 4.3 for an overview.

Table 4-5 - QOS0_no_netperf, QOS1_no_netperf, and QOS2_no_netperf

Empty Network	QOS0 no_netperf	QOS1 no_netperf	QOS2 no_netperf
RTS	**RTS**	**RTS**	**RTS**
Minimum(ms)	**Minimum(ms)**	**Minimum(ms)**	**Minimum(ms)**
0	0	0	0
Maximum(ms)	**Maximum(ms)**	**Maximum(ms)**	**Maximum(ms)**
11	100	110	120
Average(ms)	**Average(ms)**	**Average(ms)**	**Average(ms)**
2	30	31	28
STDEV	**STDEV**	**STDEV**	**STDEV**
0.516	24.581	30.596	30.841
Sent	**Sent**	**Sent**	**Sent**
43	43	45	54
Received	**Received**	**Received**	**Received**
43	43	45	53
Lost	**Lost**	**Lost**	**Lost**
0 (0% loss)	0 (0% loss)	0 (0% loss)	1 (1% loss)

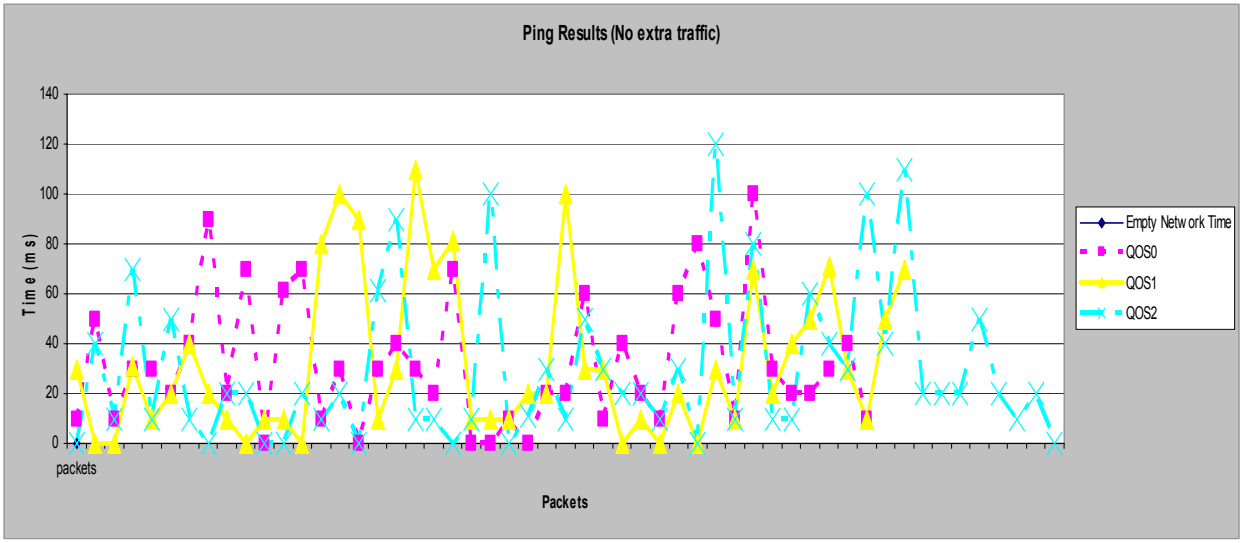


Figure 4.3 - QOS0, QOS1, and QOS2 (no netperf).

4. **CASE 4: "CAM_CTRL & Video 1 & 2 &3 with Netperf incoming direction": Tests 4, 9, & 14.**

In this test, the conditions remain as they did in CASE 3 but extra network traffic was added from "Server Side" to the "Client Side" (see Section 1) using Netperf. The frame rate under QOS0 drops down to approximately 16 fps for all three video streams due to the extra traffic on the network. QOS1 and QOS2 frame rates remain in the twenty-range. There is a discernable difference in the amount of data being transferred from the video server to the clients in QOS1 and QOS2. In order to maintain the same frame rate, the video servers in QOS1 have to transfer slightly more data (see Tables 4-2, 4-3, and 4-4 under CASE 4, tests 9 and 14, field Bits UP) than the video servers under QOS2. This is due to the extra network traffic interfering with the ACK packets from the clients under QOS1 because there is no QoS in the direction opposite of the extra network traffic. The video servers under QOS1 end up sending more data than they need to due to this. The CAM_CTRL throughput is still at ~18 Kb/sec. The same measuring technique that was used for CAM_CTRL in CASE 3 was used here. QoS is starting to show its effects on the delay time of the packets. The delay of the packets of CAM_CTRL stream under QOS0 (QOS0_netperf1) during this test was an average of 97ms with a max delay of 151ms and a min delay of 50ms. This shows a 300% increase in delay from conditions where there was no extra traffic on the network. Looking at QOS1 (QOS1_netperf1), we can see that the performance is almost the same as when there was no extra traffic on the network. Under QOS2 (QOS2_netperf1) performance, on the other hand, is mildly degraded from the previous case. This is once again due to the QoS queue overhead. Notice that the performance of QOS2_netperf1 is still much better than QOS0_netperf1; the average delay of QOS2_netperf1 is still more than twice less than QOS0_netperf1. Later tests will show that under very heavy network traffic conditions, bidirectional QoS performs extremely well. The delay times for this set of tests are summarized in Table 4-6 and Figure 4.4. Overall, in this set of tests, both QOS1 and QOS2 schemes yield much lower delay times than QOS0.

Table 4-6 - QOS0_netperf1, QOS1_netperf1, and QOS2_netperf1

QOS0_netperf1	QOS1_netperf1	QOS2_netperf1
RTS	**RTS**	**RTS**
Minimum(ms)	**Minimum(ms)**	**Minimum(ms)**
50	0	0
Maximum(ms)	**Maximum(ms)**	**Maximum(ms)**
151	120	130
Average(ms)	**Average(ms)**	**Average(ms)**
97	31	42
Sent	**Sent**	**Sent**
39	49	35
Received	**Received**	**Received**
39	49	35
Lost	**Lost**	**Lost**
0 (0% loss)	0 (0% loss)	0 (0% loss)

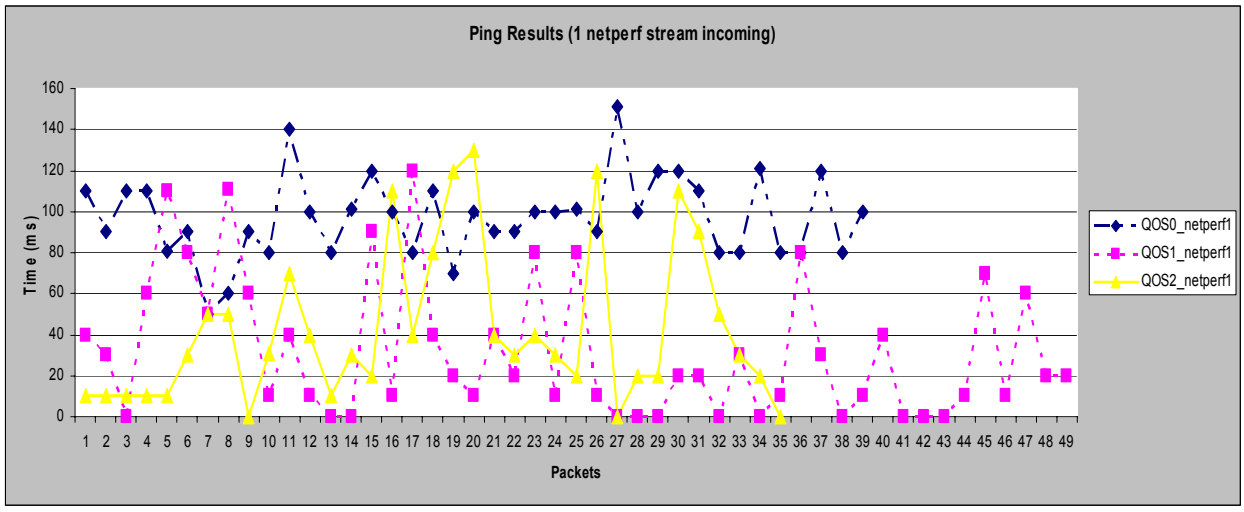


Figure 4.4 - QOS0_netperf1, QOS1_netperf1, and QOS2_netperf1

5. **CASE 5: "CAM_CTRL & Video 1 & 2 & 3 with Netperf bidirectional": Tests 5, 10, & 15.**

In this test the conditions remain as they did in CASE 3 but extra network traffic was added in both directions across the network (see Figure 4.2) using Netperf. Due to the extra traffic, the frame rates under QOS0 have dropped to approximately ~6 fps for all the video streams. Under QOS1, the frame rates also dropped but they are still above acceptable limits around 20 fps (see Tables 4-2, 4-3, and 4-4). Under QOS2, the frame rates remain around 23 fps and get as high as 27 fps (see 4-2, 4-3, and 4-4). Another notable difference is the upload rates (ACK packets) back to the video servers. Under QOS0, the upload rate is approximately 6.3 Kb/sec while under QOS1 upload rates are at around 20 Kb/sec. Under QOS2, the upload rates have also dropped from the previous CASE 4 from an average 35 Kb/sec to around 31 Kb/sec. QOS2 is able to allocate bandwidth for the ACK packets, allowing for a better bandwidth control and better filtering of the extra traffic generated by Netperf. QOS1 is unable to filter any traffic from the "Client Side" to the "Server Side." This effect is magnified when the delay times of CAM_CTRL are examined. The delay of the packets for QOS0 with bidirectional traffic (QOS0_netperf2) during this test was an average of 170ms with a max delay of 280ms and a min delay of 70ms. The average delay is almost six times (600%) longer than the delay with no extra traffic, and almost twice (200%) the average delay with only one netperf stream. Looking at QOS1 (QOS1_netperf2), we can see that the performance is significantly degraded. Compared to the previous test, the average delay went from 31ms (in QOS1_netperf1) to 112ms (in QOS1_netperf2). That's an increase by 361%. QOS2 (QOS2_netperf2), on the other hand, handles the extra traffic with only mildly degraded performance. With only one netperf stream, the average delay was 42ms. With an additional netperf stream in the opposite direction, the average delay is 69ms, reflecting an increase by only 0.6 times from the conditions where only one netperf stream is active. Notice that the performance of QOS2_netperf2 is almost three times (246%) faster than QOS0_netperf2. The delay times for this set of tests are summarized in Table 4-7 and Figure 4.5. This set of tests shows that QOS2 scheme is much better suited for handling network conditions where extra traffic flows in both directions through the network.

Table 4-7 - QOS0_netperf2, QOS1_netperf2, and QOS2_netperf2

QOS0_netperf2	QOS1_netperf2	QOS2_netperf2
RTS	**RTS**	**RTS**
Minimum(ms)	**Minimum(ms)**	**Minimum(ms)**
70	0	0
Maximum(ms)	**Maximum(ms)**	**Maximum(ms)**
280	160	170
Average(ms)	**Average(ms)**	**Average(ms)**
170	112	69
Sent	**Sent**	**Sent**
45	40	37
Received	**Received**	**Received**
44	40	37
Lost	**Lost**	**Lost**
1 (2% loss)	0 (0% loss)	0 (0% loss)

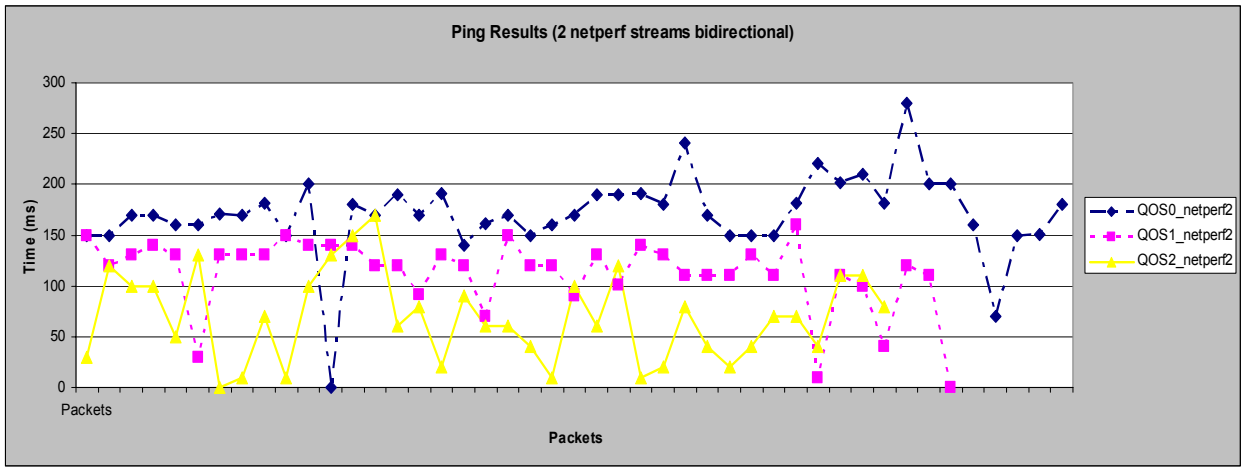


Figure 4.5 - QOS0_netperf2, QOS1_netperf2, and QOS2_netperf2

Section 4-5. Section 5 – Performance Analysis Conclusions

As can be seen from the tests that were performed on the system, ALTQ QoS implementation handles heavy network loads very well. The system is stable and has consistent and repeatable results. It was shown that with the use of ALTQ QoS, the average packet propagation time for the camera control channel, was reduced by as much as 3 times during heavy bidirectional traffic. In all cases, it was shown that the use of ALTQ QoS greatly reduced packet return times in a consistent manner. It was also shown that the video quality was greatly improved by using ALTQ. The frame rate during heavy network traffic went from 6 frames per second up to an average of 24 frames per second (with a small standard deviation of 2 or 3 frames per second).

The system has shown itself to be stable and dynamic. There were no crashes of either ALTQ routers and the dynamic changing of priority was quick and left the system fully responsive. The system can greatly improve the response time of camera controls and the quality of important video streams.

Chapter 5. Recommendations and Project Conclusions

All of the below instructions apply to FreeBSD version 4.6 and ALTQ version 3.1. As the versions of the software evolve, instructions to install them may change.

Section 5-1. Installing FreeBSD

The first step is to do a standard install of FreeBSD 4.6. Later versions can be used but the provided instructions may or may not apply. This is a fairly straight forward procedure and instructions can be found at http://www.freebsd.org. Of note are the components to be included in the installation. This depends on the user, but the most likely option to select is the "kern-developer" option. X-Windows is not required. The full link to a manual for installing FreeBSD is below:

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/install.html

This obviously needs to be done on two computers in order to setup a bidirectional QoS system.

Section 5-2. ALTQ Installation and Patching the Kernel

After the systems are fully installed, the installation of ALTQ can begin. It is not important if you chose the wrong network settings as these will be fixed later. All commands are written in *italics* for easy identification. All system responses and prompts are **bolded**.

1. Download ALTQ patch from http://www.csl.sony.co.jp/person/kjc/kjc/software.html. Untar it by command: *tar –xvzf altq*.tar.gz*
2. Before installing ALTQ, backup the original kernel source in /usr/src/sys.
3. *cd /usr/src*
4. *mkdir sys-altq* – this is the place where fresh kernel source will be kept. The altq code modifies the existing kernel.
5. *cd sys* – while the current directory is /usr/src
6. *tar xvf -. | (cd ../sys-altq; tar xf-)* – this will copy the original kernel source from /usr/src/sys into /usr/src/sys-altq.

Now apply the download patch to the fresh kernel in /usr/src/sys-altq

7. *cd /usr/src/sys-altq*
8. *patch –p0 < /altq-download-directory/sys-altq/sys-altq-<os>-<ver>.patch* – replace altq-download-directory with the directory where ALTQ was downloaded to and <os> and <ver> with the correct information.

9. *mkdir altq* – make sure that the current directory is still /usr/src/sys-altq. Now copy sys-altq files from the altq release directory to /sr/src/sys-altq/altq

10. *cp /altq-release-directory/sys-altq/altq/* altq/. –*

Making and installing the new kernel (present directory is /usr/src/sys-altq)

11. *cd i386/conf*

12. *config ALTQ*

13. *cd ../../compile/ALTQ*

14. *make depend;make clean;make;make install*

15. *shutdown –r now*

After the system reboots

16. *cd /altq-release-directory/*

17. *sh MAKEDEV.altq all*

The ALTQ major device numbers for FreeBSD is 96.

Section 5-3. Installing Userland Tools

1. *cd /altq-release-directory/*

2. *make;make install*

The default ALTQ source kernel path is /usr/src/sys-altq. By default, the altq tools are installed in /usr/local/bin and /usr/local/sbin.

To test ALTQ with a local loop:

altqd –d –f /altq-release-directory/altqd/altq.conf.samples/cbq.lo0

The command prompt should change to

altq lo0>

To exit type *quit*.

Section 5-4. Outfitting for the Field

Currently, the system consists of two computers with the following (relevant) specifications:

- Pentium III 700 MHz CPU
- 128 MB of RAM
- 20 GB Hard drive.
- 2 NICs (in each unit)

The CPU can be greatly reduced depending on the amount of traffic running through each router. If these are being used to simply control 3 to 5 streams, which would be useful for a cluster of cameras out in the field, the CPU requirements can be dropped down to Pentium II 400 MHz and possibly lower. Since RAM is relatively inexpensive and much faster, it is more efficient for the system to hold all the states of the NAT firewall in the RAM as opposed to reading them from the hard disk. The hard disk can be eliminated entirely. It is possible to make a small FreeBSD kernel which will fit onto a bootable memory flash card. All the operations can be done in RAM using ram disk. This would raise the complexity of the project but it would greatly reduce the power consumption and the physical size of the unit. With the use of Micro ATX or Mini ATX motherboards and VIA C3 CPU (see http://www.via.com.tw/en/viac3/c3.jsp) the unit can be small, efficient, and fairly inexpensive to produce. After the development costs, the assembly of the unit would cost approximately $300. The unit can be completely sealed off with only the power cable and 2 network connections going into it since the unit can use passive cooling.

For Mn/DOT networks, these units can be located at any wireless link or any link where the amount of data transmitted in both directions exceeds 75% of the link bandwidth. In cases where the amount of data transmitted is less than 75%, QoS can still be implemented with minimal detrimental effects due to queue overhead. Adding QoS to the links that do not require it, give better scalability to the network. The ALTQ routers should be placed at each end of the network link as shown in the bottom illustration of Figure 3.4. For maximum control over the network data, bidirectional QoS should be used (Chapter 4). On the remote end of the link, the hardware mentioned in the above section can be used. On the traffic operations facilities end of the link, any regular PC compatible can be used. For space and power conservation, several of these units can be combined into one unit. The hardware requirements for such a unit will depend on how many links are being combined. For five links, a multiprocessor computer should be used with five high quality NICs. These NICs should include onboard processors. One example of such a card is the Intel PRO100S PILA8470C3. All the units would be controlled through web services running on the traffic operations facilities end of the link. This can be completely separate from the ADDCO system that is currently in place at Mn/DOT. (See Figure 5.1)

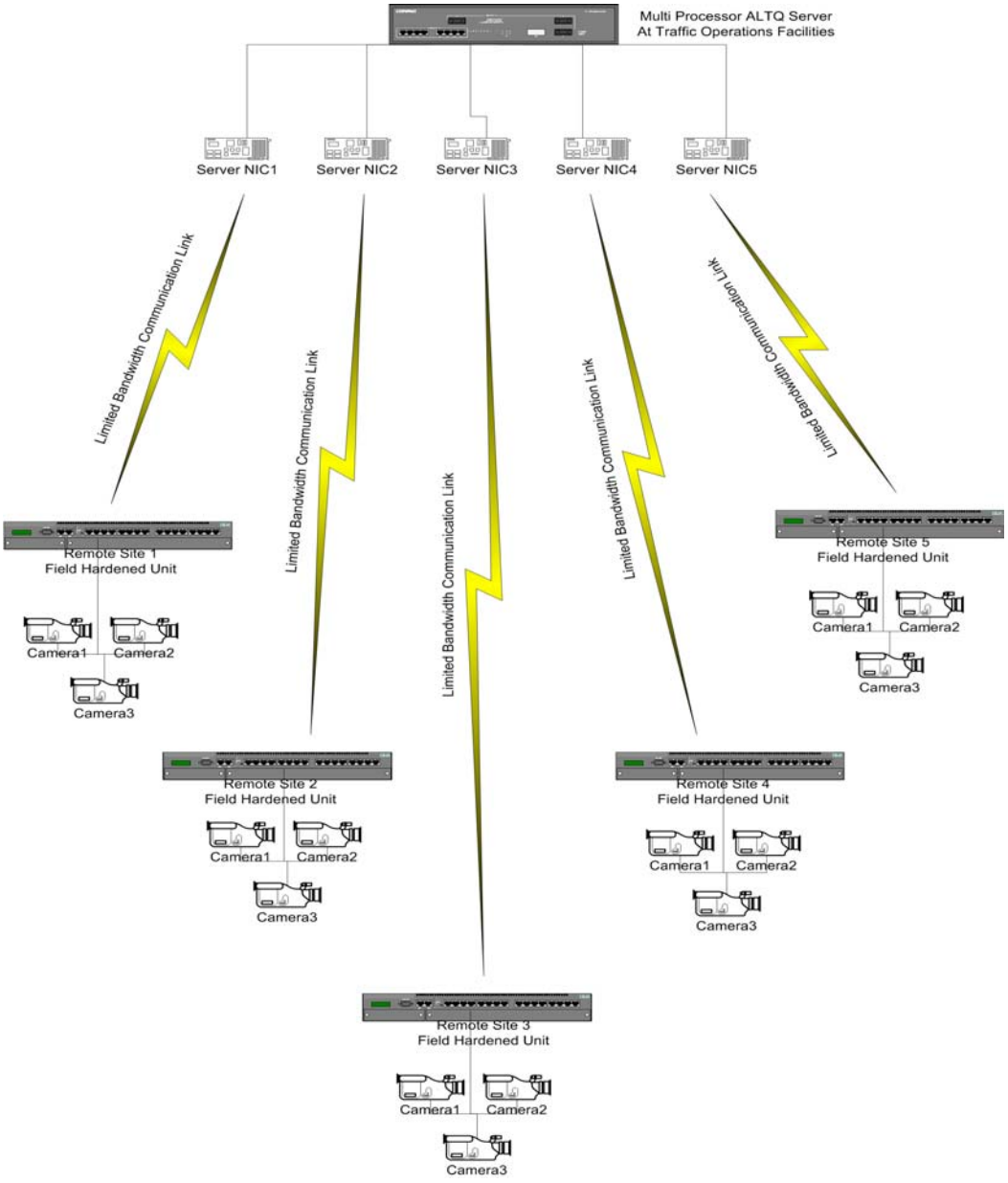


Figure 5.1 - Possible ALTQ System Layout at MNDOT

Section 5-5. Controlling Priority and Bandwidth Allocation

The changing for priority is done by using a program like Putty to SSH to the ALTQ routers and manually changing the configuration files and restarting the altqd for that particular interface. Though it sounds difficult, this is actually very simple since the only change that need to be made to the configuration file is commenting or un-commenting a few relevant lines. This only takes a couple of seconds and any user can learn to do this in a matter of minutes.

In the early days of this project, a web interface was designed to do the priority changing. This system would take the input from the user via a web interface and change the priority using the same technique as mentioned above. This interface worked fairly well but it was not designed to in a way that allows it to scale to the new bi-directional QoS. The interface system would need to be redesigned from scratch in order to support this and would be much more complex as there are many more combinations for priority assignment. The development of this web-interface did not continue as Mn/DOT did not express any interest in it. Appendix C provides the source code for the existing web interface in case others would like to build on it.

Appendix A includes a HOWTO file that describes the use of the ALTQ system set up at the ITS Lab. This README is outdated, as it was made for the original one directional ALTQ setup, but it gives a good overview of all the important system features in a short document. Most importantly, it shows how to change priorities using the SSH method. Appendix B includes all the configuration files that are currently on both ALTQ router boxes. These include the network configuration files for NAT and packet forwarding as well as ALTQ configuration files.

Chapter 6. Conclusion

As more and more networks are starting to use time dependent network services such as voice over IP or streaming video, network performance is becoming critical in providing the required quality of service required for such applications. Upgrading bandwidth is not a cheap solution nor is it a long lasting one. Traffic prioritization and bandwidth allocation offers a less expensive and a more permanent solution to these problems.

The goal of this project was the implementation of the traffic prioritization and bandwidth allocation and that is exactly what was done. The network is able to support several streams of data, a camera control channel, and low priority traffic. Users can assign priorities to different types of network data streams quickly without any packet loss. The quality of streaming video is greatly improved and the delay of packets over camera control channels is reduced. No more slow camera response or video streams of accidents at 5 frames per second. A comparison between networks with QoS vs. networks without QoS showed that in the worst case scenario, streaming video frame rate was improved by over 450% and packet delay was reduced by 600% when using QoS.

Mn/DOT would greatly benefit from the use of ALTQ QoS in its network, especially at bottlenecked remote sites. The price for implementing QoS is lower than upgrading the bandwidth of a network link that spans several miles. While upgrading may solve short term problems, the addition of more network traffic will bring the network to a crawl once again. QoS is scalable and can grow with the network to accommodate more network load and addition of more sources of network traffic.

Appendix A
ALTQ QOS System REAME

ALTQ QoS System REAME

A-1. Startup for the impatient

1. Boot ALTQ computer and login as root.

2. Type the following commands:

   BOX2# *sh /root/prakash.init*
   BOX2# *apachectl start*
   BOX2# *altqd –f /etc/altq_new.conf*
   BOX2# *altqd –f /opt/altq_in.conf*

3. Switch to a client computer and open a web-browser window. Type in the address of the video servers: 128.101.111.102, 128.101.111.103, and/or 128.101.111.105.

4. Open another browser window and type in the address of the CGI script: 128.101.111.101/cgi-bin/draft.cgi

5. Assign priority, click "submit."

A-2. How to start the ALTQ computer

Boot the ALTQ computer up and log in as root. For security purposes, the root password is not included in this README. Make sure that the Axis web-servers are up and running with a video source connected to them. These are located on the 172.16.31.0 subnet. All the clients should be on the 128.101.0.0 subnet. Type the following command to activate the proper gateway on the ALTQ system:

BOX2# *sh /root/prakash.init*

You should see a response that looks something like this:

add net default: gateway 128.101.111.254
BOX2#

A-3. How to start all the necessary processes for QoS

1. The first step is to start the web-server. The web-server used for this project is Apache 2.0.47 custom compiled to include Perl CGI support. To start the Apache, type the following command at the prompt:

   BOX2# *apachectl start*

You should see a response that looks something like this:

httpd: Could not determine the server's fully qualified domain name, using 128.101.111.101 for ServerName
BOX2#

The apachectl starts the httpd process safely and it might display a warning message about not being able to find a qualified domain name for the system and using an IP address instead. That makes sense because there is no qualified domain name for the ALTQ computer. You will be accessing it by IP address. To make sure that the apache web-server is running, type the following command at the prompt:

 # ps aux –w | grep 'httpd'

You should see a response that looks something like this:

root 305 0.0 2.0 2992 2496 ?? Ss 12:25PM 0:00.02 usr/local/apache2/bin/httpd -k start
apache 306 0.0 2.0 3008 2512 ?? I 12:25PM 0:00.00 /usr/local/apache2/bin/httpd -k start
apache 307 0.0 2.0 3008 2512 ?? I 12:25PM 0:00.00 /usr/local/apache2/bin/httpd -k start
apache 308 0.0 2.0 3008 2512 ?? I 12:25PM 0:00.00 /usr/local/apache2/bin/httpd -k start
apache 309 0.0 2.0 3008 2512 ?? I 12:25PM 0:00.01 /usr/local/apache2/bin/httpd -k start
apache 310 0.0 2.0 3008 2512 ?? I 12:25PM 0:00.00 /usr/local/apache2/bin/httpd -k start
BOX2#

This displays all the system processes that have the phrase "httpd" in them. You should see five or six lines come up on the screen. This confirms that the web server is running. The web-server will allow you to change the configuration of the coloring scheme of ALTQ. This will be further explained in section 3. [create link]

2. Starting ALTQ.

ALTQ needs to be started. There are 2 components to ALTQ.

A. Class Component.

The first component is the CBQ daemon. This does the actual queuing of the packets by their TOS bits. To start this daemon, type the following command in the prompt:

BOX2# *altqd –f /etc/altq_new.conf*

This will start the altq daemon using the /etc/altq_new.conf file. This can be any file at this point. Nothing else depends on the name of this file. However, /etc/altq_new.conf has the most current configuration in it right now. It is recommended that you use this file.

B. Marker Component.

You also need to turn on the altq daemon for changing the TOS bit of incoming packets. This is done by the following command:

BOX2# *altqd –f /opt/altq_in.conf*

This will start coloring the incoming packets with appropriate TOS bits. This must be the location of the file because it is hard coded in the Perl CGI script and the system will not work properly if a different location is used without updating the Perl CGI script first. The contents of this file depend on the last time that the system was run. Whatever configuration options were put into it last will be there this time. This means that the file can contain no filters. This should cause no problems as long as there is a file there.

C. Make sure everything is running.

Make sure the altq daemons are running by issuing the following command at the prompt:

BOX2# *ps aux –w | grep 'altq'*

You should see two daemons running using the configuration files that were specified earlier.

root 256 0.0 0.7 1148 824 ?? Is 11:58AM 0:00.00 altqd –f etc/altq_new.conf
root 258 0.0 0.7 1148 824 ?? Is 11:58AM 0:00.00 altqd -f /opt/altq_in.conf
BOX2#

A-4. How to run ALTQ system.

Now that all the necessary daemons are running, you can connect to the ALTQ system from any client using a web-browser that supports Java or ActiveX controls. Almost any web-browser works including Internet Explorer and Mozilla. Open the web-browser and type in the address of the video source in the address line.

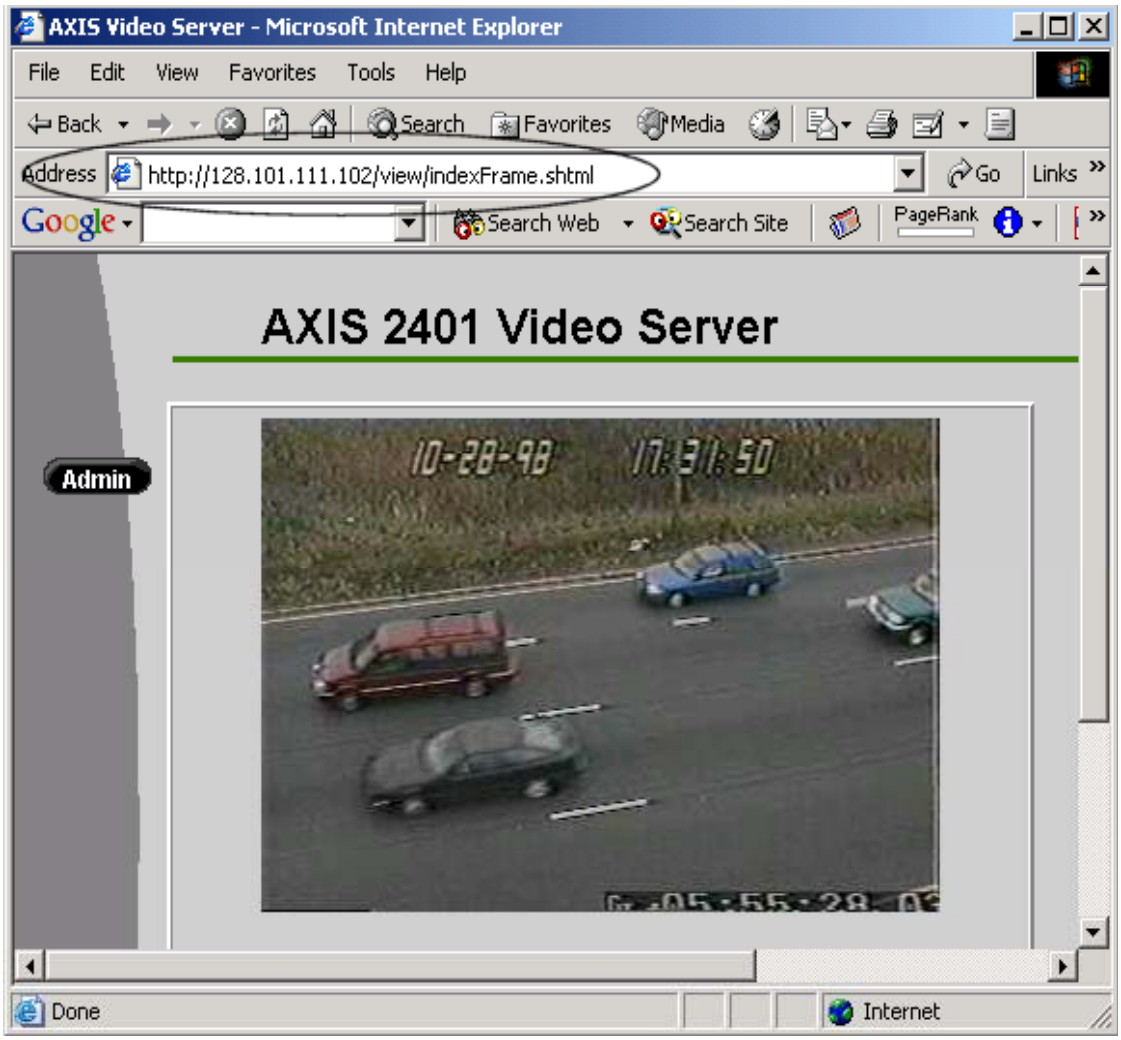


Figure A.1 – Axis Video Server View Window

Valid addresses are **128.101.111.102, 128.101.111.103,** and **128.101.111.105**. These are the three preconfigured Axis video servers. More video servers can be added as necessary.

To change the priority assignment in the ALTQ system is very easy. It can be done either by manually changing the /opt/altq_in.conf file contents and manually restarting the altq daemon, or it can be done using the web-interface system. To use the manual system, read the README file located at http://www.csl.sony.co.jp/person/kjc/kjc/software/TIPS.txt

The following are the instructions for using the web-interface system. Open any web-browser and type the address of the ALTQ computer in the address line followed by the path from the www root directory to the CGI script (for our ALTQ computer, this is http://128.101.111.101/cgi-bin/draft.cgi).

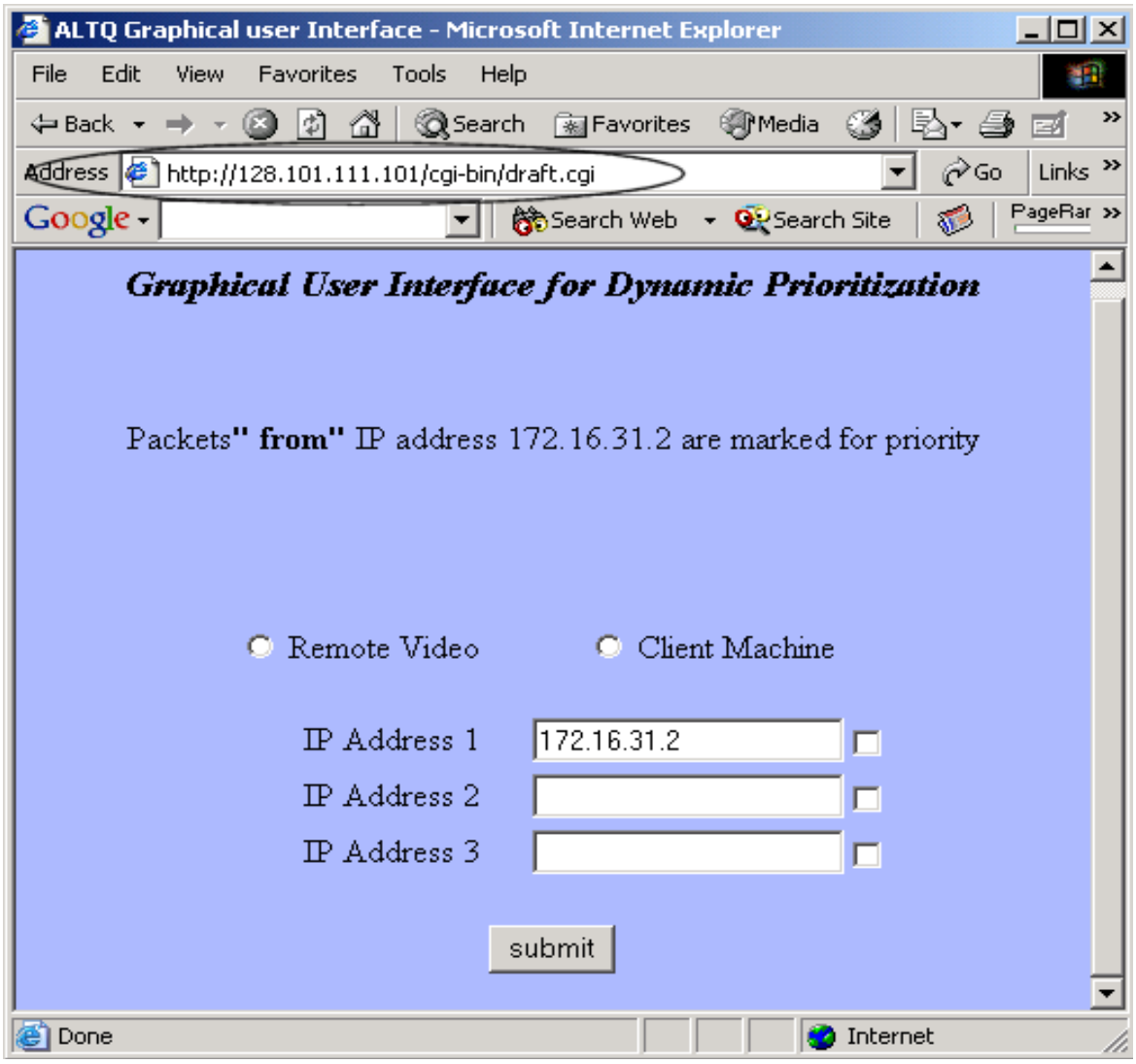


Figure A.2 - Web Interface Window

There are two configuration modes for the system. The first mode (Remote Video) allows remote video server to have the priority no matter who is watching them. The rest of the servers will get less bandwidth. The second mode (Client Machine) allows a certain client to have the priority no matter what video source the client is watching. Every other client will get less bandwidth.

A. Remote Video Mode.

To use the system in "Remote Video Mode," make sure that the "Remote Video" radio button is checked. Then proceed to type in the IP addresses of the video servers (Axis video servers, in this case) into the input boxes marked "IP

Address." Valid choices for our system are **172.16.31.2, 172.16.31.3,** and **172.16.31.5**. These are the original addresses of the Axis video servers before being forwarded to the 128.101.0.0 subnet.

After inputting the addresses of the desired video servers, make sure to click the check box next to each input box. Finally, click the "submit" button at the bottom of the page. After clicking "submit," the page will refresh and display which packets are being marked for priority. (See Figure A.2).

B. Client Machine Mode.

To use the system in "Client Machine Mode," make sure that the "Client Machine" radio button is checked. Then, proceed to type in the IP addresses of the client that you want to have priority.

Note: This is still experimental. If more than one client is specified, the system does not work at optimal capacity. More work needs to be done in this area.

After inputting the addresses of the desired clients to be marked for priority, make sure to click the check box next to each input box. Finally, click the "submit" button at the bottom of the page. After clicking "submit," the page will refresh and display which clients are being marked for priority. (See Figure A.2).

A-5. How to monitor and log ALTQ system.

Monitoring the throughput of the ALTQ system can be done in several ways. One of the easiest ways to see what is happening to the packets is to use the tcpdump command to view the TOS bits and altqstat command to see the state of classes. Below is an example done using ssh to the ALTQ machine:

BOX2# *tcpdump –i fxp0*

13:06:41.404374 128.101.111.102.http > 128.101.111.171.2905: . 236036:237496(1460) ack 1 win 32736 (DF) [tos 0x20]
13:06:41.405789 128.101.111.102.http > 128.101.111.171.2905: . 237496:238956(1460) ack 1 win 32736 (DF) [tos 0x20]
13:06:41.406253 128.101.111.171.2905 > 128.101.111.102.http: . ack 238956 win 65535 (DF)
13:06:41.407407 128.101.111.102.http > 128.101.111.171.2905: P 238956:239711(755) ack 1 win 32736 (DF) [tos 0x20]
494 packets received by filter
0 packets dropped by kernel
BOX2#

Tcpdump allows you to see that the packets are being marked properly. Notice the tos 0x20]. This is what the chosen TOS bit for that IP address is. If packets were not being marked, they are labeled with [tos 0x08] by default or none at all.

To see the state of the CBQ classes, issue the altqstat –i fxp0 command at the prompt. Below is an example:

```
Root Class for Interface fxp0: root
priority: 0 depth: 3 offtime: 1 [us] wrr_allot:
1500 bytes
nsPerByte: 800  (10.00Mbps),    Measured: 1.97M
[bps]
pkts: 2301,     bytes: 3262462
overs: 0,       overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
QCount: 0,      (qmax: 30)
AvgIdle: 37 [us],       (maxidle: 37 minidle: -
1200 [us])

Default Class for Interface fxp0: def_class
priority: 1 depth: 2 offtime: 74 [us]
wrr_allot: 5751 bytes
nsPerByte: 824  (9.71Mbps),     Measured: 0
[bps]
pkts: 0, bytes: 0
overs: 6,       overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
QCount: 0,      (qmax: 30)
AvgIdle: 35 [us],       (maxidle: 35 minidle: -
1236 [us])


Ctl Class for Interface fxp0: ctl_class
priority: 1 depth: 0 offtime: 119375 [us]
wrr_allot: 118 bytes
nsPerByte: 40000        (200.00Kbps),
Measured: 0 [bps]
pkts: 5,        bytes: 530
overs: 0,       overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
QCount: 0,      (qmax: 30)
AvgIdle: 7127 [us],     (maxidle: 7812 minidle:
-60000 [us])

Class 0 on Interface fxp0: bulk
priority: 1 depth: 1 offtime: 8151 [us]
wrr_allot: 1362 bytes
nsPerByte: 3478 (2.30Mbps),     Measured: 0
[bps]
pkts: 0,        bytes: 0
overs: 61,      overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
QCount: 0,      (qmax: 30)
AvgIdle: 862 [us],      (maxidle: 2649 minidle:
-5217 [us])


Class 1 on Interface fxp0: intr
priority: 1 depth: 1 offtime: 46312 [us]
wrr_allot: 296 bytes
nsPerByte: 16000        (500.00Kbps),
Measured: 0 [bps]
pkts: 0,        bytes: 0
overs: 0,       overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
QCount: 0,      (qmax: 30)
AvgIdle: 1425 [us],     (maxidle: 3062 minidle:
-24000 [us])


Class 2 on Interface fxp0: vide
priority: 1 depth: 1 offtime: 1311 [us]
wrr_allot: 3853 bytes
nsPerByte: 1230 (6.50Mbps),     Measured: 0
[bps]
pkts: 0,        bytes: 0
overs: 0,       overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
```

```
QCount: 0,      (qmax: 30)
AvgIdle: 422 [us],      (maxidle: 422 minidle:
-1845 [us])

Class 3 on Interface fxp0: tcp
priority: 1 depth: 0 offtime: 58515 [us]
wrr_allot: 236 bytes
nsPerByte: 20000        (400.00Kbps),
Measured: 0 [bps]
pkts: 0,        bytes: 0
overs: 0,       overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
RED q_avg:0.00 xmit:0 (forced:0 early:0
marked:0)
QCount: 0,      (qmax: 60)
AvgIdle: 3828 [us],     (maxidle: 3828 minidle:
-30000 [us])

Class 4 on Interface fxp0: http
priority: 1 depth: 0 offtime: 46312 [us]
wrr_allot: 296 bytes
nsPerByte: 16000    (500.00Kbps),    Measured:
31.40K [bps]
pkts: 1361,     bytes: 2032206
overs: 1317,    overactions: 55
borrows: 1261,  delays: 55
drops: 3,       drop_bytes: 4542
RED q_avg:6.12 xmit:1361 (forced:0 early:3
marked:0)
QCount: 0,      (qmax: 60)
AvgIdle: -7019 [us],    (maxidle: 3062 minidle:
-24000 [us])

Class 5 on Interface fxp0: vid1
priority: 7 depth: 0 offtime: 21937 [us]
wrr_allot: 1506 bytes
nsPerByte: 8000 (1.00Mbps),     Measured: 1.93M
[bps]
pkts: 860,      bytes: 1210848
overs: 855,     overactions: 0
borrows: 854,   delays: 0
drops: 0,       drop_bytes: 0
RED q_avg:0.00 xmit:860 (forced:0 early:0
marked:0)
QCount: 0,      (qmax: 60)
AvgIdle: -5565 [us],    (maxidle: 1437 minidle:
-12000 [us])

Class 6 on Interface fxp0: vid2
priority: 7 depth: 0 offtime: 21937 [us]
wrr_allot: 1506 bytes
nsPerByte: 8000 (1.00Mbps),     Measured: 0
[bps]
pkts: 0,        bytes: 0
overs: 0,       overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
RED q_avg:0.00 xmit:0 (forced:0 early:0
marked:0)
QCount: 0,      (qmax: 60)
AvgIdle: 1437 [us],     (maxidle: 1437 minidle:
-12000 [us])

Class 7 on Interface fxp0: vid3
priority: 7 depth: 0 offtime: 21937 [us]
wrr_allot: 1506 bytes
nsPerByte: 8000 (1.00Mbps),     Measured: 0
[bps]
pkts: 0,        bytes: 0
overs: 0,       overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
RED q_avg:0.00 xmit:0 (forced:0 early:0
marked:0)
QCount: 0,      (qmax: 60)
AvgIdle: 1437 [us],     (maxidle: 1437 minidle:
-12000 [us])

Class 8 on Interface fxp0: ssh
priority: 1 depth: 0 offtime: 119375 [us]
wrr_allot: 118 bytes
nsPerByte: 40000        (200.00Kbps),
Measured: 7.78K [bps]
pkts: 75,       bytes: 18878
overs: 0,       overactions: 0
borrows: 0,     delays: 0
drops: 0,       drop_bytes: 0
RED q_avg:0.00 xmit:75 (forced:0 early:0
marked:0)
QCount: 0,      (qmax: 60)
AvgIdle: 2363 [us],     (maxidle: 7812 minidle:
-60000 [us])
```

As you can see, the important information has been **bolded** to make clearer. The above output shows that Class 4 is used as the low priority traffic and Class 5, Class 6, and Class 7 are being used for the high priority video. Class 5 is the only one being used at the moment. You can also see how many packets each class drops, how much bandwidth it borrows from the parent class. For more information see http://www.csl.sony.co.jp/person/kjc/kjc/software/TIPS.txt or the manual pages.

For measurements that included extra "noise" traffic, it is difficult to differentiate the low priority video from other traffic. For these scenarios, it is easier to measure at the client using software like ethereal. This software is similar to tcpdump but allows many more options and is easier to use. It also provided a graphical interface and a good filtering utility which allows the display of only certain traffic. For more information on how to use ethereal, visit their website at www.ethereal.com .

Appendix B
ALTQ Configuration Files

B-1. Configuration Files for ALTQ1

#/etc/altq_cam_ctrl.conf

```
interface fxp0 bandwidth 10M cbq
class cbq fxp0 root NULL pbandwidth 100
#
# meta classes
#
class cbq fxp0 ctl_class root pbandwidth 2 control
class cbq fxp0 def_class root borrow pbandwidth 97 default
#
class cbq fxp0 bulk def_class  pbandwidth 23
class cbq fxp0 intr def_class  pbandwidth 5
class cbq fxp0 vide def_class borrow pbandwidth 65
#
# leaf classes
# bulk classes
#
class cbq fxp0 tcp bulk borrow pbandwidth 4 red
  filter fxp0 tcp 0 0 0 0 6
class cbq fxp0 http bulk borrow pbandwidth 5 red
  filter fxp0 http 0 0 0 80 6
  filter fxp0 http 0 80 0 0 6
#
# vide classes
#
class cbq fxp0 vid1 vide borrow priority 7 pbandwidth 10 red
  filter fxp0 vid1 0 0 0 0 0 tos 0x20 tosmask 0xff
class cbq fxp0 vid2 vide borrow priority 7 pbandwidth 10 red
  filter fxp0 vid2 0 0 0 0 0 tos 0x40 tosmask 0xff
class cbq fxp0 vid3 vide borrow priority 7 pbandwidth 10 red
  filter fxp0 vid3 0 0 0 0 0 tos 0x60 tosmask 0xff
#
# interactive classes
#
class cbq fxp0 ssh intr borrow pbandwidth 2 red
  filter fxp0 ssh 0 0 0 22 6
  filter fxp0 ssh 0 22 0 0 6
class cbq fxp0 cam_ctl intr borrow pbandwidth 2 red
  filter fxp0 cam_ctl 128.101.111.171 0 0 0 1
  filter fxp0 cam_ctl 0 0 174.16.31.4 0 1
```

#/opt/altq_in.conf – comment out conditioner-filter pair to not give priority to that IP address.

```
interface xl0
 conditioner xl0 marker1 <mark 0x20>
   filter xl0 marker1 0 0 172.16.31.2 0 0
 conditioner xl0 marker2 <mark 0x40>
   filter xl0 marker2 0 0 172.16.31.3 0 0
conditioner xl0 marker3 <mark 0x60>
   filter xl0 marker3 0 0 172.16.31.5 0 0
```

#/etc/natd.conf

```
redirect_address 172.16.31.2 10.0.0.3
redirect_address 172.16.31.3 10.0.0.4
redirect_address 172.16.31.4 10.0.0.5
redirect_address 172.16.31.5 10.0.0.6
redirect_address 172.16.31.6 10.0.0.7
```

#/etc/rc.conf

```
# -- sysinstall generated deltas -- # Thu Jul 11 14:34:07 2002
# Created: Thu Jul 11 14:34:07 2002
# Enable network daemons for user convenience.
# Please make all changes to this file, not to /etc/defaults/rc.conf.
# This file now contains just the overrides from /etc/defaults/rc.conf.
kern_securelevel_enable="NO"
moused_enable="YES"
nfs_reserved_port_only="YES"
saver="fire"
sendmail_enable="NO"
sshd_enable="YES"
usbd_enable="YES"
#Config options to make this box a gateway
gateway_enable="YES"
#default_router="128.101.111.254"
default_router="10.0.0.1"
firewall_enable="YES"
firewall_type="open"
natd_enable="YES"
natd_interface="fxp0"
natd_flags="-f /etc/natd.conf"
#natd_flags=""
ipnat_enable="YES"
ipnat_rules="/etc/ipnat.rules"
#named_enable="YES"
#hostname="Akash.me.umn.edu"
network_interfaces="fxp0 xl0 lo0"
ifconfig_lo0="inet 127.0.0.1"
ifconfig_xl0="inet 172.16.31.1 netmask 255.255.255.0"
ifconfig_fxp0_alias0="inet 10.0.0.2 netmask 255.255.255.0"
ifconfig_fxp0_alias1="inet 10.0.0.3 netmask 255.255.255.0"
ifconfig_fxp0_alias2="inet 10.0.0.4 netmask 255.255.255.0"
ifconfig_fxp0_alias3="inet 10.0.0.5 netmask 255.255.255.0"
ifconfig_fxp0_alias4="inet 10.0.0.6 netmask 255.255.255.0"
ifconfig_fxp0_alias5="inet 10.0.0.7 netmask 255.255.255.0"
dumpdev="/dev/ad0s1b"
```

#/etc/ipnat.conf

```
bimap fxp0 172.16.31.2/32 -> 128.101.111.102/32
bimap fxp0 172.16.31.3/32 -> 128.101.111.103/32
bimap fxp0 172.16.31.4/32 -> 128.101.111.104/32
bimap fxp0 172.16.31.5/32 -> 128.101.111.105/32
bimap fxp0 172.16.31.6/32 -> 128.101.111.106/32
bimap fxp0 172.16.31.2/32 -> 10.0.0.3/32
bimap fxp0 172.16.31.3/32 -> 10.0.0.4/32
bimap fxp0 172.16.31.4/32 -> 10.0.0.5/32
bimap fxp0 172.16.31.5/32 -> 10.0.0.6/32
bimap fxp0 172.16.31.6/32 -> 10.0.0.7/32
```

#add this line to /etc/inetd.conf – it allows web services to restart altqd – the #"/opt/altq_in.conf" should be replaced by the file that you are using to control

```
priority
hupaltq stream  tcp     nowait/2/20     root    /usr/bin/killall killall -HUP altqd -f
/opt/altq_in.conf

# run this script as root on startup
route add default 10.0.0.1
#ifconfig fxp0 alias 10.0.0.2 netmask 0xffffffff
ifconfig fxp0 alias 10.0.0.3 netmask 0xffffffff
ifconfig fxp0 alias 10.0.0.4 netmask 0xffffffff
ifconfig fxp0 alias 10.0.0.5 netmask 0xffffffff
ifconfig fxp0 alias 10.0.0.6 netmask 0xffffffff
ifconfig fxp0 alias 10.0.0.7 netmask 0xffffffff
```

B-2. Configuration Files for ALTQ2

#/etc/altq_cam_ctrl.conf

```
interface xl0 bandwidth 10M cbq
class cbq xl0 root NULL pbandwidth 100
#
#  meta classes
#
class cbq xl0 ctl_class root pbandwidth 2 control
class cbq xl0 def_class root borrow pbandwidth 97 default
#
class cbq xl0 bulk def_class pbandwidth 40
class cbq xl0 intr def_class pbandwidth 27
class cbq xl0 vide def_class borrow pbandwidth 30
#
#  leaf classes
#
#
#  bulk classes
#
class cbq xl0 tcp bulk borrow pbandwidth 10 red
  filter xl0 tcp 0 0 0 0 6
class cbq xl0 http bulk borrow pbandwidth 10 red
  filter xl0 http 0 0 0 80 6
  filter xl0 http 0 80 0 0 6
#class cbq xl0 udp bulk borrow pbandwidth 10 red
#  filter xl0 udp 0 0 0 0 17
#
#  vide classes
#
class cbq xl0 vid1 vide borrow priority 7 pbandwidth 5 red
  filter xl0 vid1 0 0 0 0 0 tos 0x30 tosmask 0xff
class cbq xl0 vid2 vide borrow priority 7 pbandwidth 5 red
  filter xl0 vid2 0 0 0 0 0 tos 0x50 tosmask 0xff
class cbq xl0 vid3 vide borrow priority 7 pbandwidth 5 red
  filter xl0 vid3 0 0 0 0 0 tos 0x70 tosmask 0xff
#
# interactive classes
#
class cbq xl0 ssh intr borrow pbandwidth 2 red
  filter xl0 ssh 0 0 0 22 6
  filter xl0 ssh 0 22 0 0 6
class cbq xl0 cam_ctl intr borrow pbandwidth 2 red
  filter xl0 cam_ctl 10.0.0.5 0 0 0 1
  filter xl0 cam_ctl 0 0 128.101.111.171 0 1
```

#/opt/altq_in.conf – comment out conditioner-filter pair to not give priority to that IP address.

```
interface fxp0
  conditioner fxp0 marker1 <mark 0x30>
   filter fxp0 marker1 10.0.0.3 0 0 0 0
  conditioner fxp0 marker2 <mark 0x50>
   filter fxp0 marker2 10.0.0.4 0 0 0 0
 conditioner fxp0 marker3 <mark 0x70>
  filter fxp0 marker3 10.0.0.6 0 0 0 0
```

#/etc/natd.conf

```
redirect_address 172.16.31.2 128.101.111.102
redirect_address 172.16.31.3 128.101.111.103
redirect_address 172.16.31.4 128.101.111.104
redirect_address 172.16.31.5 128.101.111.105
redirect_address 172.16.31.6 128.101.111.106
redirect_address 10.0.0.2 128.101.111.101
redirect_address 10.0.0.3 128.101.111.102
redirect_address 10.0.0.4 128.101.111.103
redirect_address 10.0.0.5 128.101.111.104
redirect_address 10.0.0.6 128.101.111.105
redirect_address 10.0.0.7 128.101.111.106
```

#/etc/rc.conf

```
# -- sysinstall generated deltas -- # Thu Jul 11 14:34:07 2002
```

```
# Created: Thu Jul 11 14:34:07 2002
# Enable network daemons for user convenience.
# Please make all changes to this file, not to /etc/defaults/rc.conf.
# This file now contains just the overrides from /etc/defaults/rc.conf.
kern_securelevel_enable="NO"
moused_enable="NO"
nfs_reserved_port_only="YES"
saver="fire"
sendmail_enable="NO"
sshd_enable="YES"
usbd_enable="YES"
#Config options to make this box a gateway
gateway_enable="YES"
default_router="128.101.111.254"
firewall_enable="YES"
firewall_type="open"
natd_enable="YES"
natd_interface="fxp0"
natd_flags="-f /etc/natd.conf"
ipnat_enable="YES"
ipnat_rules="/etc/ipnat.rules"
#named_enable="YES"
#hostname="Akash.me.umn.edu"
network_interfaces="fxp0 xl0 lo0"
ifconfig_lo0="inet 127.0.0.1"
#ifconfig_xl0="inet 128.101.111.101 netmask 255.255.255.0"
ifconfig_fxp0="inet 128.101.111.101 netmask 255.255.255.0"
ifconfig_xl0="inet 10.0.0.1 netmask 255.255.255.0"
ifconfig_fxp0_alias0="inet 128.101.111.100 netmask 255.255.255.0"
ifconfig_fxp0_alias1="inet 128.101.111.102 netmask 255.255.255.0"
ifconfig_fxp0_alias2="inet 128.101.111.103 netmask 255.255.255.0"
ifconfig_fxp0_alias3="inet 128.101.111.104 netmask 255.255.255.0"
ifconfig_fxp0_alias4="inet 128.101.111.105 netmask 255.255.255.0"
ifconfig_fxp0_alias5="inet 128.101.111.106 netmask 255.255.255.0"
dumpdev="/dev/ad0s1b"
```

#/etc/ipnat.conf

```
bimap fxp0 172.16.31.2/32 -> 128.101.111.102/32
bimap fxp0 172.16.31.3/32 -> 128.101.111.103/32
bimap fxp0 172.16.31.4/32 -> 128.101.111.104/32
bimap fxp0 172.16.31.5/32 -> 128.101.111.105/32
bimap fxp0 172.16.31.6/32 -> 128.101.111.106/32
bimap fxp0 10.0.0.2/32 -> 128.101.111.101/32
bimap fxp0 10.0.0.3/32 -> 128.101.111.102/32
bimap fxp0 10.0.0.4/32 -> 128.101.111.103/32
bimap fxp0 10.0.0.5/32 -> 128.101.111.104/32
bimap fxp0 10.0.0.6/32 -> 128.101.111.105/32
bimap fxp0 10.0.0.7/32 -> 128.101.111.106/32
```

#add this line to /etc/inetd.conf – it allows web services to restart altqd – the #"/opt/altq_in.conf" should be replaced by the file that you are using to control priority

```
hupaltq stream  tcp     nowait/2/20     root    /usr/bin/killall killall -HUP
altqd -f /opt/altq_in.conf
```

 # run this script as root on startup

```
route add default 128.101.111.254
#ifconfig xl0  alias 10.0.0.1 netmask 0xffffff00
ifconfig fxp0 alias 128.101.111.100 netmask 0xffffffff
ifconfig fxp0 alias 128.101.111.102 netmask 0xffffffff
ifconfig fxp0 alias 128.101.111.103 netmask 0xffffffff
ifconfig fxp0 alias 128.101.111.104 netmask 0xffffffff
ifconfig fxp0 alias 128.101.111.105 netmask 0xffffffff
ifconfig fxp0 alias 128.101.111.106 netmask 0xffffffff
```

Appendix C

Perl source code for a web interface to control priority allocation for ALTQ.

By Akash Malhotra and Harry Rostovtsev

```
#!/usr/bin/perl -w

use Socket;
use CGI ':all';

print header('text/html');
$radiovalue=param('video');
$IP_address_value=param('vtext1');
$IP_address_value2=param('vtext2');
$IP_address_value3=param('vtext3');

$checkbox_value1=param('vchk1');
$checkbox_value2=param('vchk2');
$checkbox_value3=param('vchk3');

$path="/opt/altq_in.conf";

print "<I><B><center><font size=\"+1\"> Graphical User Interface for Dynamic
Prioritization</font></center> </B></I><BR>\n";
print "<BR>\n";
print "<BR>\n";

if ( $radiovalue eq 'vid1')
{
###### ON off off
   if( $checkbox_value1 eq 'on' &&  !$checkbox_value2 &&  !$checkbox_value3 )
         {
           print "<center>Packets<b>\" from\"</b> IP address $IP_address_value are marked for
priority</center><BR> \n";
              open (FD,">$path") or die("Couldn't output.conf\n");
              print FD "interface xl0 \n";
              print FD " conditioner xl0 marker1 <mark 0x20> \n";
              print FD " filter xl0 marker1 0 0 $IP_address_value 0 0 \n ";
              close(FD);
           &altq_hup_signal_routine;
         }
##### off ON off
   if( !$checkbox_value1  &&  $checkbox_value2 eq 'on' &&  !$checkbox_value3 )
        {
           print "<center>Packets <b>\" from\"</b>IP address $IP_address_value2 are marked for
priority</center><BR> \n";
           open (FD,">$path") or die("Couldn't output.conf\n");
           print FD "interface xl0 \n";
           print FD " conditioner xl0 marker1 <mark 0x20> \n";
           print FD " filter xl0 marker1 0 0 $IP_address_value2 0 0 \n ";
           close(FD);
           &altq_hup_signal_routine;
        }

#### off off ON
   if( !$checkbox_value1 &&  !$checkbox_value2 && $checkbox_value3 eq 'on')
        {
             print "<center>Packets <b>\" from\"</b>IP address $IP_address_value3 are marked for
priority</center><BR> \n";
           open (FD,">$path") or die("Couldn't output.conf\n");
           print FD "interface xl0 \n";
           print FD " conditioner xl0 marker1 <mark 0x20> \n";
           print FD " filter xl0 marker1 0 0 $IP_address_value3 0 0 \n ";
           close(FD);
           &altq_hup_signal_routine;
        }
### ON ON off
   if( $checkbox_value1 eq 'on' &&  $checkbox_value2 eq 'on'  && !$checkbox_value3)
        {
             print "<center>Packets <b>\" from\"</b>IP address $IP_address_value are marked for
priority</center><BR> \n";
             print "<center>Packets <b>\" from\"</b>IP address $IP_address_value2 are marked
for priority</center><BR> \n";
           open (FD,">$path") or die("Couldn't output.conf\n");
           print FD "interface xl0 \n";
```

```
            print FD " conditioner xl0 marker1 <mark 0x20> \n";
            print FD " filter xl0 marker1 0 0 $IP_address_value 0 0 \n ";
            print FD " conditioner xl0 marker2 <mark 0x40> \n";
            print FD " filter xl0 marker2 0 0 $IP_address_value2 0 0 \n ";
            close(FD);
            &altq_hup_signal_routine;
        }
### ON off ON
   if( $checkbox_value1 eq 'on' &&  !$checkbox_value2  && $checkbox_value3 eq 'on')
        {
              print "<center>Packets <b>\" from\"</b>IP address $IP_address_value are marked for
priority</center><BR> \n";
            print "<center>Packets <b>\" from\"</b>IP address $IP_address_value3 are marked for
priority</center><BR> \n";
            open (FD,">$path") or die("Couldn't output.conf\n");
            print FD "interface xl0 \n";
            print FD " conditioner xl0 marker1 <mark 0x20> \n";
            print FD " filter xl0 marker1 0 0 $IP_address_value 0 0 \n ";
            print FD " conditioner xl0 marker2 <mark 0x40> \n";
            print FD " filter xl0 marker2 0 0 $IP_address_value3 0 0 \n ";
            close(FD);
            &altq_hup_signal_routine;
        }
#### off ON ON
   if( !$checkbox_value1 &&  $checkbox_value2 eq 'on'  && $checkbox_value3 eq 'on')
        {
              print "<center>Packets <b>\" from\"</b>IP address $IP_address_value2 are marked
for priority</center><BR> \n";
            print "<center>Packets <b>\" from\"</b> IP address $IP_address_value3 are marked for
priority</center><BR> \n";
            open (FD,">$path") or die("Couldn't output.conf\n");
            print FD "interface xl0 \n";
            print FD " conditioner xl0 marker1 <mark 0x20> \n";
            print FD " filter xl0 marker1 0 0 $IP_address_value2 0 0 \n ";
            print FD " conditioner xl0 marker2 <mark 0x40> \n";
            print FD " filter xl0 marker2 0 0 $IP_address_value3 0 0 \n ";
            close(FD);
            &altq_hup_signal_routine;
         }
### ON ON ON
   if( $checkbox_value1 eq 'on'&&  $checkbox_value2 eq 'on'  && $checkbox_value3 eq 'on')
        {
            print "<center> Packets <b>\" from\"</b>IP address $IP_address_value are marked for
priority</center><BR> \n";
            print "<center>Packets <b>\" from\"</b>IP address $IP_address_value2 are marked for
priority</center><BR> \n";
            print "<center> Packets <b>\" from\"</b>IP address $IP_address_value3 are marked for
priority</center><BR> \n";
            open (FD,">$path") or die("Couldn't output.conf\n");
            print FD "interface xl0 \n";
            print FD " conditioner xl0 marker1 <mark 0x20> \n";
            print FD " filter xl0 marker1 0 0 $IP_address_value 0 0 \n ";
            print FD " conditioner xl0 marker2 <mark 0x40> \n";
            print FD " filter xl0 marker2 0 0 $IP_address_value2 0 0 \n ";
            print FD " conditioner xl0 marker3 <mark 0x60> \n";
            print FD " filter xl0 marker3 0 0 $IP_address_value3 0 0 \n ";
            close(FD);
            &altq_hup_signal_routine;
        }
### OFF OFF OFF
   if( !$checkbox_value1 &&  !$checkbox_value2 && !$checkbox_value3 )
        {
            print "<center><b>\"NO\"</b> packets are marked for priority</center><BR> \n";
            open (FD,">altq_in.conf") or die("Couldn't output.conf\n");
            close(FD);
        }
}
else
{
    ### ON off off
       if( $checkbox_value1 eq 'on' &&  !$checkbox_value2 &&  !$checkbox_value3 )
```

```
              {
                 print "<center>Packets <b>\"for\"</b> IP address $IP_address_value are marked
for priority</center><BR> \n";
                  open (FD,">$path") or die("Couldn't output.conf\n");
             print FD "interface xl0 \n";
                 print FD " conditioner xl0 marker1 <mark 0x20> \n";
                  print FD " filter xl0 marker1 $IP_address_value 0 0 0 0 \n ";
             close(FD);
               &altq_hup_signal_routine;
               }
    ### OFF ON OFF
       if( !$checkbox_value1 &&  $checkbox_value2 eq 'on' &&  !$checkbox_value3 )
               {
              print "<center>Packets <b>\"for\"</b> IP address $IP_address_value2 are marked for
priority</center><BR> \n";
                      open (FD,">$path") or die("Couldn't output.conf\n");
                      print FD "interface xl0 \n";
                      print FD " conditioner xl0 marker1 <mark 0x20> \n";
                      print FD " filter xl0 marker1 $IP_address_value2 0 0 0 0 \n ";
                      close(FD);
                      &altq_hup_signal_routine;
             }
    ### off off ON
       if( !$checkbox_value1 &&  !$checkbox_value2 &&  $checkbox_value3 eq 'on')
               {
                      print "<center>Packets <b>\"for\"</b> IP address $IP_address_value3 are
marked for priority</center><BR> \n";
                      open (FD,">$path") or die("Couldn't output.conf\n");
                      print FD "interface xl0 \n";
                      print FD " conditioner xl0 marker1 <mark 0x20> \n";
                      print FD " filter xl0 marker1 $IP_address_value3 0 0 0 0 \n ";
                      close(FD);
                      &altq_hup_signal_routine;
               }
    ### ON ON off
       if( $checkbox_value1 eq 'on' &&  $checkbox_value2 eq 'on'  && !$checkbox_value3)
       {
                      print "<center>Packets<b>\" for\"</b> IP address $IP_address_value are
marked for priority</center><BR> \n";
                      print "<center>Packets<b>\" for\"</b> IP address $IP_address_value2 are
marked for priority</center><BR> \n";
                      open (FD,">$path") or die("Couldn't output.conf\n");
                      print FD "interface xl0 \n";
                      print FD " conditioner xl0 marker1 <mark 0x20> \n";
                      print FD " filter xl0 marker1 $IP_address_value 0 0 0 0 \n ";
                      print FD " conditioner xl0 marker2 <mark 0x40> \n";
                      print FD " filter xl0 marker2 $IP_address_value2 0 0 0 0 \n ";
                      close(FD);
                      &altq_hup_signal_routine;
               }
    ### ON off ON
       if( $checkbox_value1 eq 'on' &&  !$checkbox_value2 && $checkbox_value3 eq 'on')
               {

                      print "<center>Packets<b>\" for\"</b> IP address $IP_address_value are
marked for priority</center><BR> \n";
                      print "<center>Packets<b>\" for\"</b> IP address $IP_address_value3 are
marked for priority</center><BR> \n";
                      open (FD,">$path") or die("Couldn't output.conf\n");
                      print FD "interface xl0 \n";
                      print FD " conditioner xl0 marker1 <mark 0x20> \n";
                      print FD " filter xl0 marker1 $IP_address_value 0 0 0 0 \n ";
                      print FD " conditioner xl0 marker2 <mark 0x40> \n";
                      print FD " filter xl0 marker2 $IP_address_value3 0 0 0 0 \n ";
                      close(FD);
                      &altq_hup_signal_routine;
       }
    ### off ON  ON
       if( !$checkbox_value1 &&  $checkbox_value2 eq 'on'  && $checkbox_value3 eq 'on')
               {
```

```
                    print "<center>Packets<b>\" for\"</b> IP address $IP_address_value2 are
marked for priority</center><BR> \n";
                        print "<center>Packets<b>\" for\"</b> IP address $IP_address_value3 are
marked for priority</center><BR> \n";
                        open (FD,">$path") or die("Couldn't output.conf\n");
                        print FD "interface xl0 \n";
                        print FD " conditioner xl0 marker1 <mark 0x20> \n";
                        print FD " filter xl0 marker1 $IP_address_value2 0 0 0 0 \n ";
                        print FD " conditioner xl0 marker2 <mark 0x40> \n";
                        print FD " filter xl0 marker2 $IP_address_value3 0 0 0 0 \n ";
                        close(FD);
                        &altq_hup_signal_routine;
                }
    ### ON ON ON
        if( $checkbox_value1 eq 'on'&&  $checkbox_value2 eq 'on'  && $checkbox_value3 eq 'on')
                {
                        print "<center>Packets<b>\" for\"</b> IP address $IP_address_value are
marked for priority</center><BR> \n";
                        print "<center>Packets<b>\" for\"</b> IP address $IP_address_value2 are
marked for priority</center><BR> \n";
                        print "<center>Packets<b>\" for\"</b> IP address $IP_address_value3 are
marked for priority</center><BR> \n";
                       open (FD,">$path") or die("Couldn't output.conf\n");
                        print FD "interface xl0 \n";
                        print FD " conditioner xl0 marker1 <mark 0x20> \n";
                        print FD " filter xl0 marker1 $IP_address_value 0 0 0 0 \n ";
                        print FD " conditioner xl0 marker2 <mark 0x40> \n";
                        print FD " filter xl0 marker2 $IP_address_value2 0 0 0 0 \n ";
                        print FD " conditioner xl0 marker3 <mark 0x60> \n";
                        print FD " filter xl0 marker3 $IP_address_value3 0 0 0 0 \n ";
                        close(FD);
                        &altq_hup_signal_routine;
                }
    ### off off off
        if( !$checkbox_value1 &&  !$checkbox_value2 && !$checkbox_value3 )
                {
                        print "<center><b>\" NO\"</b>packets are marked for priority</center><BR>
\n";
                        open (FD,">altq_in.conf") or die("Couldn't output.conf\n");
                        close(FD);
            }
        }

sub altq_hup_signal_routine
{
        $machine = 'localhost';
        $port = 8888;
      $proto=getprotobyname ('tcp');
      $iaddr=inet_aton($machine) or die "no host: $machine";
      $paddr=sockaddr_in($port,$iaddr);
      socket(HUPALTQ, PF_INET, SOCK_STREAM, $proto) or die "socket: socket is dead";
      connect(HUPALTQ, $paddr) or die "connect: Connect Failed";
      close(HUPALTQ);
}

print "<html>\n";
print "<head>\n";
print "<title>ALTQ Graphical user Interface</title>\n";
print "<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=iso-8859-1\">\n";
print "</head>\n";
print "<body bgcolor =\"#AABBFF\">\n";
print "<BR>";
print "<BR>";
print "<BR>";
print "<form action=\"draft.cgi\" method=\"post\" name=\"testform\">\n";
print "<table align=\"center\" width=\"75%\">\n";
print "<tr>\n";
print "</tr>\n";
print "<tr>\n";
print "<tr>";
```

```
print "<td align=\"right\"><input type=\"radio\" name=\"video\" value=\"vid1\">
Remote Video</td>";
print "<td align=\"center\"><input type=\"radio\" name=\"video\" value=\"hst1\">
Client Machine</td>";
print "<tr>";
print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";


print "<tr><td align=\"right\">IP Address 1</td>\n";
print "<td align=\"center\"><input type=\"text\" name=\"vtext1\"
value=\"$IP_address_value\"><input
type=\"checkbox\" name=\"vchk1\" value=\"on\"></td></tr>\n";

print "<tr ><td align=\"right\">IP Address 2</td>\n";
print "<td align=\"center\"><input type=\"text\" name=\"vtext2\"
value=\"$IP_address_value2\"><input
type=\"checkbox\" name=\"vchk2\" value=\"on\"></td></tr>\n";

print "<tr><td align=\"right\">IP Address 3</td>\n";
print "<td align=\"center\"><input type=\"text\" name=\"vtext3\"
value=\"$IP_address_value3\"><input
type=\"checkbox\" name=\"vchk3\" value=\"on\"></td></tr>\n";

print "</tr>\n";
print "<tr>\n";

print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";
print "<tr><td></td></tr>\n";

print "<td colspan=\"2\" align=\"center\"> <input type=\"submit\"
name=\"submit\" value=\"submit\"></td>\n";
print "</tr>\n";

print "</table>\n";
print "</form>\n";
print "</body>\n";
print "</html>\n";
```