Investigation of a Preemptive Network Architecture

by

Christopher James Lefelhocz

**Submitted to the Department of
Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements
for the Degree of**

Master of Science in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology
May, 1994

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

Investigation of a Preemptive Network Architecture

by

Christopher James Lefelhocz

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 1994
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering and Computer Science

ABSTRACT

Two network architectures, cell and packet, form the basis of most high bandwidth network research. If analyzed from the perspective of building a switch, both architectures have unique advantages. The preemptive architecture described herein proposes to create a network that takes unique advantages from both the cell and packet architectures. Specifically, the advantages gained were in bounding delay and lower required processing power. Also investigated was the possibility that Required Processing Power(RPP) of a preemptive switch will increase at a lower rate compared to that of a cell or packet architecture, as port size and/or link speed increase. To adequately understand the power of a preemptive architecture, simulation was used to estimate the RPP needed for a switch.

The results show the feasibility of such an architecture. Two sets of simulations were run. The first used general poisson sources with two levels of priority. Simulations found upwards of 30 ports could be loaded with only 146 MIPS of processing power without serious degradation in end-to-end delay. The second used TCP bulk data and Variable Bit Rate(VBR) video encoding sources. Simulation results show upwards of 10 ports could be loaded with only 146 MIPS of processing power. The 146 MIPS of processing power is equivalent to one port in a cell architecture. Thus, the preemptive switch architecture gives at least an order of magnitude in savings in comparison to the cell architecture.

A feature of the traffic mix used in simulation was recognized as critical to minimizing RPP. When the source traffic is grouped correctly, the two groups formed have a bimodal characteristic. The bimodal characteristic is that one group dominates the link bandwidth and the other group dominates the units processed per second. The report shows how this characteristic can be exploited to help reduce RPP without loss in performance of the preemptive architecture.

Thesis Supervisor: Dr. David D. Clark
Title: Senior Research Scientist, MIT

Acknowledgments

Special thanks to Andrew Heybey, Oumar Ndiaye, Timothy Shepard, Karen Sollins, and John Wroclawski and the rest of Advanced Network Architecture group. They made the past two years of research a fun and often inventive experience.

Karen Ho, Janey Hoe, and Lisa Taylor gave me the inspiration to continue my thesis until completion. Their dedication and perseverance was honorable and often motivated me beyond my own expectations.

Another person who helped me was Ye Gu. He should be regarded for his dedication and compassion. Often times he reminded me of the goals I had set for myself. At the same time he reminded me that hard work should be rewarded with hard play.

I would like to thank Janey Hoe and Kevin Lew for comments on the final drafts of this thesis and following report.

I would like to thank two of my former "advisors" Fredric D. Luthy and Ronald P. Bianchini. The former is a Junior High School teacher of math who introduced a 7th grader to the wonderful world of computer science. The latter is a B.S. advisor who motivated an undergraduate to pursue even more research and showed selflessness in letting him decide where to attend graduate school.

I also thank my current advisor, David D. Clark, whose calm demeanor and large knowledge base helped in my pursuit of this report. His understanding of my physical limitations was a consideration that few students find in a graduate advisor. Also, his often gentle pushing and prodding helped in my understanding and often motivated me to further explore the work I was doing.

For support and understanding beyond any expectation, I thank Julie Stein and Carrie Gray. Both of these women represent a special part of my life and showed special support when it was most needed.

Finally, I would like to thank my family. The unspoken support of my brothers John W., Paul, and Douglas was all I could ask for. My mother and father, Charlotte P. and John F., have always stood one-hundred percent behind my decisions even though they didn't always understand them.

This research was supported by the Advanced Research Projects Agency under contract DABT63-92-C-0002, GTE under Research Agreement dated 10/1/92, and the Corporation for National Research Initiatives.

Contents

Chapter 1

Introduction

A noble goal in networking today is to create a cheap, fast, high capacity network that integrates many different types of service. While this goal is attainable, there are differing views as to how the network should be designed. The primary architectures under consideration are the packet and cell architectures. From the switch perspective, a packet network has the advantage of requiring less processing power. A cell network has the advantage of easier integration of services. Can the advantages of both architectures be combined to form a new architecture? This report presents an alternative architecture, which explores this exact question.

1.1 Motivation

The latter part of the goal stated above is to be able to integrate many different types of traffic into one communications substrate. The variance in the types of traffic ranges from real-time traffic to data traffic. Each class of traffic has its own set of requirements for acceptable transmission. For instance, real-time traffic has requirements relating to information being sent in a timely fashion with little emphasis on the guarantee that all information is received. In contrast, data traffic has requirements relating to the information being sent in a less timely fashion, but requiring acknowledgment and verification that information is correctly received. Each class of traffic will have a different set of requirements. The objective of the integrated network is to meet all requirements for the different classes where possible.

Another part of the goal is to be able to build the integrated network cheaply. There are many choices that affect the cost of the network. Among the choices is the design and implementation of the switches in the network. A primary component that determines the cost of a switch is the amount of processing power it contains. If the processing power can somehow be minimized, then the cost of the switch should be less. For this report we look at the specific task of minimizing the processing power in the switch.

Two switching architectures that are either being proposed or in use today are called the packet and the cell architectures. The packet architecture works on the principle that the end-to-end unit called the packet is the unit of transfer throughout the network. Packets can be of variable size and represent a piece of the data being sent. The cell architecture differs by making the packets be a fixed size. The capability gained is to be able to preempt data we are sending at fixed intervals. Preemption, or the capability to interrupt transmission of some unit of data to transmit an unrelated unit of data, offers the advantage of bounding delay for a set of data being transmitted. This bounding of delay makes integration of services easier.

Since packets are generally larger than cells, the processing power needed to process in a packet network is higher than the processing power needed in a cell network. However, as was mentioned above, cell networks can integrate services more easily. The goals stated above imply that a hybrid solution should be found if possible. In other words, our goal is to find an architecture where only the packet processing power is required, but ease in integration of services is the same as in a cell network.

1.2 A Possible Alternative Solution

The new architecture will be called the *preemptive* architecture. Its primary advantage is that unlike both cell and packet architectures, the local unit of transfer can be preempted. This is believed to give us three important capabilities. The first is the ability to change what data is sent immediately. This leads to bounding of total delay (cell advantage). The second is the ability to make local data units larger. This gives us a lower required processing power (packet advantage). Finally, the capability to make units larger and integrate them easily on a local basis facilitates the increase of a switch's port size or link bandwidths.

1.3 Results of Simulation

The results presented in this report have a two-fold objective. The first is to present arguments for the feasibility of the preemptive architecture developed and presented in Chapters 2 and 3. The second is to present the interesting effect priority has on the capability of a preemptive system. Sections 4.2 and 4.3 present the Comparison and Priority simulations respectively.

The Comparison simulations will use the standard traffic mix presented therein and compare the Switch Processing Power(SPP) of a preemptive architecture to the SPP of a cell architecture. Results show that the preemptive architecture uses about 30 times less processing power than the cell architecture for these simulations. This reduction in processing power can be exploited in three ways. The first is obviously a cheaper switch with less processing power included in the design. The second is a large scalable switch which uses the same amount of processing power as the cell architecture, but has a large port size. Finally, the third is a scalable switch which contains links with higher bandwidths than the cell architecture. These features gained by reducing processing power required in a preemptive architecture can be used either singularly or combined to form an new more powerful switch.

The Priority simulations recognize that the traffic mix can be broken into two groups. The two groups are large end-to-end units and small end-to-end units. With respect to the bandwidth used, large end-to-end units dominate over small end-to-end units. With respect to the number of units to process per second, small end-to-end units dominate over large end-to-end units. This bimodal characteristic can be exploited to further the reduction of processing power. In the original traffic mix, large end-to-end units could preempt small end-to-end units. By modifying it so that small end-to-end units preempt large end-to-end units, the bimodalness of the traffic mix can be exploited. This exploitation results in an improvement in the end-to-end delays found from simulation.

1.4 Organization of Report

In Chapter 2 the fundamental design of the preemptive network is explained. The basis for most design decisions is presented here. These principles are the primary drive to define the network architecture. Chapter 3 presents a host and switch model of possible implementations given the network architecture described in Chapter 2. These models will be used in simulation. They also are used to justify the effectiveness of both the network and switch architectures. Chapter 4 presents the simulation results. First, the simple poisson source is presented. Next the TCP bulk transfer along with Variable Bit Rate(VBR) video encoding is presented. Finally, Chapter 5 presents the major conclusions of the simulation results and presents possible future work based on the results attained thus far.

1.5 Related Work

Due to the novelty of this work, there is not a large selection of background material. The related work found can be broken into three categories. The three areas are general queueing theory on preemption, preemptive systems in a shared medium environment, and preemption strategies in ATM. These areas cover mainly the performance of a preemptive system, but do not explore the required processing power of such a system. This differs from my work which is to explore the trade-off of required processing power versus the performance of a system.

A number of sources exist which cover priority queueing. The preemptive architecture described above is classified as a preemptive resume queueing strategy. Both [1] and [10] cover how a M/G/1 system performs with a preemptive resume strategy of two priority classes. Our system is slightly more complex than either of the M/G/1 models presented. This work may be used to help in attempting to characterize the architecture performance in a closed form. However, getting a closed form would be difficult and is not necessary to evaluate the system.

Another area where a preemptive approach has been used is in the shared medium network. These approaches are used to gain better performance for the expected delay of real-time traffic. The two mediums are a slotted ring [9] and distributed queue dual bus [8]. Since this is a shared medium, there is no required processing power unless we estimate the processing power necessary at each connection point. This work is important since it may be an input to an actual implementation. Thus, if time permits, we may use this work to simulate a set of sources on a shared medium which have preemption capability across that medium to a switching network.

Finally, as with any new architecture there is bound to be a comparison to a rival architecture that is popular at the time. The rival architecture in this case is Asynchronous Transfer Mode(ATM). ATM uses the small cell architecture with each cell being 53 bytes (currently) in length. ATM can be considered a preemptive resume system if the correct algorithm is used to decide how cells are sent. Such a scheme is discussed in [2] where a discrete model of preemption is presented. ATM is not used in this research for two reasons. The first is that our characterizations can be more general and not restricted to ATM. The second is that it is easier to examine the preemptive architecture without the constraints of the ATM standard. If our research is successful, then a probable direction is to engineer the preemptive architecture into an ATM environment. For these reasons we will look at preemption more generally and leave it to engineering to incorporate the basic ideas into the ATM environment.

Chapter 2

Fundamental Design

2.1 Motivation Revisited

As stated in the Introduction, the traffic types we expect to handle range from real-time to data. Real-Time traffic is the traffic which has a constrained time-delay bound. An example of real-time traffic is full-motion video which has several frames per second of data to be transferred in a timely fashion. Data traffic on the other hand has a flexible delay bound that varies from transfer to transfer. An example of this is a file transfer. The goal of integrating traffic from real-time to data is a very interesting area of research and as part of this work, we will examine some features of such a network.

In examining an integrated network, we will look specifically at the switch device. The switch device is a router of information. It is responsible for connecting several machines together to communicate. The switch is also responsible for handling the routing, processing, and transferring of these pieces of information in the network. Our primary concerns in measuring the performance of a switch are the delay incurred on real-time traffic being transferred through the switch and the *Switch Processing Power*(SPP). Delay affects the overall network performance. SPP affects the design and implementation of a switch. SPP is related to delay in that SPP can control the delay bound on information being transmitted by the switch. To understand this relationship we will define both delay and SPP more rigorously.

2.1.1 Delay in an Integrated Network

Total delay is defined as the interval of time between when an end-to-end unit begins transfer from the source to the end of transfer at the sink. The data unit can either be a packet, cell, or an ADU(defined below). Each data unit transferred between a source-sink pair will have a delay which is dependent on the physical distance between the source-sink pair, the size of the data unit, the other traffic sent partially or completely on the route, and the processing by intermediate switches in the route of a data unit. We call these four factors, *propagation, transfer, queueing,* and *access* delay respectively. Propagation delay is dependent on the path taken and for our purposes is assumed to be fixed for a source-sink pair. Transfer delay is based on the lowest bandwidth link the data unit is transferred through. It is also assumed to be roughly fixed. Queueing delay is dependent on the network traffic and algorithm used to order data units being sent in the network. While this is not fixed, it is not our primary interest of study in this work. We use a simple algorithm to order data for transmission. Access delay is dependent on the architecture chosen. This is our primary interest for study in this work.

As stated above, access delay is the processing by intermediate switches in the route of a data unit. Access delay can be further broken into two parts. The first is the time spent identifying, copying, and scheduling a data unit for transfer from the input to the output of a switch. We call this the *Work Delay.* The second part of access delay is the time the data unit waits for another data unit to end transmission on a link. This is called *Work Transfer Delay.* Both work delay and work transfer delay are determined by the network and switch architectures chosen. However, there are a few basic properties that are general over most architectures.

Work delay is the time to process one data unit. Thus, the inverse of the work delay is the number of data units that can be processed in a time interval or its rate. On the average, for a switch to have good performance, the number of incoming data units to be processed must be less than the number of data units that can be processed. This is shown by equation 2.1 with both sides being in units of cells, packets, or ADUs per second.

$$data\ unit\ rate \leq \frac{data\ unit}{work\ delay} \tag{2.1}$$

If the incoming data units is more than the inverse of the work delay then queueing occurs before the processor. This means that data must wait to be processed. While the data waits for processing time, the link will go idle. When the link goes idle utilization will drop and performance will degrade. Therefore, we want to keep the work delay small enough to be able to process data units without substantial queueing.

The amount of work transfer delay for one data unit is dependent on the time at which the unit has completed being processed. If the link is idle after processing then the work transfer delay is zero. However, if another data unit is being transferred then we have work transfer delay. As the size of data units grow the amount of time waiting on another data unit to finish transfer increases. Thus, the amount of work transfer delay increases. As it increases, the end-to-end delay increases. At some point the end-to-end delay becomes unacceptable for some traffic. The simple solution is to keep data unit sizes small and then the work transfer delay is minimized. However, choosing the correct size for a network is a tough problem which is discussed in 2.2.1.

Combining work and work transfer delays, we can examine the overall effect of varying access delay. This is shown in Figure 2-1. This is an example of a plot of three histograms of total delay on the same graph. Each histogram represents a different plot of total delay based on the same source-sink pair with the same traffic characteristics. The only variance is in the access delay. Assuming that to a first approximation the queueing delay stays constant, as access delay increases, the total delay increases. This is shown by the equation $ad_1 < ad_2 < ad_3$, where ad_x is the access delay of plot x. Thus, the average delay increases and the histogram plot moves to the right. At some point the end-to-end delay for most end-to-end units sent becomes unacceptable. This is denoted by the dotted vertical line labeled unacceptable delay. Given that we know what the unacceptable delay is (a choice for the implementor), we can find what the maximum access delay will be. In section 2.3, we examine some of the advantages to making the access delay longer and its effects on the network architecture.

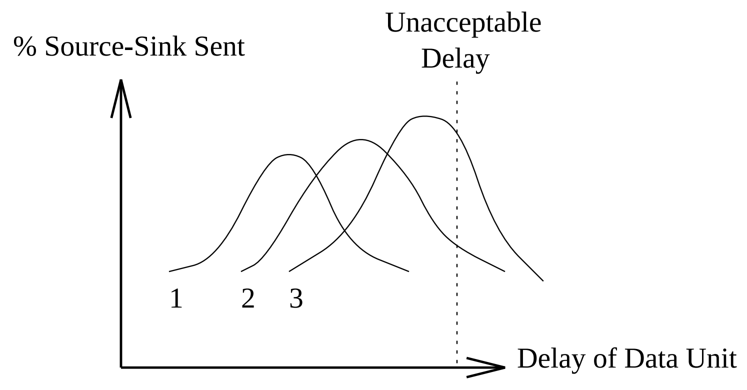

Figure 2-1: Three Histogram Plots where $ad_1 < ad_2 < ad_3$

2.1.2 Switch Processing Power

Switch processing is the capability to process incoming pieces of information for transmission. The amount of processing in a switch is the *Switch Processing Power*(SPP). We quantify SPP in units of *millions of instructions per second*(MIPS). SPP is strictly the amount of power we are given. A higher SPP means that we have more instructions to process data with. This can be either to make more complex algorithms or to service more pieces of data. In either case, given a good implementation, an increase in SPP should see a decrease in delay.

How do we know if we have enough SPP for a switch architecture? By comparing it to the *Required Processing Power*(RPP). RPP is the amount of processing power necessary to guarantee a certain access delay on a link. Thus $port_i$ on a switch has a RPP_i which must be met by the SPP or more concretely:

$$SPP \geq \sum_{i=1}^{N} RPP_i \tag{2.2}$$

There are two important points to recognize about this simple equation. The first is that it is a simplification of the relation. The SPP may not be available to all ports of the switch (i.e. SPP may be a summation of the processing power at each port). The second is that the RPP varies over time. Depending on the traffic of a particular link, RPP will change.

It is important to understand the difference between SPP and RPP. SPP can be thought of as the current capacity of the switch whereas RPP is the capacity necessary to maintain low access delay. RPP is determined by the capacity of the links, the size of the switch, and the architecture. SPP is determined by how much money was invested into the switch. Keeping SPP above RPP for a particular switch is obviously preferred. Minimizing RPP is even better.

2.2 Two Traditional Architectures

We now look at two traditional network architectures to see how they affect the access delay and SPP. The two architectures will be called the packet architecture and the cell architecture. These architectures are not tied to a particular implementation or standard. Rather they are models of what currently is being examined or in use today. By examining this work, I hope to convey the problems in building either system with respect to SPP and access delay.

The following three equations are used as a basis for much of the discussion explaining why the architectures have problems.

$$access\ delay = work\ delay + work\ transfer\ delay \tag{2.3}$$

$$RPP = \frac{processing\ instructions}{data\ unit}\frac{data\ unit}{work\ delay} = \frac{processing\ instructions}{work\ delay} \tag{2.4}$$

$$data\ unit\ time = \frac{data\ unit\ size}{link\ bandwidth} \tag{2.5}$$

Equation 2.3 represents the access delay of a switch from the definition in 2.1.1. Equation 2.4 represents the RPP of a switch link. One way to think about the RPP equation is the number of instructions required per data unit times the rate of data units coming in, which yields the number of instructions per second. As can be seen, RPP is dependent on the number of instructions to process and the work delay to do the processing. Finally, equation 2.5 refers to the relationship between a data unit's size in bits to its size in seconds. The size in seconds is simply the time it takes to transmit the data unit on a link of a particular bandwidth. Given two different links with two different bandwidths, the same data unit will have two different data unit times even though the data unit's size is constant.

2.2.1 The Packet Architecture

The first architecture we look at is the packet architecture. A packet is a unit of data supplied by the source to the network. While in the network, a packet can be broken into smaller pieces for transmission. We call the breaking of any end-to-end unit into smaller pieces *fragmentation*. A packet may only be fragmented by a switch or other device before it begins transmission. Packets and fragments are of variable length. The transmission of one packet/fragment can not be interrupted to send another packet/fragment. The packet/fragment size is dependent on what end-to-end applications and congestion control algorithms are being used.

A problem with the packet architecture is that the access delay is dependent on the packet time, ie. the larger the packets sent, the larger the access delay will be and the more likely that it will be unacceptable. If we examine access delay as the summation of work and work transfer delay then we see that for a given processing power and bandwidth: 1) work delay is constant and 2) the work transfer delay will vary. In general we can set the work delay to be as large as the average packet time. This minimizes RPP. It is important to note that increasing the RPP will decrease the work delay, but not decrease the access delay. Any time gained by decreasing the work delay only increases the time in the work transfer delay. This is since the packet time is fixed once transmission starts on a link. One way to get around this is to make smaller packets than are sent through the network. However, to pick a packet size we have to know the packet route, minimum bandwidth in the route, the traffic characteristics in the route, and the SPP at each switch along the route. If any switch is modified in the route, or the route changes then the packet size must be recalculated. If we can not do this calculation, then the access delay varies. With a large variance in access delay, there is no guarantee that data is always transferred in a timely fashion. Thus, real-time support is hard to maintain in such a network.

The primary disadvantage to the scheme above is not in the calculation, but rather in the control of the network. Upgrading or modifying a switch is a local change, but it has global effects. Since packet time helps to determine access delay, modifying the bandwidth of a switch in turn changes the access delay. Thus control algorithms must keep track of a switch's state to dynamically choose a packet time. This adds a lot of responsibility to the end-to-end control algorithms which may be unnecessary.

2.2.2 The Cell Architecture

The cell architecture changes the packet architecture by adding a smaller unit of transfer. In the cell architecture packets still exist. However, packets are broken into fixed size segments called cells. Thus, one way to view the cell architecture is a packet network with fixed fragmentation. Cells are the unbreakable unit and thus the smallest/largest unit of transfer end-to-end. A standard cell size is chosen for the whole network and the size does not vary depending on traffic or transmission speeds. Cell size is strictly dependent on the standard specified for the network.

For a cell architecture, the access delay of the system varies based on the link bandwidths of the network. Since a cell is a fixed size, for a fixed bandwidth it has a fixed cell time, and thus the access delay is the same magnitude as sending one cell. When the system is initially built, the time of a cell is determined by the minimum bandwidth in the network. We need to find the cell time which gives an access delay that is acceptable.

Since processing is done on a cell by cell basis and cells are a fixed size, the time devoted to work and work transfer delays are constant for a given link bandwidth. If link bandwidth increases, work and work transfer delays decrease since we must still maintain the cell by cell processing. As work delay decreases, the amount of time to "think" about which cell to send decreases as well. A work around solution is to make processing algorithms "dumber" which run faster. However, it is not clear that we can make dumber algorithms that run faster as quickly as the bandwidth of links increase. Furthermore even a tougher argument against this solution is that the control algorithms that are simpler at higher speeds may not be compatible with the more complex algorithms at lower speeds. To maintain the same algorithm running at higher speeds will result in an increase in RPP.

2.3 An Alternative Architecture

We have seen how two different network architectures have problems with modification of a switch. A number of possible solutions could be raised and tried at this point. Some solutions try to modify either the cell or packet architecture to correct the problems raised above. However, in an effort to better understand how the fundamental properties relate we start from scratch and examine an alternative architecture. The ideas and concepts raised in the alternative architecture could be used in either the packet and cell architectures. For now we start from fundamentals and move forward not concerning ourselves with integrating into current architectures.

We start from equations 2.3 and 2.4. Now suppose that the access delay to guarantee a total delay bound is given to the network designer. He now has the choice of how to break the access delay over the work and work transfer delays. Increasing the work transfer delay has the effect of decreasing the amount of work delay available and thus increasing RPP. Decreasing the work transfer delay has the opposite effect. Therefore the network designer will try to minimize the work transfer delay and maximize the work delay. In a cell architecture we do just that by setting the access delay to be the time that it takes to send a cell. However, also in a cell architecture, every cell is processed. Which means that we have only enough time to process based on the cell size. This means that:

$$RPP = \frac{proc.\ instructions}{work\ delay} = \frac{(link\ bandwidth)(proc.\ instructions)}{cell\ size} \tag{2.6}$$

This is a problem since if we increase the link bandwidth then the RPP increases. We would prefer to keep it constant.

The packet architecture has the opposite problem. The RPP is much lower, since our work delay can be as large as the average size of packets going through the link. However, our work transfer delay varies depending on the packet size. As stated above we must then decide on packet sizes to control the work transfer delay. This works nicely until we start to upgrade links which must support the smaller packets for other slower links in the network. RPP will not remain constant due to the support the smaller packets. Since:

$$RPP = \frac{proc.\ instructions}{work\ delay} = \frac{(link\ bandwidth)(proc.\ instructions)}{average\ packet\ size} \tag{2.7}$$

The average packet size will remain fairly constant until most of the links have been upgraded. So at least initially for the faster links, the work delay will decrease and consequently the RPP will increase.

It is clear that if the work transfer delay is decreased, then the work delay can be increased without loss in performance. Increasing the work delay lowers the RPP. However, in both the cell and packet architectures minimizing work transfer delay results in a binding (either forced or by upgrade) between the work delay and the end-to-end unit of transfer. If there is a mechanism which breaks the binding then a solution exists that could keep RPP minimized and at the same time control access delay. Such a mechanism exists and is called *Preemption*.

Preemption is the capability to stop transferring some piece of data and start transferring an unrelated piece of data. Preemption in our case will be assumed to be an instantaneous event. With preemption we break the relationship between work delay and the end-to-end unit of transfer. Now both can be used to minimize the RPP of the switch. The expectation is that by using preemption the RPP will be lower than either of the two traditional architectures. Also by using preemption, the bandwidth of the links can change without drastically changing the network architecture. We will see why after we define end-to-end and local units of transfer.

2.3.1 Network Data Unit

When creating a preemptive architecture the first realization is that the definitions for packet and cell do no fit as a basis for the unit of transfer. The primary difference is that the size of the local unit of transfer is not dependent on the end-to-end unit of transfer, but rather on the state of the port locally. This is due to preemption where a port can choose to break a end-to-end transfer unit into two or more local transfer units of any size at any time. A packet network does not have this capability and a cell network fixes the breaking point. We call the new local transfer unit a *Network Data Unit*(NDU). An NDU is the localized unit of transfer which represents part of an end-to-end unit of transfer. A set of NDUs are combined to form the end-to-end unit. An NDU can be of any size to represent part of, or the whole end-to-end unit.

An NDU is used to transfer data locally on a link. Another transfer unit is needed to give us an end-to-end property as well as a method of combining NDUs locally in switches. This transfer unit is built out of one or more NDUs. We call this transfer unit an *Application Data Unit*(ADU).

2.3.2 Application Data Units

An ADU is defined in [3] as a "suitable aggregate of data from an application for which frame boundaries are preserved by the lower levels of the network." For our purposes we define it as the data supplied to the network by some source for transmission and is expected as the same unit by some sink. While in the network, the ADU is represented by one or more NDUs on each link it traverses. The NDU sizes can vary from link transfer to link transfer. At the switch, each NDU is broken into its data from the ADU and its control information. The data is then recombined with any ADU data previously received from other NDUs. The control is used to identify the NDU as the beginning of a new ADU or a portion of an ADU already partly sent. For the ADU to be transmitted on the next link as an NDU, the ADU need not be completely present. Only a portion of the ADU needs to be present to be sent forward. Reassembly of the NDUs into an ADU can occur either at the sink, or the last switch depending on the implementation. An example follows which shows a short traversal of the network.

Figure 2-2, gives us an example to show how ADUs are broken and recombined. At the edge of the network, the ADU is created which is then sent to switch 1 as a complete NDU a. From switch one to switch two, the ADU gets broken into three separate NDUs (b, c, and d) which are transmitted separately. This is due to higher priority traffic preempting our data. From switch two to switch three two NDUs are needed to transmit the data (e and f). We note that f is the same size as d and contains the same data. However, we give it a new label since the NDU was recreated at the output link of switch two. Finally, the NDUs are received, recombined, and fragmented for transmission as two new NDUs (g and h).

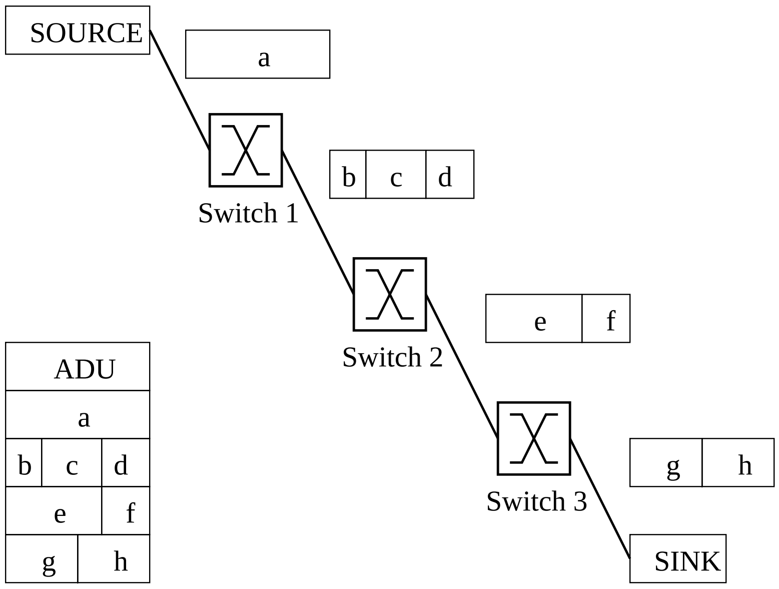

Figure 2-2: Equivalence of NDUs to ADU

2.3.3 How Preemption Helps

RPP will be lower for preemption because of the encouragement to dump large ADUs onto the network. Excluding reliability issues, dumping a large ADU on the network is preferred by both the host and network in a preemptive system. If for some reason the ADU is too large to be transferred in one unit then it can be broken into two or more NDUs. This provides a convenient mechanism to let real-time traffic constraints be met by an intermix of NDUs of real-time and data traffic. The user would rather process information in bulk since it requires less overhead at hosts with minimal loss of control in the network.

As bandwidth increases, if the NDU's time to send is constant then the size in bytes increases. Until the NDU size is as large as the ADU it represents, there is no change in access delay. When NDUs reach ADU sizes then access delay will begin to decrease and thus RPP will increase in proportion. However, in a real-time network, with video traffic, running out of bytes to send is not likely to happen. At this point, an ADU is just like a packet. Thus RPP will be as low as possible.

2.3.4 Cut Through

One of the problems that would exist in a naive preemptive architecture is reassembly of NDUs into ADUs at each switch. While Figure 2-2 shows how each ADU is broken into NDUs on a link by link basis, we wouldn't expect the complete ADU to be reassembled. For each switch to have complete reassembly be required would mean an added transmission delay at each switch. This is since complete reassembly requires that the complete ADU be received before transmission begins. As ADU sizes increase, the complete reassembly and transmission times increase as well. To solve this problem the preemptive architecture uses *Cut Through*.

Cut through is the capability to start sending data as soon as it has been received. We do not have to receive the complete ADU before we start sending it, rather we must only have the header to identify, process, and schedule the ADU. As NDUs are received they are recombined with any remaining pieces of an ADU to be sent and then sent as a larger NDU. Using cut through gives a pipelining mechanism seen in small packets and cells. It is the large transfer unit's answer to small transfer unit's store and forward.

Chapter 3

Implementation of Preemption

3.1 Host Link Model

The source of any data sent to a network must first go through a host to reach a switch. The host contains a connecting link to a switch. The host is responsible for a) generating ADUs from data received from applications, b) sending NDUs from host to switch, and c) receiving NDUs from a switch and reassembling into an ADU. The reassembly could be waived for applications that are stream oriented or are not concerned with ADU boundaries.

The implementation of such a device physically is in the form of a link card in the host computer. For our purposes in modeling we use the block diagram in Figure 3-1. There are five modules to the link card. They are the main bus, send and receive memory, processor, sending line, and receiving line. The main bus acts as an interface to the rest of the host. The memory is used to store ADUs for transfer and NDUs that are received for reassembly. The processor is used to control the flow of outgoing NDUs and reassembly of incoming NDUs. The receive and send lines are the circuitry to receive and send NDUs onto the network.

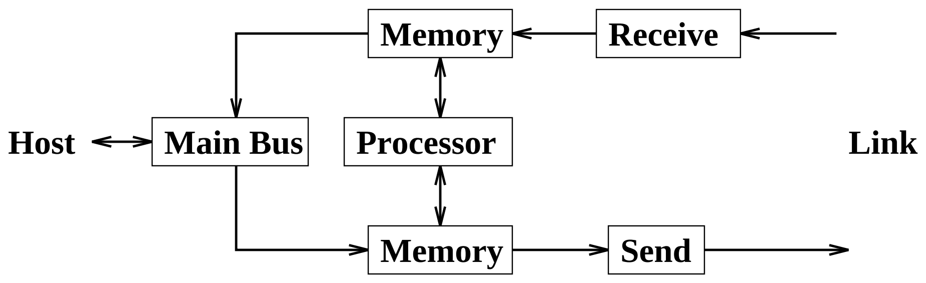

Figure 3-1: Block Diagram of Host Link Card

A number of implementations could result even from this block diagram. There are a hundred engineering decisions to be made such as whether the memory for sending and receiving should be shared, is the processor in reality the main processor for the host or an embedded controller, etc. However, our primary interest is in the algorithms used to send and receive NDUs. Many could be studied, but only one is tried since we are more interested in studying the dynamics of the switch architecture.

The congestion control algorithm chosen is a simple one since it is not the primary interest of this work. It is a basic priority scheme where each ADU is assigned a priority before entering the network and maintains the same priority throughout its transfer. ADUs are sent from highest priority to lowest priority. If two or more ADUs with the same priority contend for the link, then they are handled in a first come first serve manner. When an ADU A arrives having a higher priority than the ADU B currently being sent then ADU A preempts ADU B and breaks ADU B into two NDUs. Then ADU A is sent. Upon completion, the rest of ADU B is transmitted.

ADUs are only broken into two or more NDUs when preemption occurs. A list is kept of the remaining ADU data to be sent and the order for it to be sent. The processor is responsible for maintaining the list, setting up NDU headers, and choosing which ADU to drop if the memory buffer is to overflow. The last responsibility will not be examined at all in this work. Rather we assume that buffers are infinite, so there is no checking for memory overflow.

As the NDUs are received, they are combined to form the ADU. Once the complete ADU has been received it is transmitted from the hostlink card to the main computer. We are assuming for this work that there is enough buffer capacity to hold all the NDUs received before a complete ADU has arrived.

3.2 Switch Architecture

We now look at a preemptive switch architecture. By preemptive we mean that a switch can a) generate NDUs which may be preempted in the middle of transmission at its output ports, and b) handle preempted NDUs at its input ports. The internal switching and blocking of the switch will affect its performance, but for this exercise we will assume that it is ideal. Instead, we examine how the switch performs given a fixed time to process each ADU. We will first examine and improve a simple algorithm for processing of ADUs by the switch. This algorithm will then be used in the on going research to see what performance gains can be made from the preemptive system.

As NDUs are received by the switch, we must break the NDU into a payload and a header (or control). Once we break the NDU into the two separate pieces, the payload can be recombined with ADU data still in the switch. The header and some switch state are used to decide if any changes need to be made to the output scheduling. We call the operation of breaking the NDU destruction. The destructor is shown in Figure 3-2 along with the rest of a block diagram of the architecture.

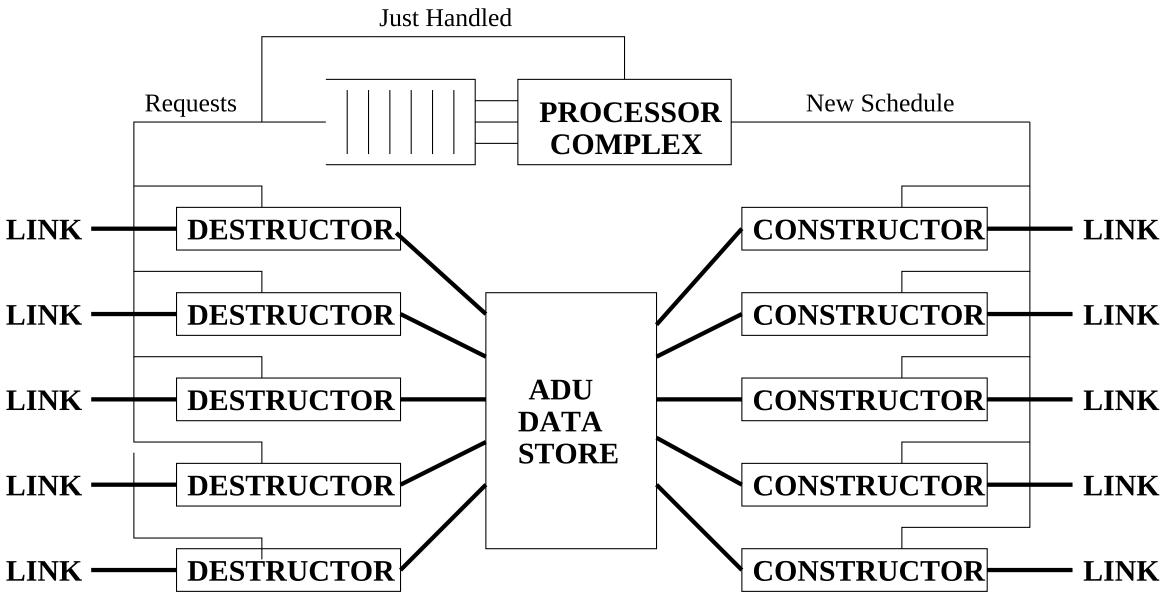

Figure 3-2: Switch Architecture

The ADU data is sent to the ADU data store by the destructor. The destructor then sends a request for processing of the ADU. There are two parts to processing an ADU by a switch. The first is the receiving, identification, and prioritization of data. The second is the scheduling of the data for transmission. We will call the first action the Handling of the data and the second action the Scheduling of the data. For this work we will assume these actions take roughly equal time to execute. Given the two processing actions we can create an algorithm to decide when and how to handle and schedule ADUs. The handling and scheduling operations are requested by the destructor and executed by the processor complex. The handling operation generates a scheduling request and the scheduling operation generates a schedule.

For a priority system, the simplest case is to have each NDU handled and scheduled immediately as it is received. The handler looks at the NDU and identifies its ADU priority and destination port. The scheduler then uses this information to create a schedule of the order of ADUs to be sent out a particular port. This algorithm requires that each NDU header be handled and scheduled once when executing. This can be improved in two ways. First we can reduce the number of handled NDU headers by caching. Second, we can reduce the number of schedule operations by having NDU data be recombined and inheriting the old ADU data's schedule.

To reduce the amount of handling going on in a switch we have to have some way to identify NDUs whose ADU is cached. So for a given ADU only one handling need occur. The first NDU initiates the handling and then each NDU after that belonging to the same ADU initiates a scheduling which uses cached information about the ADU. The destructor performs the caching and does not send the header to be handled if the ADU information is cached.

To reduce the amount of scheduling is slightly more difficult. Scheduling has to occur only when the destination port of an ADU does not include the ADU in its schedule. This is true when there is no data of an ADU in the switch. This occurs in two different situations. First, it occurs when the first NDU for an ADU arrives at the switch. Second, it occurs when an NDU arrives and all previous data received for that ADU has been transmitted to the next switch or destination. However, if ADU data exists in the switch then the two data segments may be combined. Once the two segments are combined, the data can then be transferred as one or more NDUs on the next transmission link. The destructor is again the hardware piece which identifies if the ADU data can be combined or must be scheduled. If it can be combined then the destructor does not generate any request. If it can not be combined, then the destructor generates a scheduling request.

Once a scheduling request is made, a list is generated by the scheduling algorithm of how ADUs should be transmitted. This list is then given to the constructor block of the switch located at the output transmission port. The constructor uses the list to construct NDUs which are transmitted on the link. The header may be a copy of an old NDU header that is passed on from the destructor in some fashion, or a new header to be sent with the NDU payload. From the input/output perspective, the switch processes NDUs, but internally it handles and schedules ADUs.

3.3 Simple Switch Processing Power Analysis

3.3.1 Complexity Module Analysis

A valid question to raise at this point is the issue of complexity of each module of the switch architecture. If the complexity outweighs the gain in SPP then the architecture will be no better than standard architectures proposed in the past. In particular we are concerned with showing the simplicity of the destructor and constructor. The processor complex is the switch processing power since it is an array of processors. The ADU data store has two functions: 1) storage of blocked ADUs and 2)switching the ADUs for transmission. The expected complexity for the memory and switching fabric is similar to those presented in [5], [7], and [11] since that is their primary function. Thus we should examine the destructor and constructor more closely to understand their complexities.

As was shown in the previous section the destructor has three functions to perform, A) identify an NDU with an ADU, B) strip data and store it away in the ADU store, and C) generate any handling and scheduling requests. To identify an NDU with its ADU is a straightforward procedure. Each NDU has an ADU ID in its header and we look up in a table the information on the ADU. Included in each table entry is ADU's status, ADU's data left to send, and if ADU has an outstanding request. The buffer for the table is small since the number of outstanding ADUs at any one time can be limited. As part of the ADU's status we know whether the ADU is new or not. If after looking up we find that the NDU just received is the first NDU of a new ADU then we can generate a request for handling. Requests are in the form of the NDU header of information along with the ADU table data. This functionality covers A and B above. To cover C, we look at the status, the left to send, and the outstanding request information in the table. If the we have data that hasn't been transmitted yet for that ADU then the new NDU data is stored with the old ADU data. Otherwise, a scheduling request is made. As can be seen all three functionalities can be taken care of by a state machine along with some memory for table space. The amount of processing done by the destructor is negligible compared to the processing done in the processor complex.

The constructor is even simpler than the destructor. It simply takes a schedule list of the ADUs and their corresponding NDUs to send them. Each entry in the schedule list contains the ADU to send, the NDU header, and the size of the NDU to send. The constructor requests data from the data store and then sends the data in the NDU to the next switch/host. The header is appended to the data for identification at the next switch. Upon completion or partial completion of sending an ADU, the constructor informs the data store, the processor complex, and the destructor of the amount of data sent for an ADU. Preemption occurs when a new schedule is generated and the head of the schedule differs from the ADU currently being sent. The constructor like the destructor can be done with a simple state machine and enough memory to store the schedule list.

3.3.2 Base and Upgrade Analysis

Given that we can build a switch that has the preemption capability, what kind of performance can we expect from the network? This is the question we choose to ask in this research. We are interested in seeing how little processing power is needed while keeping the performance of the system acceptable. Below is a brief argument as to how the preemptive architecture will perform better than the two other architectures described above. The objective of this report is to back up these arguments with concrete simulation data and models of how the systems perform.

To keep the total delay acceptable, we must gauge what access delay is acceptable to the system. In a preemptive architecture such as ours, we see that the access delay could be zero, or instantaneous access. The only factor in access delay would be due to queueing of requests, and work delays for scheduling and handling by the shared processor(s) in the switch. If we can guarantee that the size of queueing requests will not go above a certain point then we can guarantee an access delay bound. This then guarantees a delay bound of the different services of the network.

The switch architecture proposed tries to minimize SPP in two ways. It first uses preemption to minimize RPP. Then it shares the SPP among all the ports. This makes equation 2.2 valid. The sharing is done by queueing the requests to the processor complex. Thus we can take advantage of the fact that RPP varies over time and the summation across several ports will be less than giving each port a processor that can handle the maximum RPP.

The algorithm specified also tries to send NDUs that are as large as possible. Only when a higher priority ADU arrives that must be sent down the same link will an NDU be preempted. This fragments the preempted ADU into two or more NDUs. Given that NDUs will be larger than either cells or packets the required processing power by the system should be at or below the level of the other two architectures and thus SPP is lower as well. When new ports are added to a switch, the increase in switch processing power will be linear. This is because the size of input NDUs do not change with port increases. We show an example of the switch processing power vs. port size with a preemptive system in Figure 3-3.

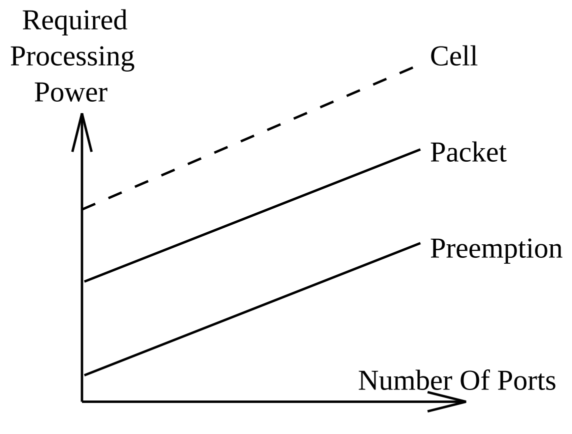

Figure 3-3: Comparison of Required Processing Power of Preemptive System

When we increase link bandwidth, the effects are much more startling. As explained above, access latency plays a part in the delay of an ADU being transmitted. If access latency decreases, then the work delay decreases. As bandwidth increases, the access latency can remain constant and the delay will either remain constant or decrease. If we keep the access latency constant, then an NDU will be larger since more data is pumped through in the constant time. Therefore, as link bandwidth increases, the NDUs increase in size and the required processing power stays constant. This stops occurring when an NDU reaches the size of the ADU it represents. Then more processing power is required at a linear growth rate which is changing as more NDUs become the size of ADUs. It is important to note that the required processing power will never exceed the packet requirement. This is shown in Figure 3-4. The amount of bandwidth that can be added before the processing power starts its linear growth is not clear. It is largely dependent on the NDU and ADU sizes which are in turn dependent on the traffic requirements and characteristics. The expectation is that in a well designed system, the amount of bandwidth that can be added should be at least an order of magnitude. If we still wish for static processing power requirements after the NDU size has reached the size of the ADU then it may be necessary to change the scheme so that an NDU contains more than one ADU. This however has many implications which are beyond the scope of this work.

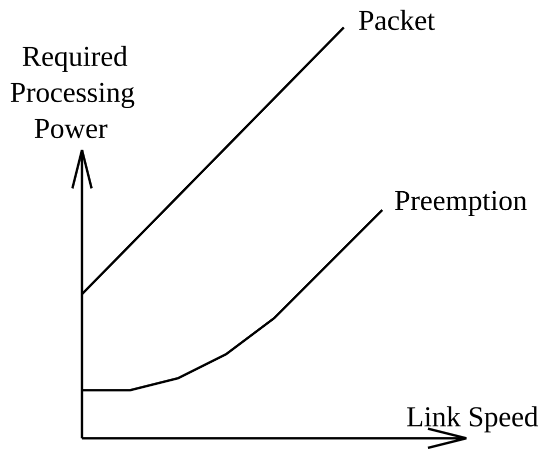

Figure 3-4: Comparison of Required Processing Power of Preemptive System

We expect to see a delay histogram for the network that is as good as the cell and packet architectures while reducing the cost of the required processing power in the switch. The objective is to see if we can get order of magnitude difference in switch processing power from lowering the required processing power and sharing the switch processing power among all the ports. We now talk about some problems with the current argument.

3.3.3 The Fragmentation Problem

When the average NDU size is decreased, the number of NDUs needed to represent an ADU increases. This is currently the primary concern in building a preemptive system. One problem is that an ADU could get fragmented many times in being sent through switches. As the NDUs get smaller and smaller, the required processing power of the switch increases since there are more NDUs to process. Queueing of requests for processing could occur causing degradation in the access delay of the switch. If fragmentation is not controlled then the latency can not be bounded and the delay of the system will be high, and thus unacceptable. Also, the amount of bandwidth used for the data transfer decreases since a header is needed for each NDU. It is believed however that the data transfer bandwidth is not greatly affected since the header required on each NDU is small.

Can we characterize the amount of fragmentation that will occur in a switch? How will an increase in fragmentation at the input of switch affect the fragmentation at the output of the switch. Is there some way to bound how bad the fragmentation gets so that the output of the switch is guaranteed to have fragments of only a certain size or larger? Does the switch need to be designed for the worst case processing requirements or can it be architected such that as the input fragmentation gets worse, the output fragmentation makes up for it? Can we characterize this input fragmentation to output fragmentation? These are some of the questions this report will try to answer about fragmentation.

Chapter 4

Simulation Experiments

This chapter presents two separate results from the simulation experiments performed. The first is a comparison of the RPP of a preemptive switch vs. a cell based switch. Three topologies were run for comparison of the two architectures. The second is an interesting result found while simulating. The result is the effect the ADU priority can have on the RPP. Both results are significant in presenting the viability of an implementation of the architecture presented in Chapters 2 and 3.

4.1 Modeling

4.1.1 Topology

Our basic topology is shown in 4-1. We note that we have three switches in succession with data being transferred in one direction. Each source contributes an equal amount of traffic. As each source contributes an amount, the link load grows. Therefore, the link load between switch one and two is less than the link load between two and three. Switch three will be loaded the heaviest with respect to ADU processing and link traffic. It is expected that most queueing will occur here. The basic goal of simulations with this topology is to establish a framework to examine the effects of lowering the SPP.

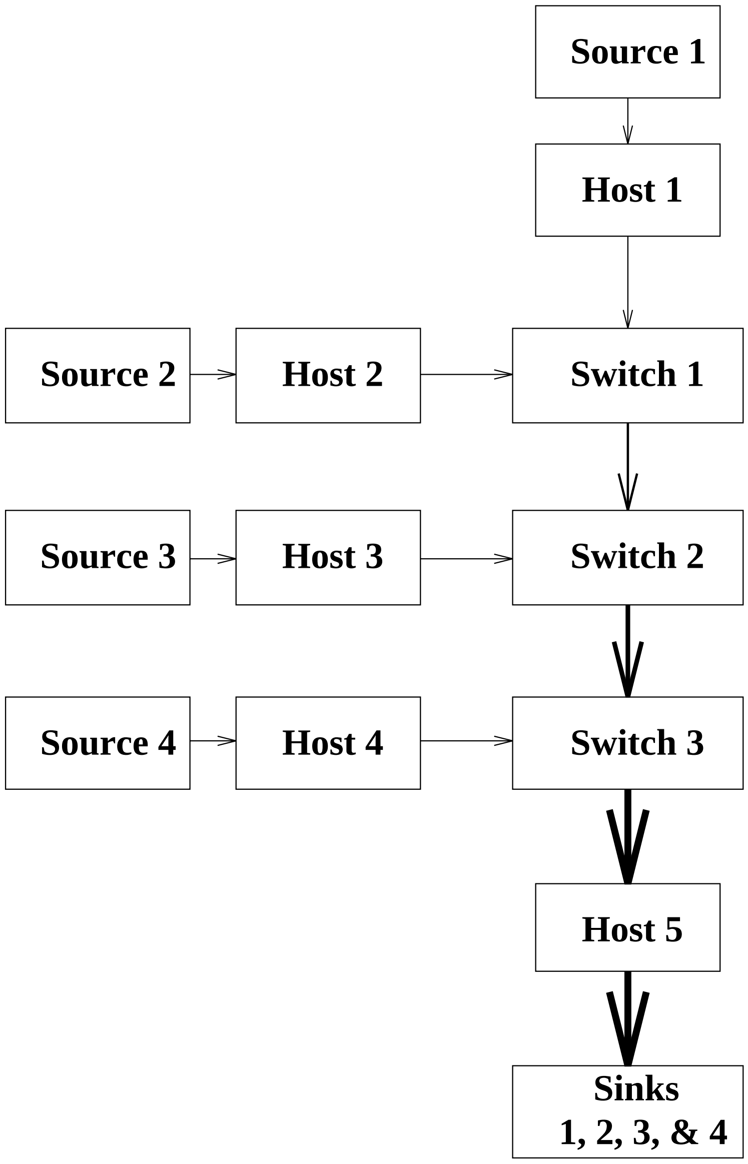

Figure 4-1: Base Topology

From the basic topology we add another set of sources to fully load each link between switches. The extra loading is added and removed on a hop by hop basis. This is shown in 4-2. Switch 2 is the heaviest loaded switch for this topology. It processes six sources worth of traffic. Switches 1 and 3 process four and five sources worth of traffic respectively. We expect to see source 4 have lower delays since it will not be delayed through switch 2. The goal here is to examine how integrating traffic affects the switch's performance based on SPP.

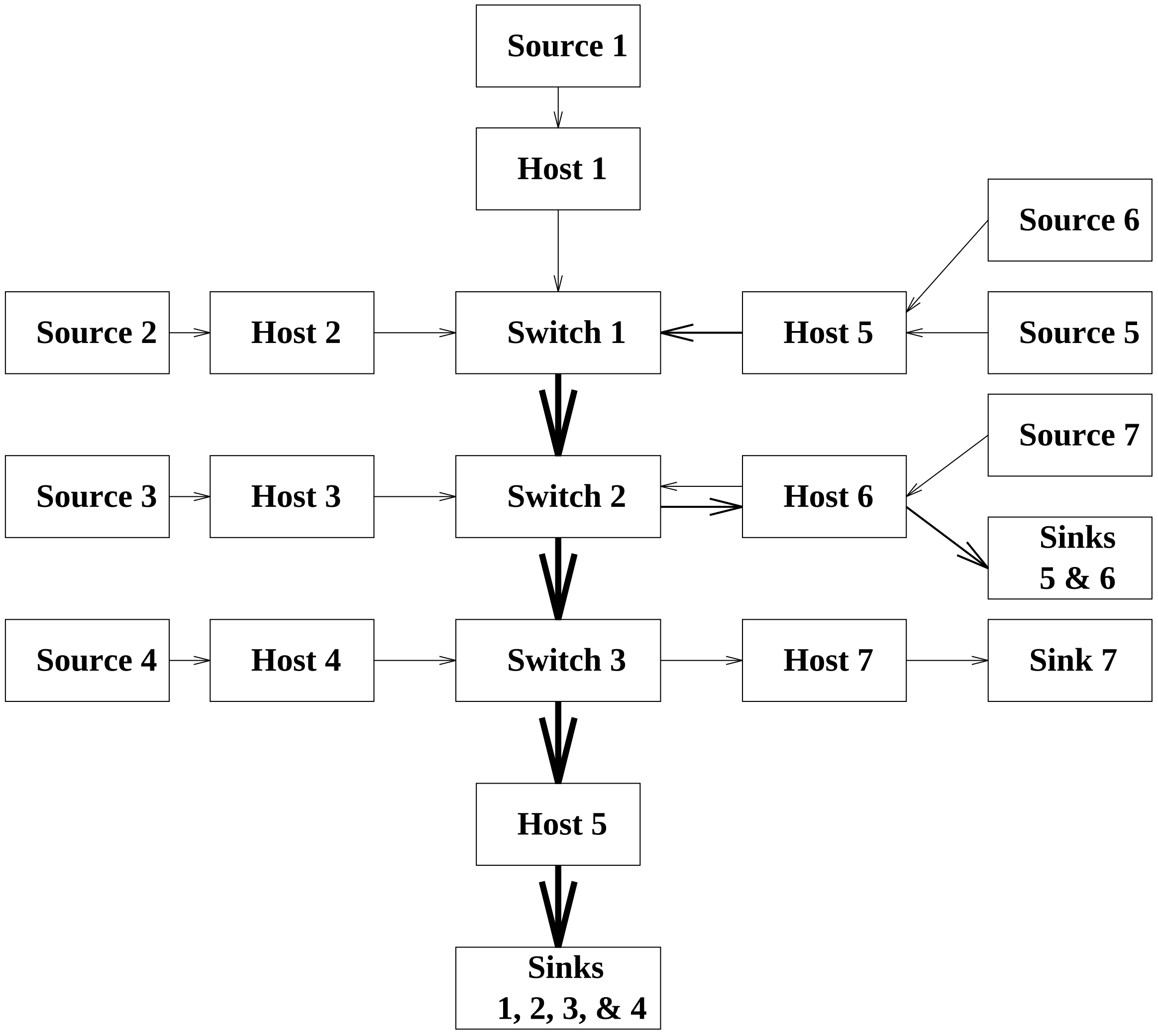

Figure 4-2: Fully Loaded Link Topology

Finally, we examine a topology where we load the SPP heavily and the links moderately. To do this type of loading, we insert heavily loaded links which do not affect the base traffic directly. Figure 4-3 helps to show how this is done. Sources 1, 2, 3, and 4 form the base traffic. The rest of the source-sink pairs form the cross traffic. They affect the base traffic by using SPP and thus loading the switches. Therefore we expect to see degradation in performance even though we are not directly changing the base source traffic.

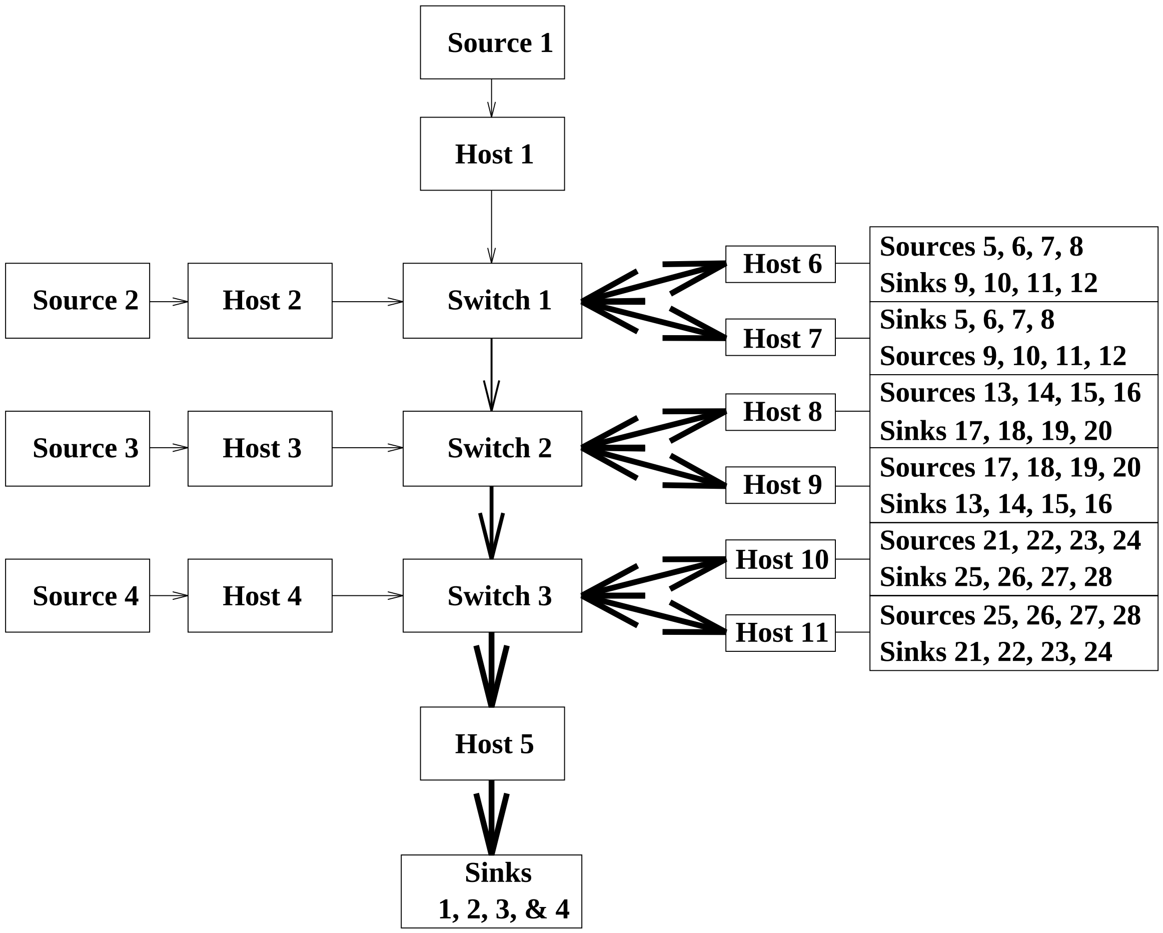
Source 1
Host 1
Source 2
Host 2
Switch 1
Source 3
Host 3
Switch 2
Source 4
Host 4
Switch 3
Host 5
Sinks 1, 2, 3, & 4
Host 6
Host 7
Host 8
Host 9
Host 10
Host 11
Sources 5, 6, 7, 8 Sinks 9, 10, 11, 12
Sinks 5, 6, 7, 8 Sources 9, 10, 11, 12
Sources 13, 14, 15, 16 Sinks 17, 18, 19, 20
Sources 17, 18, 19, 20 Sinks 13, 14, 15, 16
Sources 21, 22, 23, 24 Sinks 25, 26, 27, 28
Sources 25, 26, 27, 28 Sinks 21, 22, 23, 24

Figure 4-3: Fully Loaded SPP Topology

4.1.2 Source Traffic Mix

Each source in the topology figures actually represent a set of sources. For the simulations run in this report two sets of source traffic were chosen. The first used simple poisson sources to generate traffic. The second used the TCP bulk and VBR Video sources. Each set of source traffic is presented separately.

Poisson Sources

We break the poisson sources into three classes. The first class is large ADUs with high priority which is called the "A" traffic. These ADUs represent the real-time traffic for the network. The second class is large ADUs with low priority which is called the "B" traffic. These represent the bulk transfer ADUs which exist within the network. They also can represent real-time traffic being sent at lower priority. A user may want to do this to reduce the cost of using the network while still getting some type of service from the network. Finally, the third class are small ADUs which is called the "Ack" traffic. These represent the acknowledgments and any small data users send back and forth. In table 4.1 important characteristics of each type of traffic is shown. Note that a 50 byte header is put on the front of each ADU for network use. This is a guess as to the actual header size needed. Each source contains one-fourth of the each traffic class listed below (a, b, and ack).

Name	Priority	Data Size	ADU Size	ADUs/sec
A	1	26000	26050	313
B	10	26000	26050	313
Ack	10	50	100	4845

Table 4.1: Total Poisson Source Traffic

TCP Bulk and VBR Video Sources

In this source model, the traffic is broken into two separate classes. The TCP bulk sources are given a low priority. This is since the bulk transfers are generally not real-time traffic in networks. TCP sources also generate acknowledgment ADUs to verify data sent to the sink. The acknowledgment ADUs flow from the sinks to the sources. The priority of the acknowledgment ADUs is low. The VBR video sources are clearly a real-time traffic source. These sources received high priority in the network.

The TCP sources use a simple TCP model containing slow-start [6]. The ADU size is 500 bytes. Ideally, the number of acknowledgment ADUs sent would be one acknowledgment per round trip time. However, in this model an almost one to one mapping occurred. Assuming that TCP sends at 50% utilization of a 155 Mbps link, about 19000 TCP ADUs and slightly less than 19000 acknowledgment ADUs are sent. These approximations were seen in the simulations run. Note that the acknowledgment ADUs are processed by the same processor complex even though they are being sent down a different link.

To generate the VBR video source traffic, [4] was used. The coding is of the movie "Star Wars" and was nicknamed as such. A set of data files representing the ADU size for each frame in the picture were read and then sent through the network. It was found through experimentation that 13 VBR sources generated approximately 40% utilization of a 155 Mbps link. These 13 sources were split equally across sources 1, 2, 3, and 4, with source 1 having four VBR sources instead of three. ADUs were generated by each source at a rate of 24 per second. Table 4.2 shows a summary of the total source traffic.

Name	Priority	Approx. Data Size	ADU Size	Approx. ADUs/sec
Star	1	26000	26050	312
TCP	10	500	550	19000
Ack.	10	50	100	19000

Table 4.2: Total TCP & VBR Source Traffic

4.1.3 Switch and Host Architecture Modeling

The switch architecture presented in 3.2 was modeled as a switch with a single processor in the processor complex. The parameters given to the switch were the link bandwidth, propagation delay, handling delay, and scheduling delay. For all simulations we set the link bandwidths to be 155 Mbps. Propagation delay was set at 100 microseconds for all simulations. Handling and scheduling delays were varied and examined as part of the switch loading.

The primary objective in the simulations run below is to see how the performance degrades based on lower SPP. As equation 2.2 shows, we want to keep the SPP above the sum of RPPs for the links in the switch. Since we are using a preemptive architecture we assume that no work transfer delay exists. From 2.4, we know that the RPP is equal to the number of processing instructions over the work delay. We assume that it takes around 400 instructions to do the processing necessary. Therefore, the following equation represents the SPP of the switch:

$$SPP = \frac{processing\ instructions}{work\ delay} = \frac{400}{handling\ delay + scheduling\ delay} \tag{4.1}$$

As the equation shows, larger handling and scheduling delays will result in lower SPP. We will show our results in terms of the sum of the handling and scheduling delay. This means that as we have increased the handling and scheduling delay we decrease the SPP and performance of the switch will degrade. This is examined further in the following sections.

The host model follows the architecture presented in 3.1. Parameters are link bandwidth, propagation delay, handling delay, and scheduling delay. They match the switch parameters for each simulation run. Since the host model was not our primary interest in research, we do not explore its properties unless they had some direct effect on the switch results presented below.

4.1.4 Port Size and Link Speed Comparison

To examine the simulations and get an idea of how useful the switch architecture is we must have some comparison benchmark. We use the cell architecture as a basis for comparison. Given the cell architecture has a 53 byte cell length, 400 instructions to process a cell, and a 155 Mbps link, we expect the switch processing power to be:

$$RPP_{cell} = \frac{Bandwidth}{Bits\ per\ Cell}\frac{Instructions}{Cell} = \frac{155{,}000{,}000}{53(8)}400 = 146\ MIPS/Port$$

In the preemptive simulations, we calculate the RPP by taking the number of instructions and dividing it by the summation of the handling and scheduling times and then dividing by the number of loaded links the switch uses. We assume that the number of instructions to process is around 400 and that the handling and scheduling times are equal. Therefore, the equation simplifies to:

$$RPP_{preempt} = \frac{\frac{Instructions}{Handling\ Time + Scheduling\ Time}}{n} = \frac{\frac{400}{2(Processing\ Time)}}{n} = \frac{\frac{200}{Processing\ Time}}{n},$$

where Processing Time is the handling or scheduling time, which are equal, and n is the number of loaded ports in the simulation.

The objective is to minimize $RPP_{preempt}$. To minimize $RPP_{preempt}$ we find the largest processing time that still gives us adequate performance for end-to-end delay in a simulation. We know the number of loaded links since it is fixed for a given simulation. Thus $RPP_{preempt}$ is easily found and can be compared to RPP_{cell}. An example: assume that the end-to-end delay does not degrade in a simulation until the processing time is 40 microseconds. The heaviest loaded switch contains 1 fully loaded link. Thus the $RPP_{preempt}$ is:

$$RPP_{preempt} = \frac{200\, instructions}{(40\, uS)(1\, ports)} = 5\ MIPS/Port$$

Continuing the example, in comparison to RPP_{cell}, we see that $RPP_{preempt}$ is roughly 29 times smaller than RPP_{cell}. Thus for every port supported in the cell architecture, 29 ports could be supported in the preemptive architecture.

This forms a nice comparison for the port size and shows the scalability of the preemptive architecture with respect to the number of ports. Is the scalability the same for the link speed? While different link speeds were not examined in the simulations, an argument can be made that a conservative estimate can be made based on port size. If we know the number of ports than can be supported with a given processing power, then we also know the total bandwidth that is supported since the total bandwidth can be broken up among the ports in any uniform or non-uniform manner. Both uniform and non-uniform divisions of bandwidth can be made since the processor complex is shared among ports. In other words, the link speed of each port does not have to be equal.

4.1.5 Graphing Terminology

For the presentation of the results of this report graphing was used. All graphs are plots in time vs. time units. There are three types of graphs that are shown. They are:

end-to-end delay Graph of the end-to-end delay of one or more source-sink pairs which compare the end-to-end delay to the processing time of each switch in the simulation.

access delay Graph of the access delay of each switch with respect to the processing time of the switch.

send-receive size Graph which presents the NDU size with respect to the processing time of the switch.

In this case access delay is the time it takes to process an ADU for transmission. If any queueing occurs before the processor complex then that is included in the access delay. While NDU size is usually not thought of in seconds or microseconds, for these simulations, simple conversions can be made by remembering that a 155 Mbps link supplies 155 bytes of data in 8 microseconds.

Each plot in a graph has a label to associate with either a particular type of traffic or switch feature. The following table can be used as a reference to the labels' meaning. Note that the x means that it represents a number which corresponds either to a source-sink pair or a switch.

x**a** The source-sink pairs for high priority large ADUs better known as "A" traffic. Used in end-to-end delay graphs.

x**b** The source-sink pairs for low priority large ADUs better known as "B" traffic. Used in end-to-end delay graphs.

ack$x$ The source-sink pairs for low priority small ADUs better known as "Ack" traffic. Used in end-to-end delay graphs.

star$x$ The source-sink pairs for high priority "Star Wars" traffic. Used in end-to-end delay graphs.

acc-x The access delay of switch x. Used in access delay graphs.

rec-x The NDU receive size for switch x. Used in send-receive size graphs.

sent-x The NDU send size for switch x. Used in send-receive size graphs.

A connector("-") is used when two sets of data points are combined. This is done for the source-sink pairs 1 and 2 since they both go three hops to reach the sink.

4.2 Poisson Comparison Simulations

4.2.1 Poisson Base Topology

End-to-End Delay

Figure 4-4 is the average end-to-end delay of all source-sink pairs for the base simulation. As can be seen, the A traffic takes initially around 2.2 milliseconds to reach its sink. Of the time taken, 1.6 milliseconds can be attributed to transfer delay, and another 0.2 milliseconds to propagation delay. Therefore, between handling, scheduling, and queueing delays, another 0.4 milliseconds are used. Initially, the time is used by the queueing delay. We note that as we increase the processing time, or decrease the SPP, the end-to-end delay does not degrade in the average case until 68 microseconds. After the 68 microsecond mark, we see a significant decrease in the end-to-end performance. However, this is not the whole story.

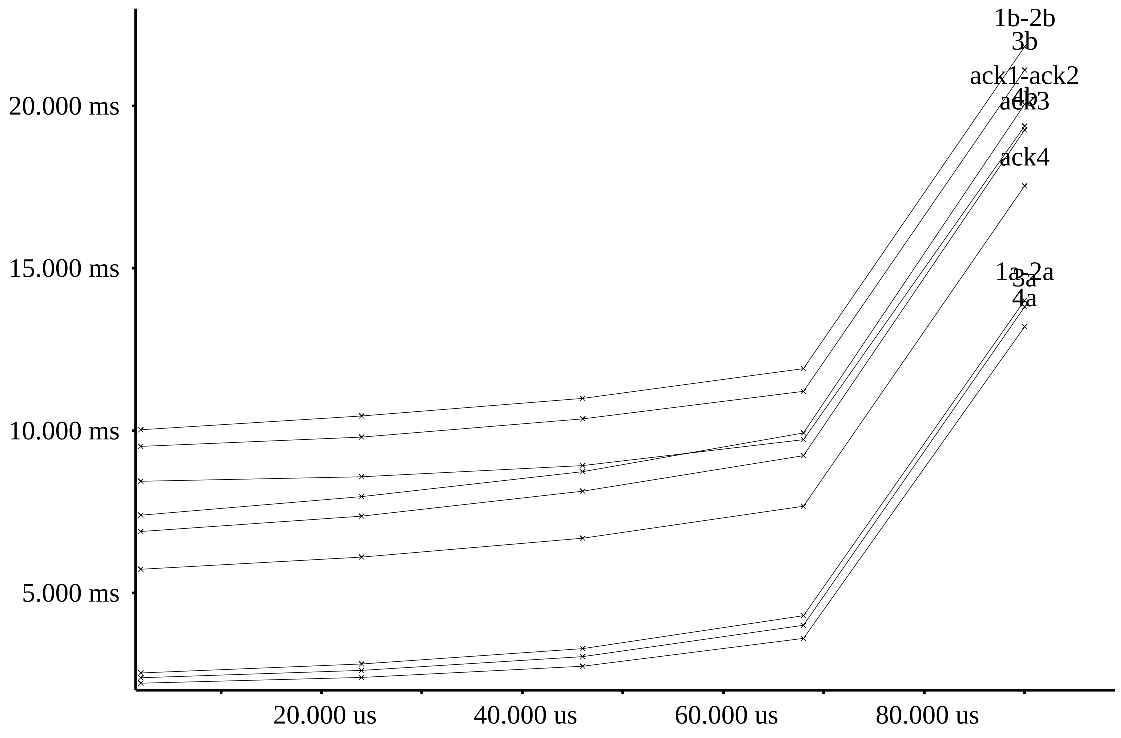

Figure 4-4: Average End-to-End Delay of All Traffic for Poisson Base Topology

While the average does not degrade until we increase the processing time to 68 microseconds, another characteristic must be investigated. Since we are trying to guarantee service for the high priority traffic, we must be able to bound the delay for almost all the high priority traffic. To do this, we examine the 99.9 percentile of the end-to-end delays. This gives us a rough bound on most traffic. Figure 4-5 shows a blow-up of just the A traffic for the simulation run. Just examining the A traffic for the first four points, we see that the 99.9 percentile increases dramatically after the 46 microsecond processing time. This shows that having the processing time at 68 microseconds would be more than likely unacceptable. A better estimate is 46 microseconds.

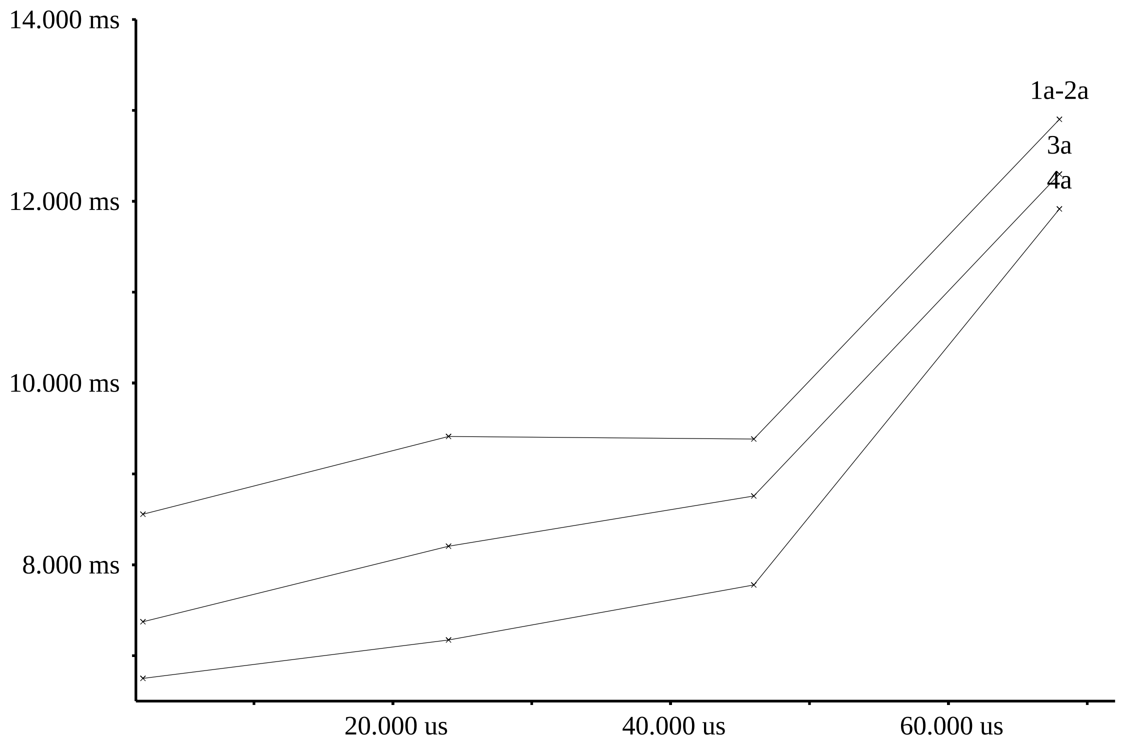

Figure 4-5: 99.9 Percentile End-to-End Delay of "A" Traffic for Poisson Base Topology

The cause of the dramatic increase in the end-to-end delay is due to the processing time increasing to overload switch 3 with ADUs to process. For this to be true, queueing must occur before the processor complex. We can measure this queueing by measuring the access delay. Figure 4-6 shows the average access delay for the base topology simulation. As can be seen, switch 3 starts to be loaded when we are at 68 microseconds processing time. This again is the average loading and Figure 4-7 shows the 99.9 percentile where the loading happens earlier at 46 microseconds as expected.

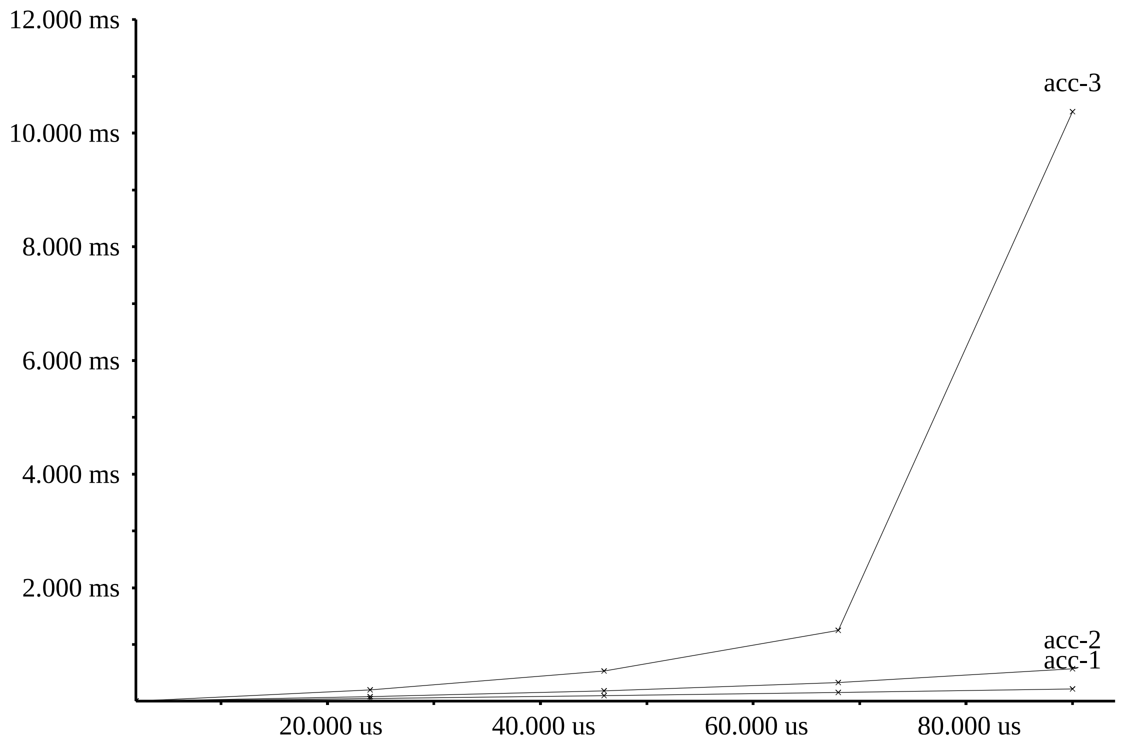

Figure 4-6: Average Access Delay of Switches for Poisson Base Topology

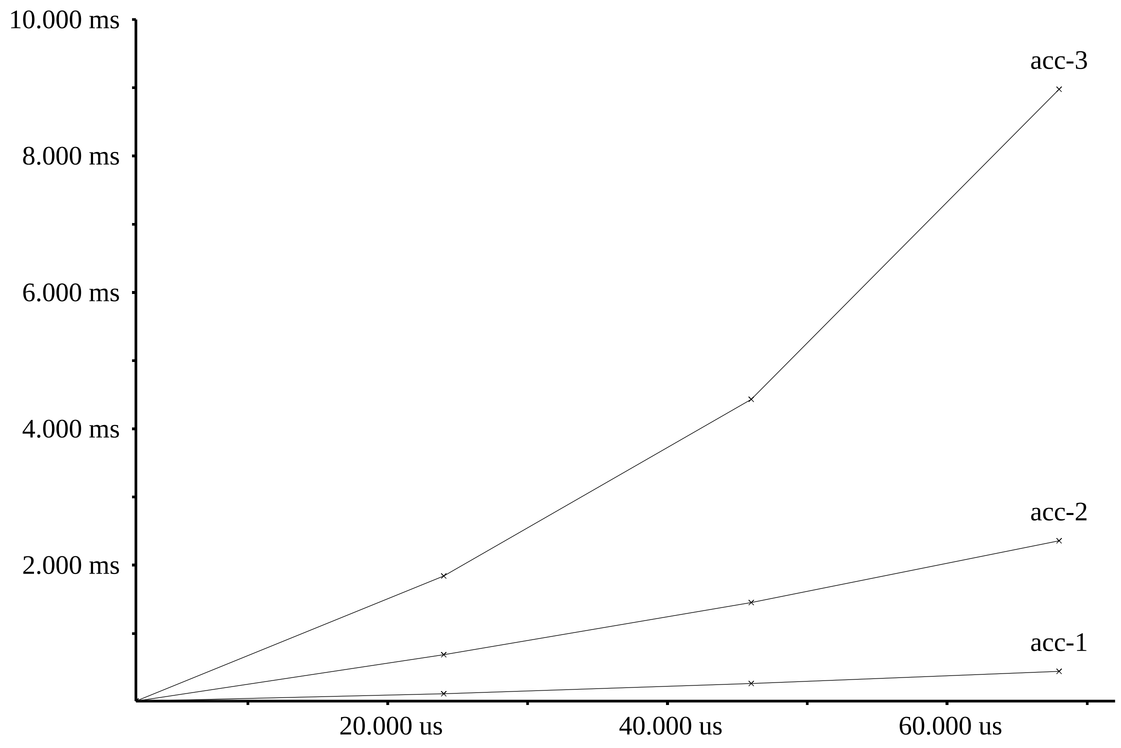

Figure 4-7: 99.9 Percentile Access Delay of Switches for Poisson Base Topology

A processing time of 46 microseconds seems acceptable based on the first set of simulations. Only one link was fully loaded on switch 3. Therefore, we calculate the SPP for the preemptive switch to be:

$$RPP_{preempt} = \frac{\frac{200}{46}}{1} = 4.3MIPS/port$$

When comparing it to the RPP_{cell} found above, we see that $RPP_{preempt}$ is over 30 times smaller than RPP_{cell}. Therefore, at least 30 ports could be supported with the same processing power using the preemptive architecture that the cell architecture uses for one port.

NDU size

Examination of fragmentation of the ADUs into NDUs from input to output is done via the send-receive size graphs. For the base topology simulation the results are shown in Figure 4-8. Upon first examination the send and receive sizes appear to fluctuate heavily based on the processing time. However, upon closer examination of the y-axis, we see that the fluctuation is at most 6 microseconds. Even at 8 microseconds fluctuation, we are only varying the size by 155 bytes. The size of each NDU in the worst case is upwards of 2,900 bytes. This is roughly 55 times the size of a cell. Thus for every NDU sent, an equivalent 55 cells would be sent in rapid succession.

It is also interesting to note here that the sent size is only slightly smaller than the received size at each switch. This is because very few ADUs need to be preempted since the large ADUs have the highest priority and the small ADUs share the lowest priority. Since there are few large ADUs, there are only a few chances for preemption to occur. Thus, the size of the NDUs stays very close to the size of the ADUs.

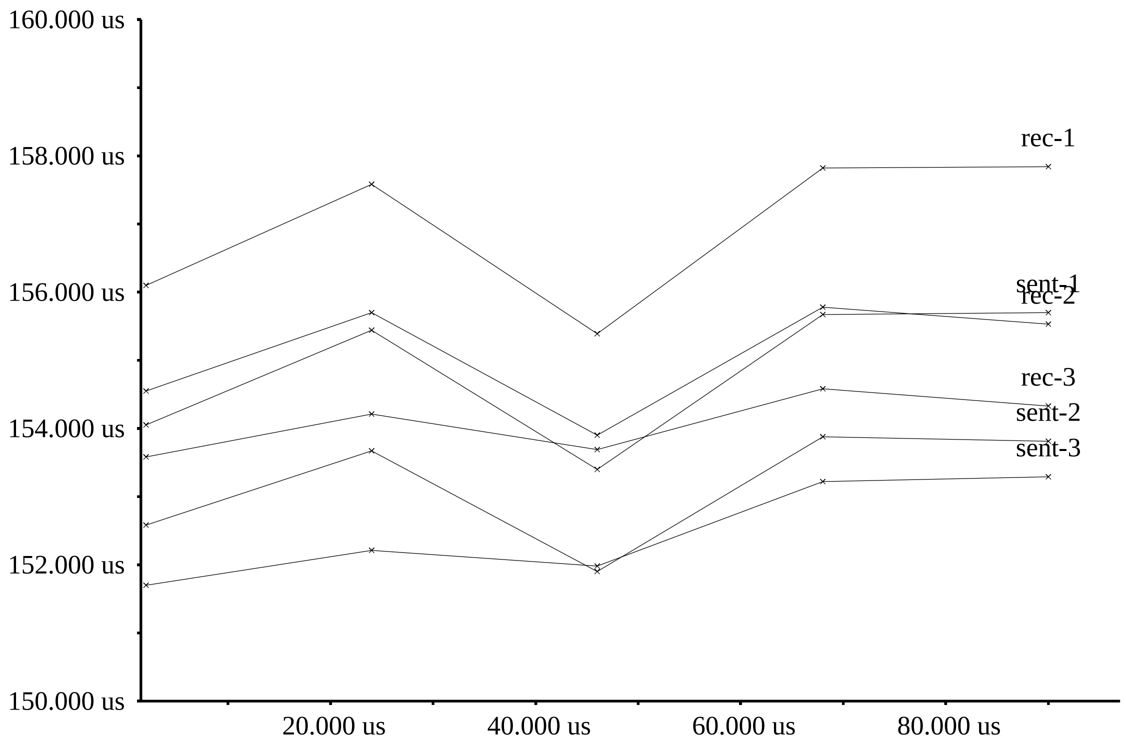

Figure 4-8: Average Send/Receive Size for Poisson Base Topology

4.2.2 Poisson Link Loaded Topology

The basic objective of these simulations is to understand how the average NDU size changes based on a series of loaded links and what the end-to-end delay is when a series of loaded links are traversed by an ADU. While the three switches each have a link that is fully loaded, switch 2 requires the most processing power. This is since we only add one hop traffic to the base topology. Switch 2 must handle 1.5 fully loaded links. Both of the other switches handle less. Therefore, we expect to see the end-to-end delay of source-sink pair number 4 be lower than the other end-to-end delays. This is because the source-sink pair never passes through switch 2 in sending data. Further data below will support this claim.

End to End Delay

We see from Figure 4-9 that the average end-to-end delay does not start to degrade heavily until after the processing time is above 41 microseconds. The results of the 99.9 percentile were similar to the base topology in that the actual processing time we should use is slightly smaller than 41 microseconds. Rather, the processing time to not degrade at the 99.9 percentile should be at 28 microseconds. Looking at the figure we note that the end-to-end delay of source-sink pairs of 4 do not degrade at the same rate as the other traffic source-sink pairs. This is in support that switch 2 is the heavily loaded switch. The source-sink pair 4 traffic does not go through switch 2 and thus is not affected by the heavily loaded switch. Further support for this is shown in Figure 4-10 where as processing time increases the average access delay of switch 2 dominates the access delay of the other two switches.

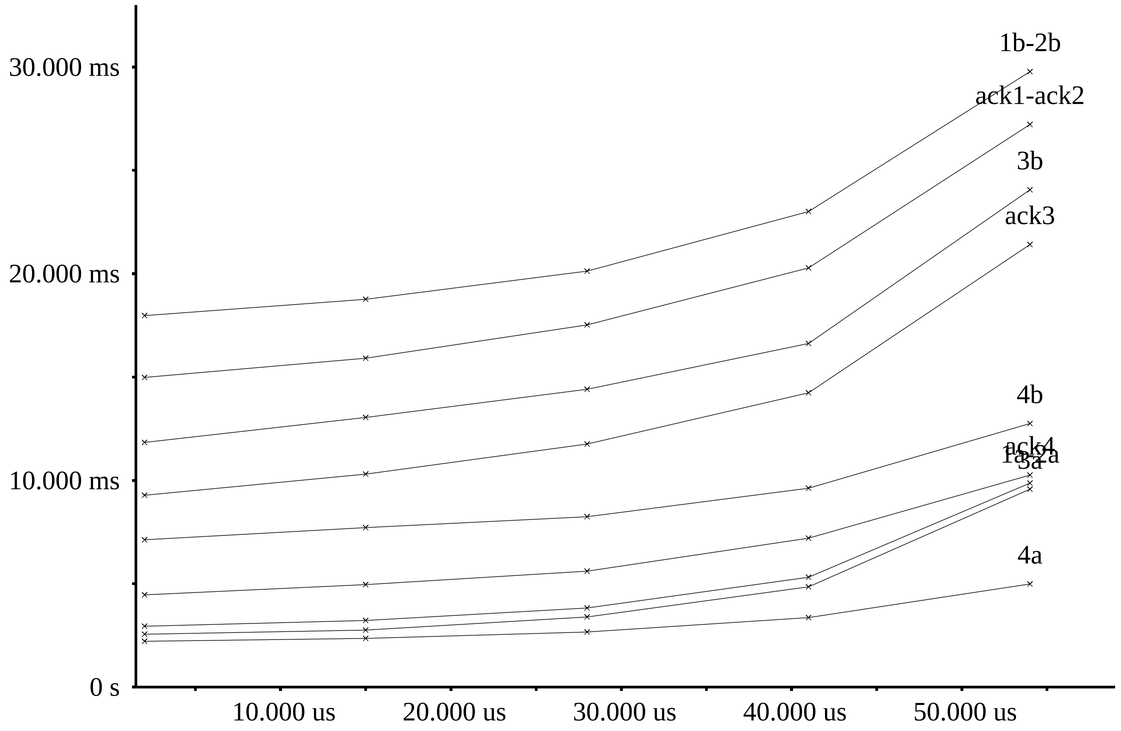

Figure 4-9: Average End-to-End Delay for Poisson Link Loaded Topology

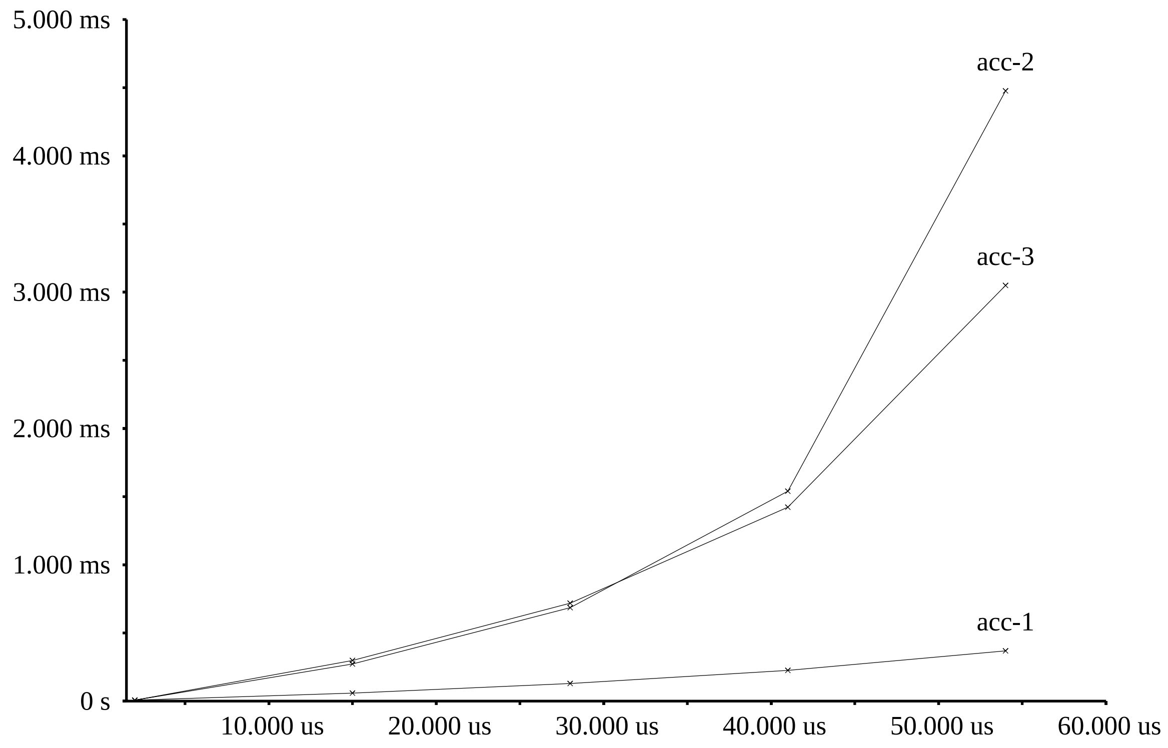

Figure 4-10: Average Access Delay for Poisson Link Loaded Topology

A processing time of 28 microseconds seems acceptable based on the 99.9 percentile readings not shown. Only 1.5 fully loaded links were used in switch 2. Therefore, we calculate the RPP for the preemptive switch to be:

$$RPP_{preempt} = \frac{\frac{200}{28}}{1.5} = 4.8MIPS/port$$

When comparing it to the RPP_{cell} found above, we see that $RPP_{preempt}$ is over 30 times smaller than RPP_{cell}. Therefore, at least 30 ports could be supported with the same processing power using the preemptive architecture that the cell architecture uses for one port.

NDU Size

The send-receive size graph shown in Figure 4-11 shows the variation in NDU size over the three switches. An interesting result is that the size actually does not change all that much from the input of the switch to the output of the switch. The amount of change is at most on the order of 10 microseconds. Even at with a conservative estimate as this, we are talking about only 100 bytes which in comparison to the 2,900 byte average NDU is small. The explanation for the large NDU size is that very few ADUs must be getting preempted. This actually makes sense since the number of preempting ADUs is on the order of a few hundred given the traffic mix table above. Again, in comparison, we need 55 cells to support the average NDU size for transmission in a cell architecture.

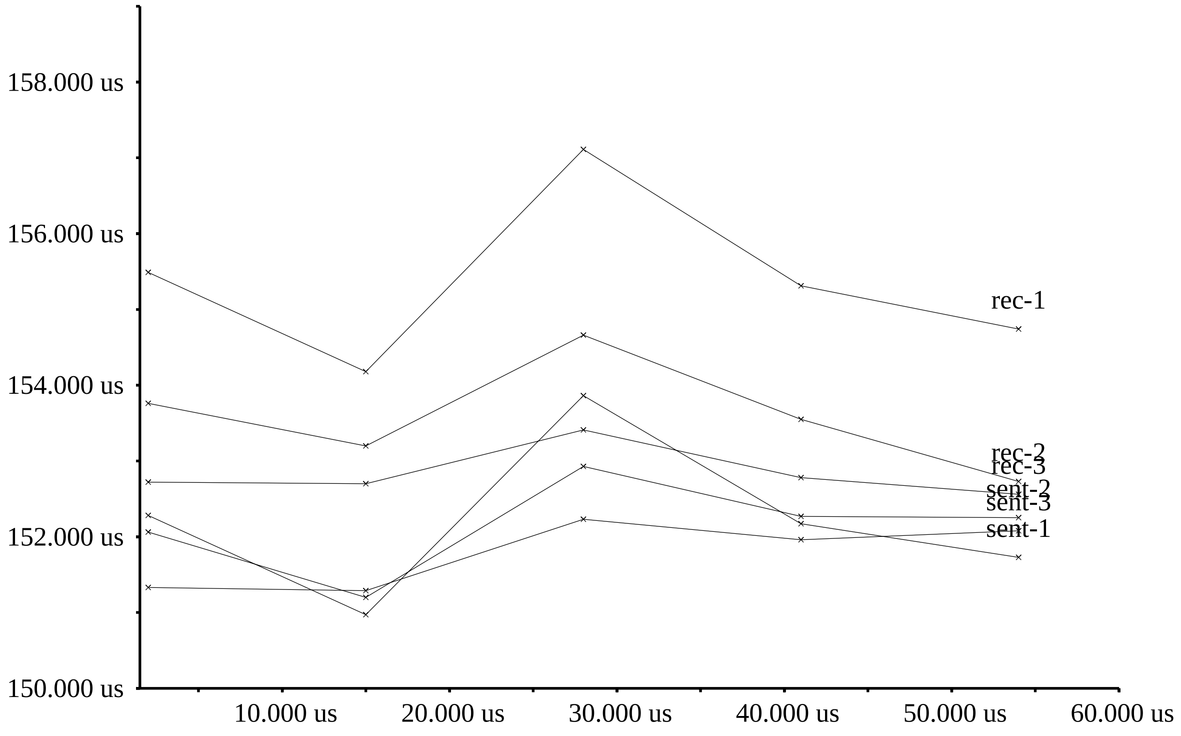

Figure 4-11: Average Send/Receive Size for Poisson Link Loaded Topology

4.2.3 Poisson Processor Loaded Topology

For these set of runs switch 3 is the heaviest loaded switch. It controls three fully loaded links. We want to see how the sharing of the processor complex can affect the scalability of the switch architecture. Thus, we are curious to know if there were any unseen effects in combining many ports together in sharing the processing power. From these results we can get an even better feel for the capability of the switch architecture.

End-to-End Delay

Figure 4-12 shows the end-to-end delays. As can be seen, the delay is roughly around the same as the end-to-end delay of the base case simulations above. The 99.9 percentile graph has the same characteristic as the other 99.9 end-to-end delay graphs. It provides that the degradation actually happens before 23 microseconds. The previous processing time is 16 microseconds. We use 16 microseconds as an acceptable time for conservative estimates of RPP.

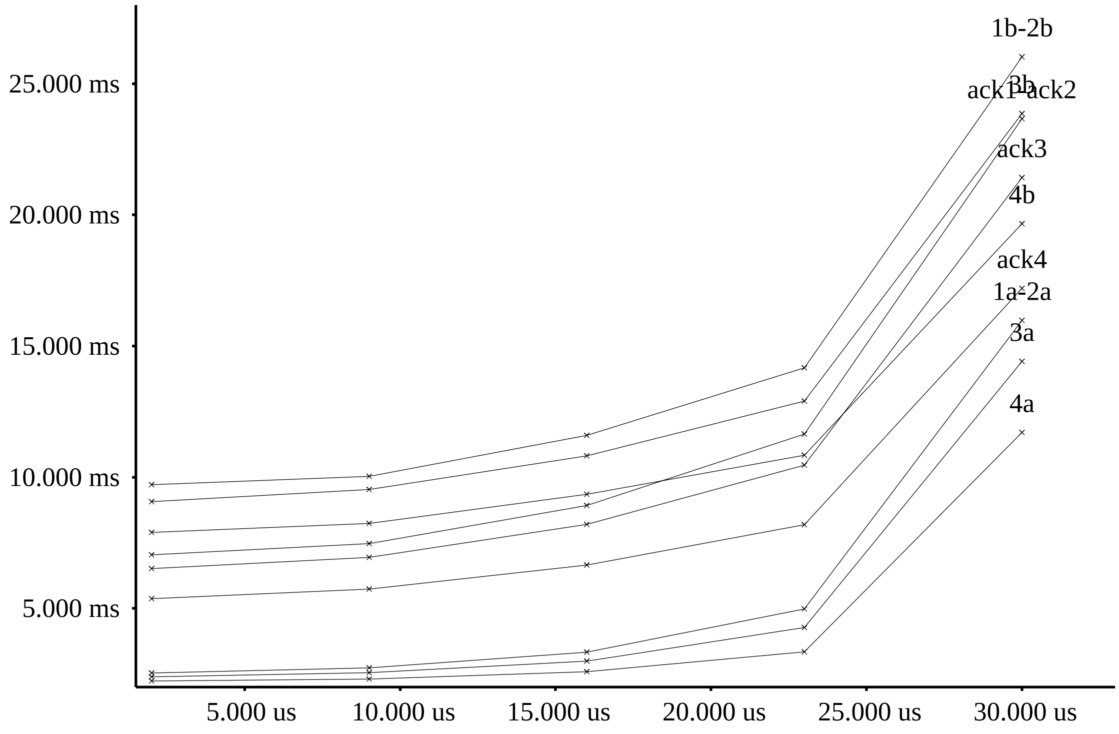

Figure 4-12: Average End-to-End Delay for Poisson Processor Loaded Topology

Using the 16 microseconds processing time and the fact that switch 3 is loaded by three links, we compare the RPP_{cell} and $RPP_{preempt}$. $RPP_{preempt}$ is again found to be over the 30 times smaller than RPP_{cell}. Once again, at least 30 ports could be supported with the same processing power using the preemptive architecture that the cell architecture uses for one port.

NDU Size

The NDU size for these loaded processor topology had the same characteristics as the other two simulations.

4.3 Poisson Priority Simulations

While working on the simulations for the above source traffic an interesting result was found. We start by noting the bandwidth and ADU percentages of each traffic source (Table 4.3). As can be seen, the A and B traffic combined take up a large percentage of the bandwidth. While, on the other hand, the Ack traffic takes up a large ADU percentage. Since the small ADUs take a large percentage of the ADU traffic, it is the primary load on the processing power. The Large ADUs take very little processing power since there are so few of them. We use the term *bimodal* to represent the property that large ADUs dominate one mode of characterization and small ADUs dominate the other mode of characterization.

Name	Priority	Bandwidth %	ADU %
A	1	48.6	5.7
B	10	48.6	5.7
Ack	10	2.8	88.6

Table 4.3: Percentages for Total Poisson Source Traffic

If we set the small ADU priority to be low then the small ADUs will queue behind the higher priority ADU and "compress". Figure 4-13 shows this phenomenon when the small ADU priority is low. By compress we mean to say that the small ADUs will bundle together and blast the link at once. This is similar to the effect seen with TCP and ACK-compression in [12]. At the next switch, the compression causes an overload in the amount of ADUs to be processed in a short interval of time. Thus, the required processing power increases for a brief period of time when the burst of small ADUs arrives.

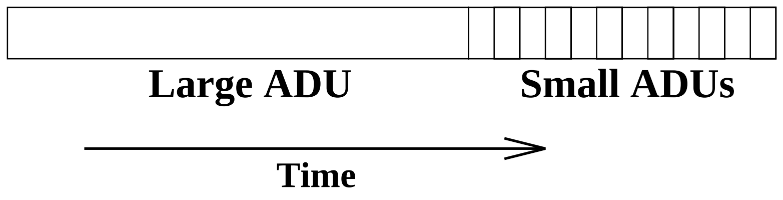

Figure 4-13: Low Priority Small ADU Phenomenon

This burst causes queueing before the processor complex. The queueing increases the delay of all ADUs arriving at the switch during a burst, since all ADUs are processed through the same complex. So, the burstiness of the ADUs arriving is a consideration and problem for the priority system and traffic mix. To alleviate this problem we can change the priority of the small ADUs to be higher than the large ADUs. This will spread the ADUs to process over a longer period of time, thus decreasing the amount of queuing before the processor complex and decreasing the delay seen by each ADU at the switch.

Changing priority of Acks does not only affect the access delay, but it also slightly increases the queueing delay for some types of traffic. Since we are increasing the priority of the Acks, the end-to-end delays for traffic whose priority is lower than the Acks, but was higher before the change in priority of the Acks will increase. For example when Acks were moved from a priority of 10 to 5, the B traffic suffers since it must now queue behind the Ack traffic. This loss in performance is very small since the percentage of bandwidth traffic the Acks use is also very small.

For purposes of support in this theory three priorities were tried. The first was with Acks having highest priority, ie. equal to 0. This means that real-time traffic is preempted by the Ack traffic. The second was having Ack traffic be at a priority between the high and low priorities of the large ADUs, equal to 5. This causes compression behind only the high priority ADUs. Lastly, having the ack priority equal to the low priority ADUs was retrieved from the previous simulations, equal to 10. This represents what it would be if typical TCP traffic were what we are modeling. The expected result is to see the first case have the best performance since the required processing power will be lower.

4.3.1 Added Graph Terminology

There is one new type of graph used to compare the three different levels of priority of the Ack traffic. This graph is called the priority comparison graph. It compares either the end-to-end delay of some particular source-sink pair, or the access delay of a particular switch. Three lines exist in the graph. The line labeled "Low" is the graph line for the simulation where Ack traffic was of priority 10. The line labeled "Mid" is the graph line for the simulation where Ack traffic was of priority 5. The line labeled "High" is the graph line for the simulation where Ack traffic was priority 0. This graph is used to display the advantage of increasing the priority of the Ack traffic.

4.3.2 Poisson Priority Base Topology

End-to-End Delay

Figure 4-14 shows an example of the results gotten from simulation for the three different levels of priority of Ack traffic. In this case we are examining the average end-to-end delay of high priority traffic of the source-sink pairs 1a-2a. As can be seen, this graph supports the theory of overloading. In all cases, the other source-sink pairs had the same characteristic curve set. We note that the crossover doesn't happen until after the switch has be overloaded heavily (68 microseconds). The amount of delay saved in the overloaded case is around 2 milliseconds. However, at that point the delay for almost all ADUs is unacceptable anyway.

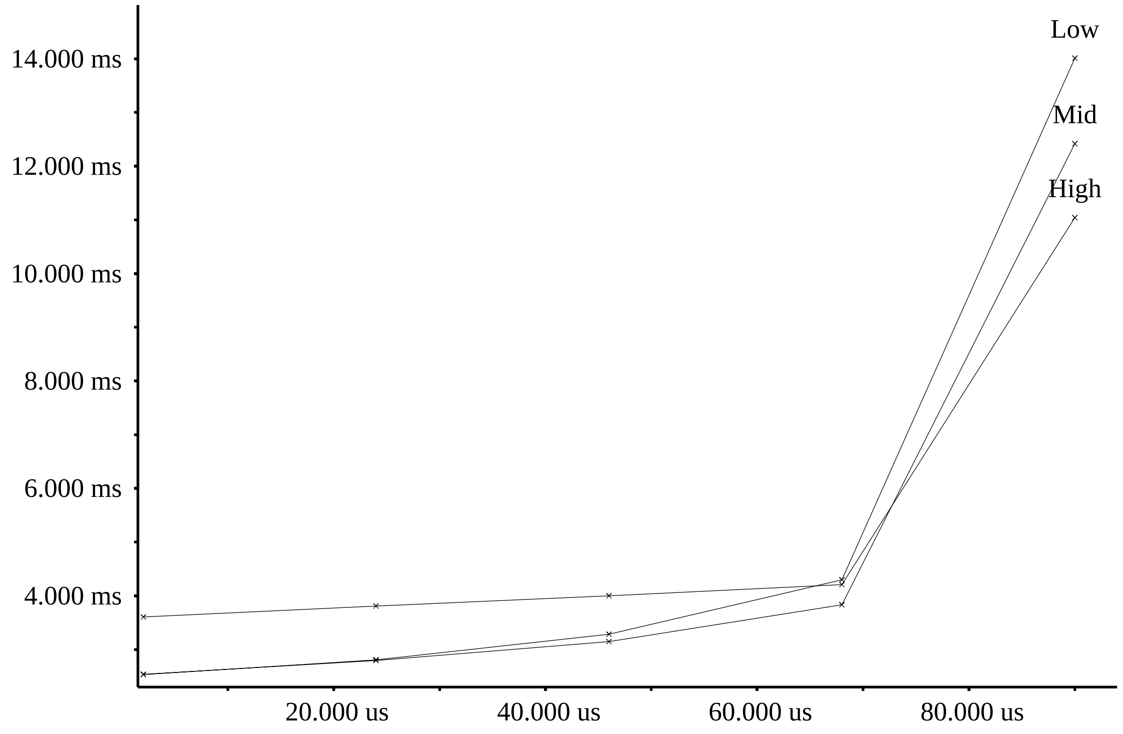

Figure 4-14: Priority Comparison of Average End-to-End Delay (1a-2a) for Poisson Base Topology

If the average curves don't appear as encouraging to support the high priority idea, then perhaps the 99.9 percentile curves will be more encouraging. However, in the base case it was found not to be true. The graph in Figure 4-15 shows the small difference in the end-to-end delays of the three curves. Again all the source-sink pairs followed this form. Both of these graphs support the argument that increasing the Ack traffic priority does help, but that it doesn't help as significantly as was hoped.

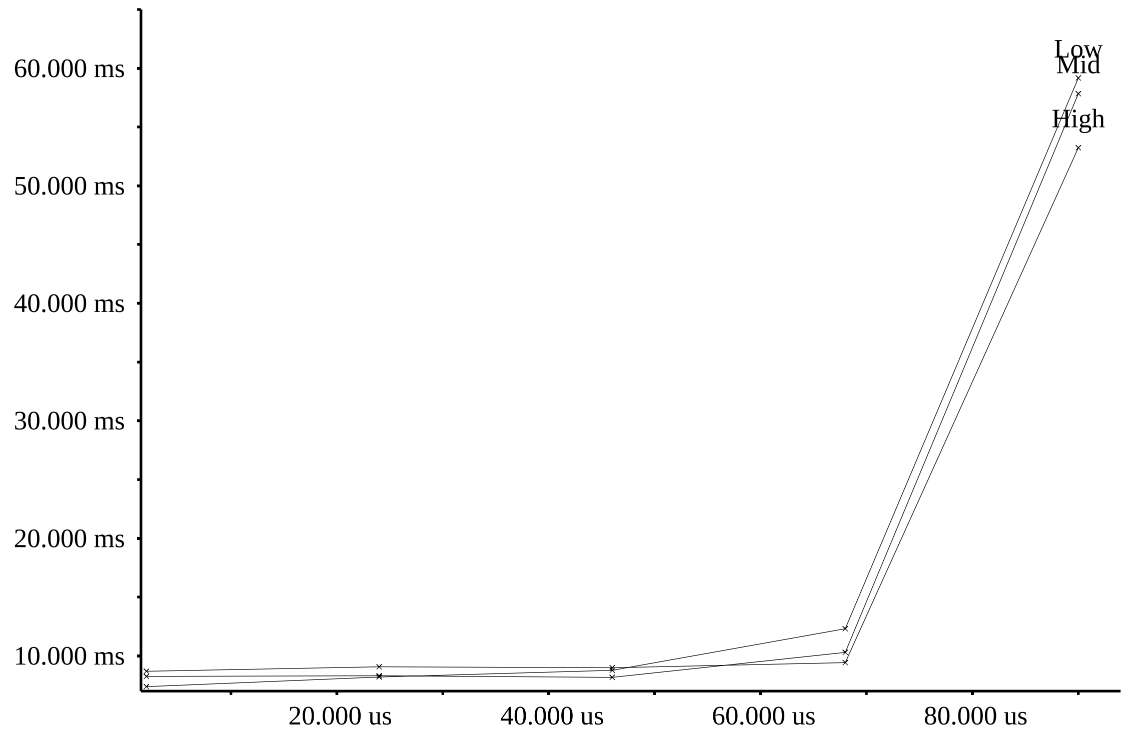

Figure 4-15: Priority Comparison of 99.9 Percentile End-to-End Delay (3a) for Poisson Base Topology

While this is a discouraging result, some care must be taken in interpreting the importance. The property seems to hold and is useful in the extreme case. If we are planning to minimize the SPP, we may find this feature useful when many links are loading the processor complex. We will again do a comparison in the processor loading case. The results there may show that the change in priority helps even more.

NDU Size

Taking the small ADUs and increasing the priority to be between the high and low priorities should have the expected effect of decreasing the size of the NDUs being sent. Since the small ADUs now have a higher priority than the low priority large ADUs, we expect to see the low priority ADUs getting preempted by the small ADUs.

This causes the low priority ADUs to be broken into two or more NDUs which causes the average NDU size to go down. The expected result is to see the receive NDU size be somewhat larger than the send NDU size. This can be seen in Figure 4-16. We lose about 20-30 microseconds in length on the size of the NDU being sent out. We attribute this to the preemption. Still, NDU size is not a concern since the average size out of switch 3 is just over 1900 bytes or 36 cells.

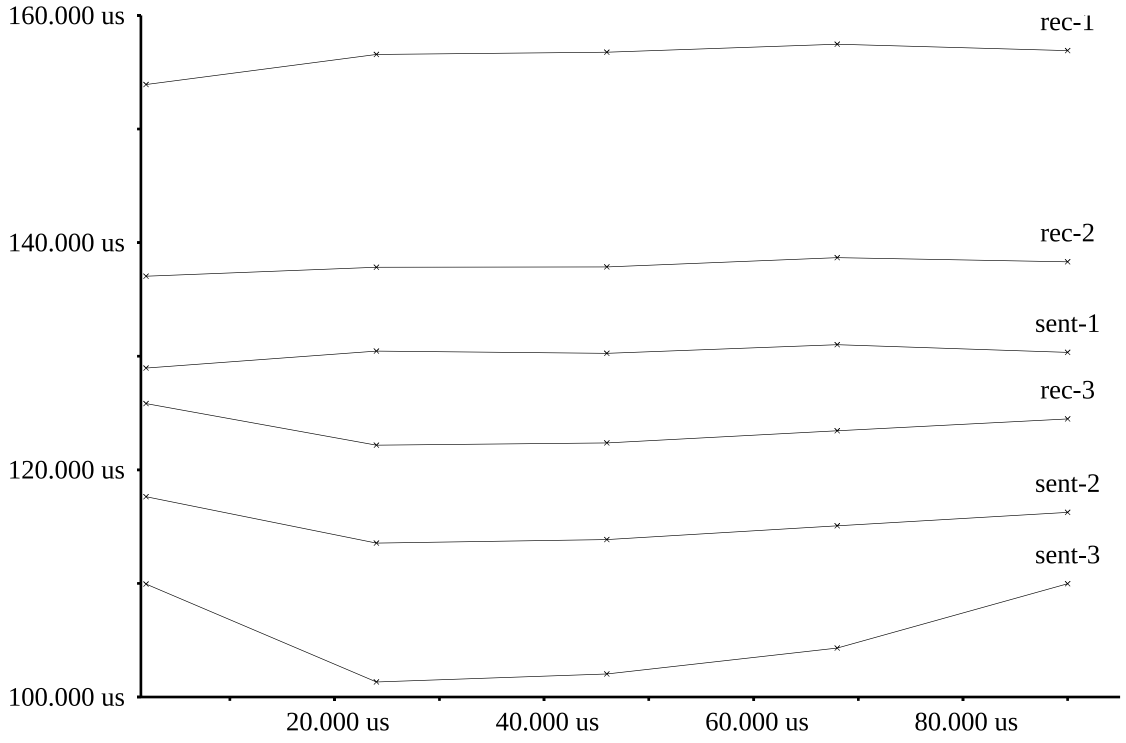

Figure 4-16: Average Send/Receive Size of Mid. Priority Ack Traffic for Poisson Base Topology

If increasing the priority from low to middle decreased the average NDU size, then increasing the priority from middle to high will decrease it again. We show this in Figure 4-17. The average NDU output size for all three switches has dropped at least 20 microseconds from increasing the priority from middle to high. Also, we see a 40 microsecond drop with respect to the input NDU size of switch 1 to its output size. Again, preemption is the primary culprit. Even though the sizes have dropped, the NDU size is still huge in comparison to the cell size. At approximately 80 microseconds in length, the average size of the NDU coming out of switch 3 is 29 cells. Furthermore, little change seems to occur in the sizes based on the processing power of the switch.

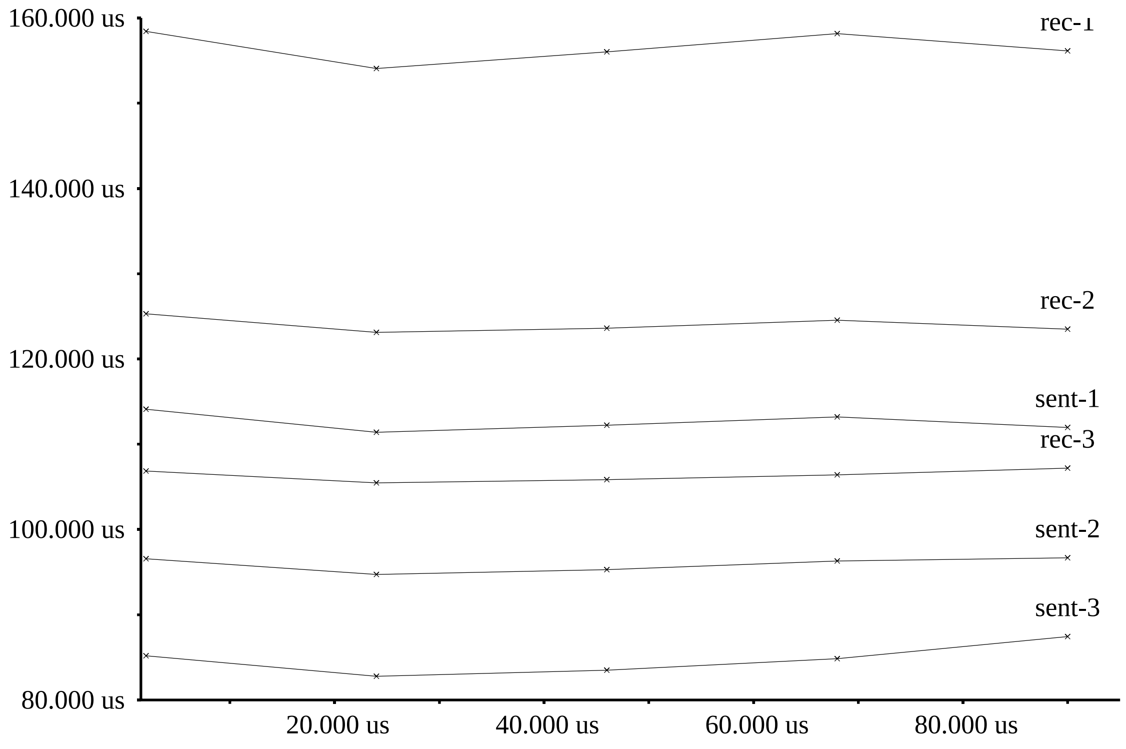

Figure 4-17: Average Send/Receive Size of High Priority Ack Traffic for Poisson Base Topology

As the graphs 4-8, 4-16, and 4-17 show, the average NDU size decreases as the priority of the ack ADUs goes higher. This is because the Ack ADUs preempt more ADUs as their priority is increased. The links between switch 1 and switch 2 and between switch 2 and switch 3 are not completely loaded. Loading those links to capacity, should have a dramatic effect on the average NDU size since more ack ADUs can preempt at each switch. We explore this possibility in the following section.

4.3.3 Poisson Priority Link Loaded Topology

End-to-End Delay

Since switch 2 is handling more than one fully loaded link, an expectation is to see some advantage gained by making the Ack ADUs have higher priority. The results show support for this beyond the original expectation. Figure 4-18 shows, the end-to-end delay with Ack traffic at high priority. Comparing this graph to Figure 4-4 we see that the delays are lower even at the overloaded point 55 microseconds.

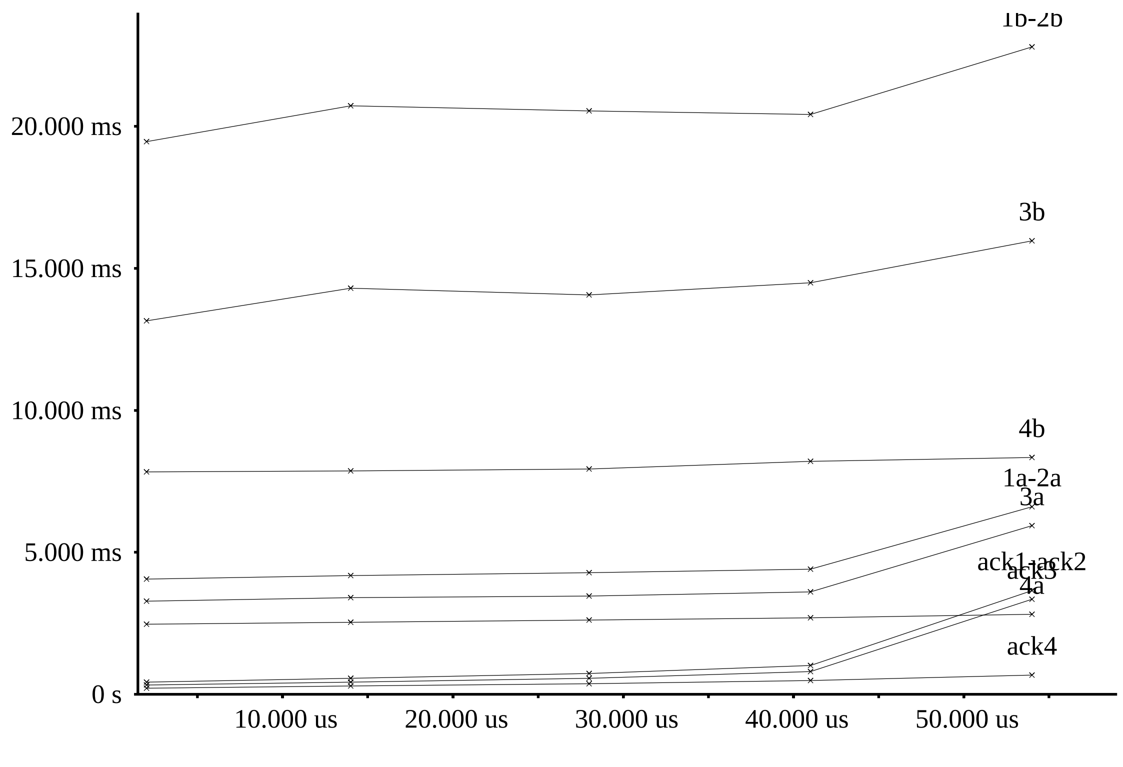

Figure 4-18: Average End-to-End Delay with High Priority Ack Traffic for Poisson Link Loaded Topology

To further support the compression idea, we examine the access delays for switch 2 for the three different Ack priorities. This is shown in Figure 4-19. As can be seen, the high priority simulation has a lower access delay as Processing Time is increased. The improved access delay is caused by switch one sending a mix with the Ack ADUs spread out. Ack compression doesn't exist for the higher priorities since Ack ADUs do not queue behind the large ADUs. This can also be seen by doing end-to-end delay comparisons for each source-sink pair. In all cases, the trend was the same as the trend seen in the base simulations for Figure 4-15. This result strongly suggests that the sharing of the processor complex may become important with relation to priority of ADUs.

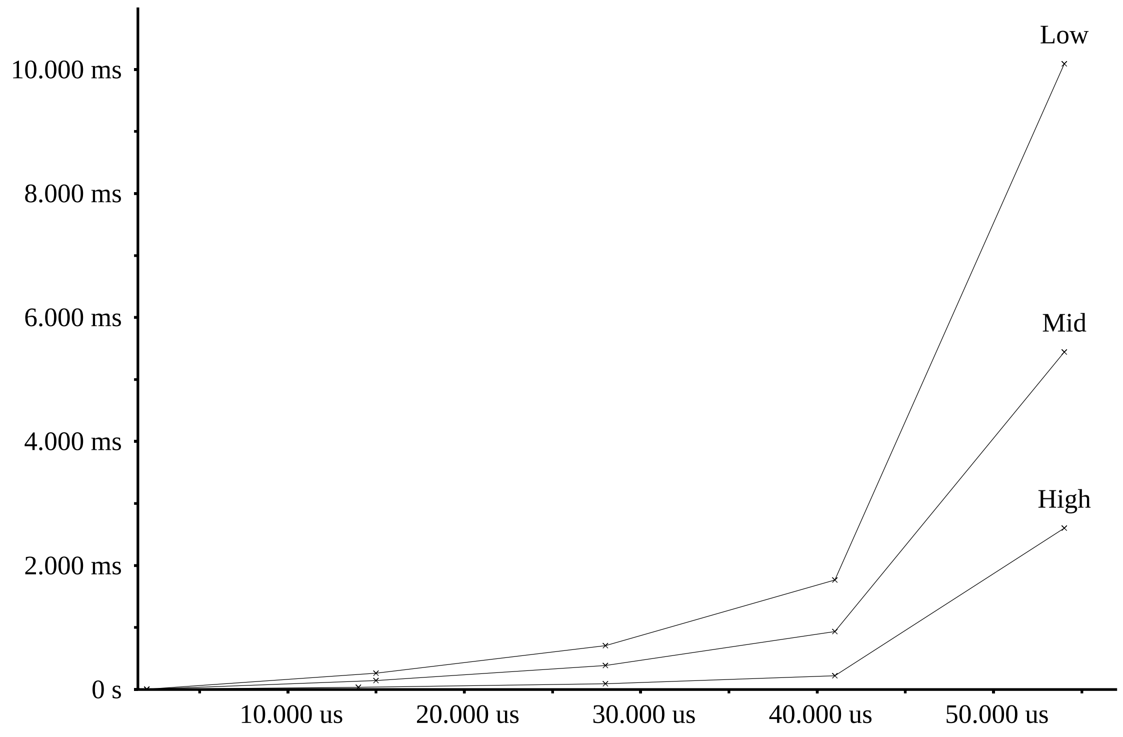

Figure 4-19: Priority Comparison of Average Acc. Delay (Switch 2) for Poisson Link Loaded Topology

NDU Size

It was expected that the NDU size would be significantly smaller for the simulations with heavy link loading. The smallest average size NDU was found when the the ack ADUs had the highest priority. However, as Figure 4-20 shows, the output size seems to be fairly regular and not dependent on the output link. If such a property holds, then the amount of fragmentation an ADU goes through in the network would be more dependent on the other traffic in its path than the actual path length. This also implies that fragmentation could be controlled by controlling the reservation of traffic in the network.

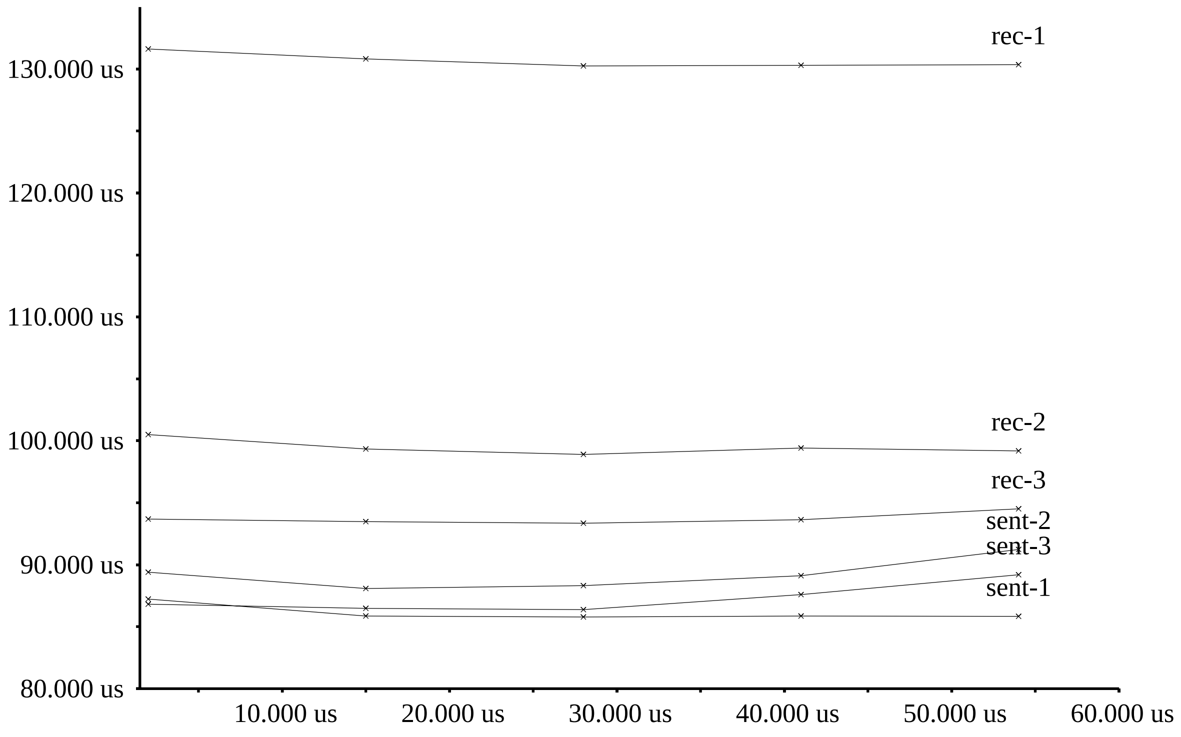

Figure 4-20: Average Send/Receive Size of High Priority Ack Traffic for Poisson Link Loaded Topology

4.3.4 Poisson Priority Processor Loaded Topology

Changing the priority of the acks has an even greater effect with cross traffic than in either of the cases above. Figure 4-21 shows a typical end-to-end curve comparison. All the other curves fit this form. The delay increase is much more dramatic since the compression is three times as bad. As each loaded cross traffic link is added, the effect of priority will be even greater. Thus keeping the ADUs to process spread out will keep the switch from queueing requests at the processor complex.

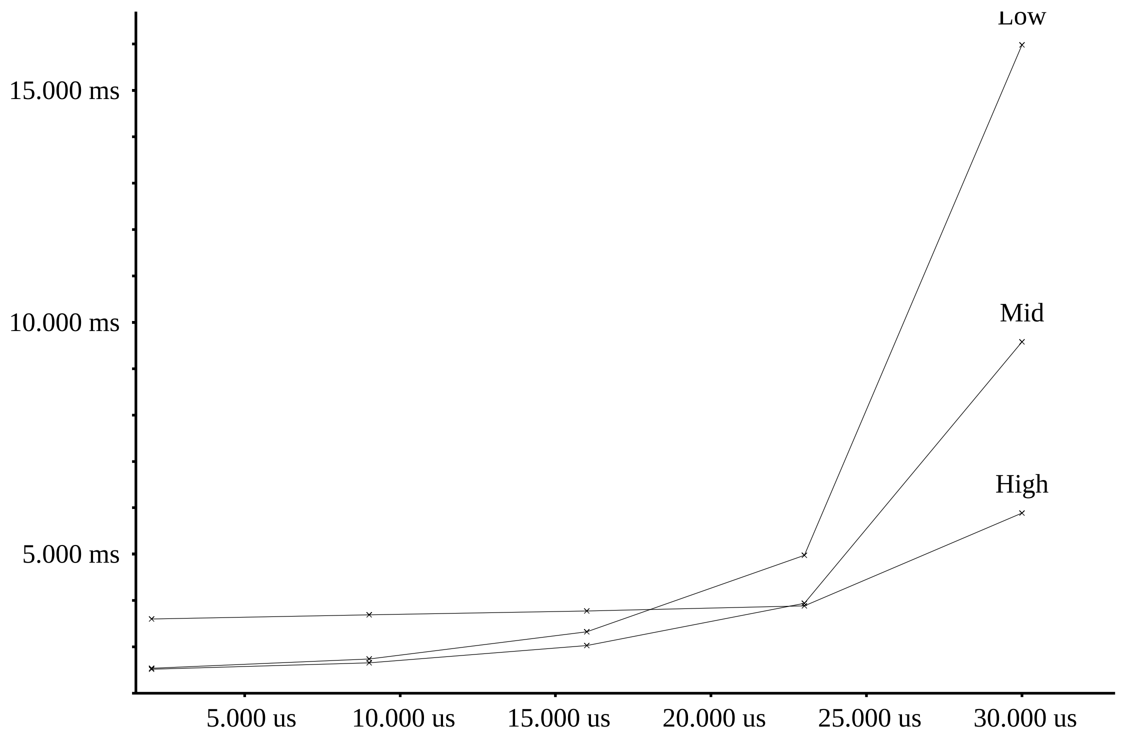

Figure 4-21: Priority Comparison of Average End-to-End Delay (1a-2a) for Poisson Processor Loaded Topology

The importance of compression can also been seen in the access delay. Another comparison with the various priorities is shown in Figure 4-22. We use the average access delay from switch 3 as an example. Since there is no link queueing associated with access delay. The low priority delay is immediately greater than the high priority delay.

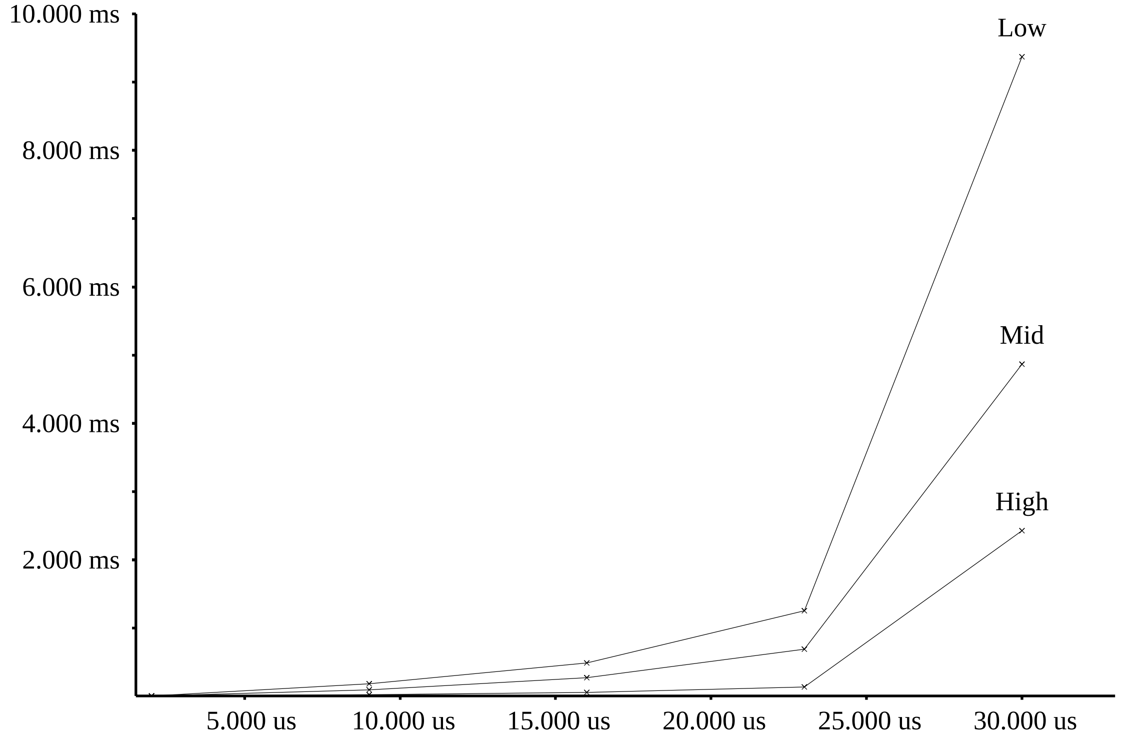

Figure 4-22: Priority Comparison of Average Acc. Delay (Switch 3) for Poisson Processor Loaded Topology

Finally, the property holds also when the we are examining the 99.9 percentile of data. We see in Figure 4-23 that the delays are increasing, with the lowest priority ack simulation having the fastest increase. However, this only occurs just when we are starting to push the limit of the processing power. The expectation is that most switches would be designed to be backed off from this point in which case any of the priority schemes could be used. This is discussed further in the conclusion of this work.

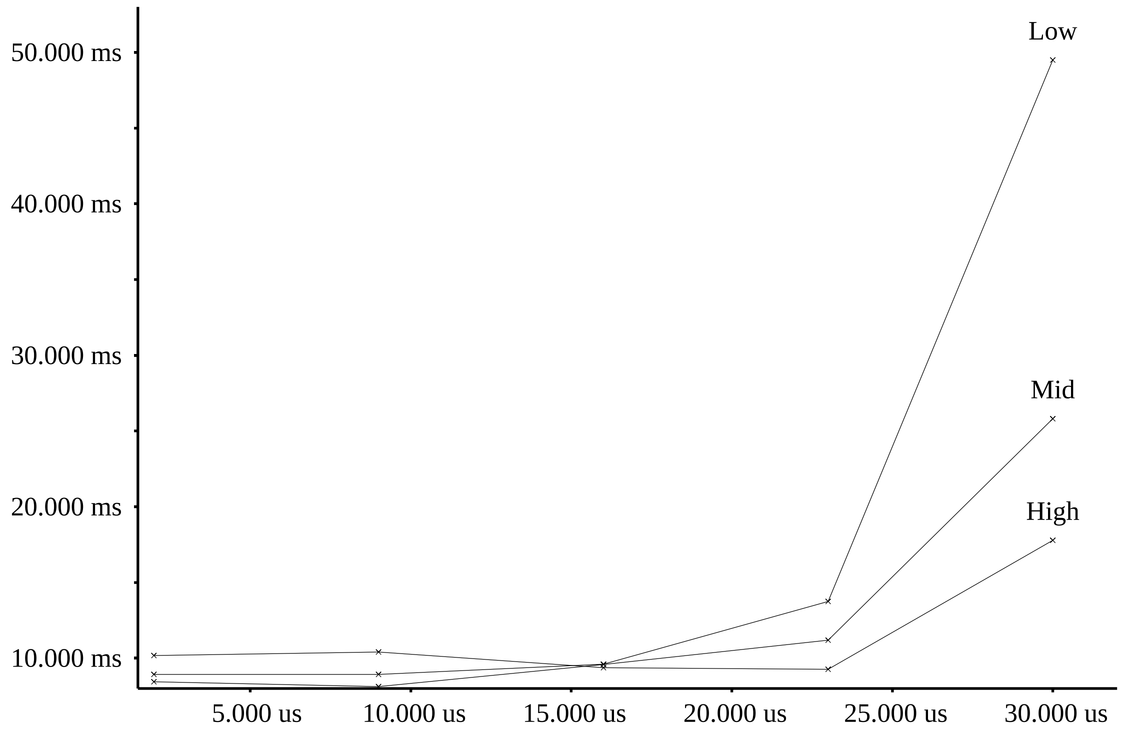

Figure 4-23: Priority Comparison of Average End-to-End Delay (1a-2a) for Poisson Processor Loaded Topology

NDU size

The expectation was that the NDU sizes would still be reasonably large for the processor loaded case. However, when comparing the send and receive NDU sizes for high priority acks as in Figure 4-24, the output NDU sizes did not come out as high as expected. The sizes are still slightly larger than the link loaded case, but they are in the 80 to 95 microsecond range. This in the worst case is a byte size NDU of 1500 bytes or about 30 cells. It should be noted that while this is a large size, it is only about half the expected average size if no ADUs are broken.

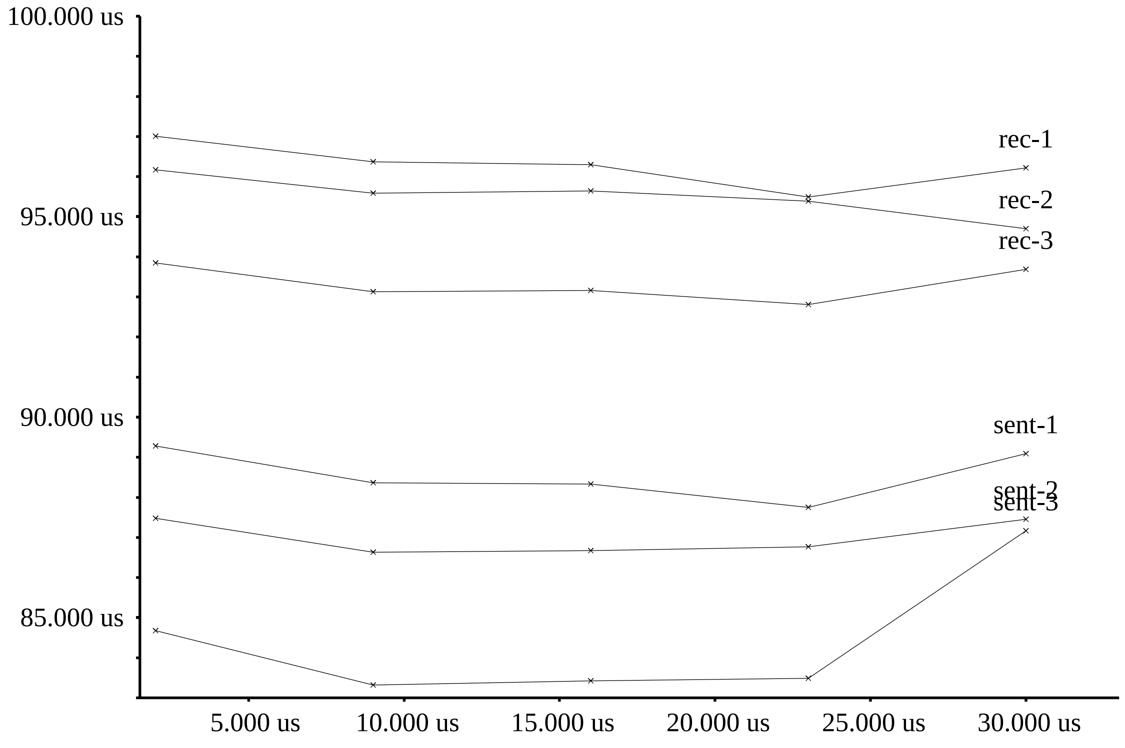

Figure 4-24: Average Send/Receive Size of High Priority Ack Traffic for Poisson Processor Loaded Topology

4.4 TCP Bulk and VBR Video Sources

The results of the TCP bulk and VBR video source simulations closely followed the results of the poisson source simulations. The primary difference between the two sets of simulations was the number of ADUs being processed and the average ADU size being processed. The access delay trends were identical. Therefore, the access delay plots for each of the TCP simulations were omitted.

4.4.1 TCP Base Topology

End-to-End Delay

Figure 4-25 shows the end-to-end delay of the VBR source-sink pairs for the basic topology. Since bulk transfer has less constraints on the end-to-end delay, the end-to-end delays of the TCP bulk sources was not shown. The throughput of the bulk transfers was examined. In all cases, the throughput was such that the traffic being offered to the network was somewhere between 80 and 90 percent utilization for loaded links. We are more interested in how the real-time delays are affected by the changes in processing power.

As can be seen, the delay increases much more quickly than in the poisson simulations. This can be mainly attributed to the fact that 40,000 ADUs are being processed a second as opposed to 5,000 in the poisson case. Depending on the constraints of the network, it may be acceptable to have processing time at either 9 microseconds or 16 microseconds. The 99.9 percentile plot didn't show any significant difference between the two processing times. Therefore, we choose the better of the two delays and used 16 microseconds as being acceptable.

At 16 microseconds and only one link fully loaded (at switch 3), we calculate the RPP to be:

$$RPP_{preempt} = \frac{\frac{200}{16}}{1} = 12.5 MIPS/port$$

When comparing it to the RPP_{cell} above, we see that $RPP_{preempt}$ is over 10 times smaller than RPP_{cell}. So at least 10 ports or an order of magnitude could be supported with the same processing power using the preemptive architecture.

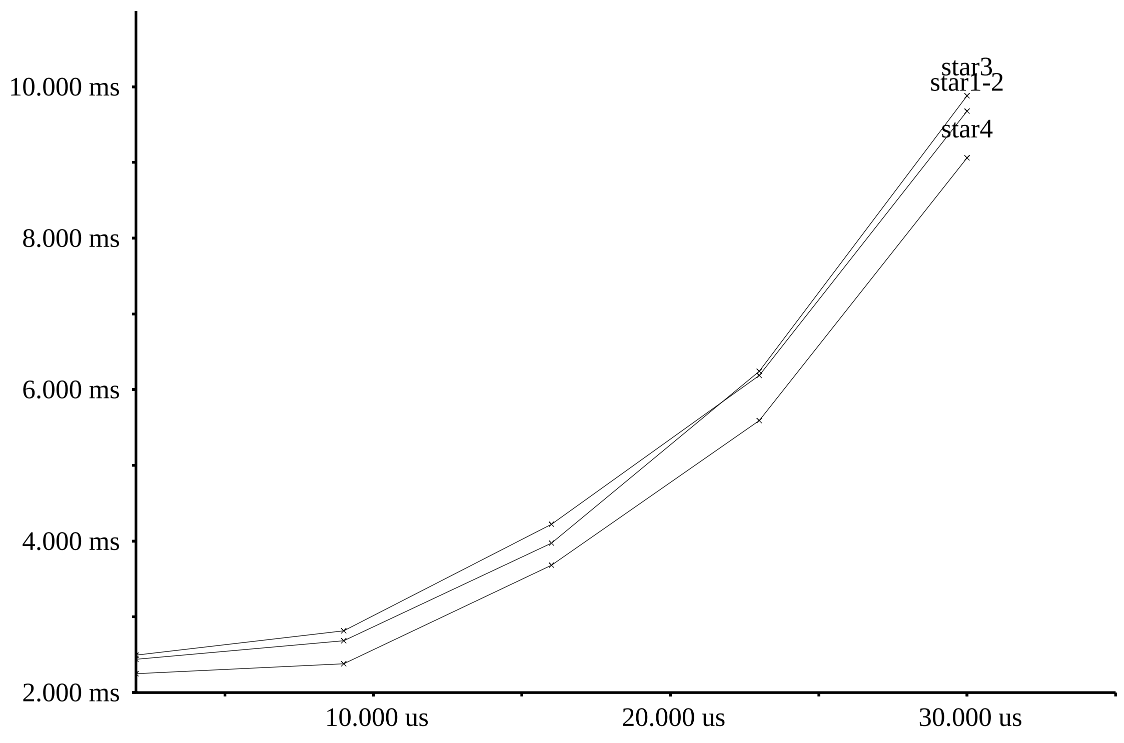

Figure 4-25: Average End-to-End Delay of VBR Traffic for TCP Base Topology

NDU size

The send-receive plot shown in Figure 4-26 shows an interesting effect of increasing processing time. We first note that the send and receive plots form a fairly uniform distance between each send and receive pair. This is primarily due to the throughput of each TCP source. The TCP sources with longer end-to-end delays either because of queueing or propagation delay have lower throughput. Thus, sources one and two have about the same throughput, which is less than the throughput of source three.

Also, sources one, two, and three have lower individual throughput than source four. As we increase the processing time, the real-time traffic ADU size remains constant. However, since there is less processing power, TCP times out more frequently and thus "backs off" . With less TCP ADUs to process in a given time, the average send and receive ADU sizes increase.

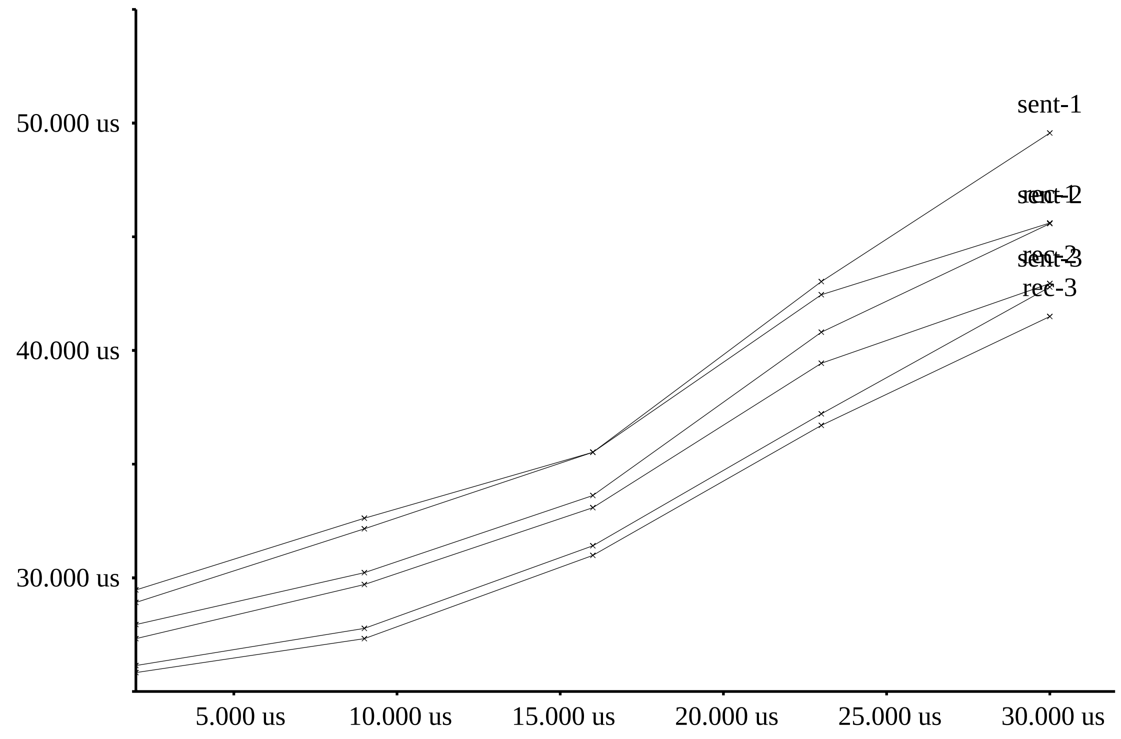

Figure 4-26: Average Send/Receive Size for TCP Base Topology

4.4.2 TCP Link Loaded Topology

End-to-End Delay

As in the poisson simulation for the link loaded topology, switch 2 requires the most processing power. Thus looking at Figure 4-27, we see the same effect of sources one, two, and three having longer delays than source four. The delay does not appear to be too significant until 16 microseconds. Examining the delay in the 99.9 percentile in Figure 4-28, we see that some delay is incurred at 16 microseconds. However, whether the added 4 millisecond delay is unacceptable or not is a engineering decision. We assume that the delay is acceptable and calculate the $RPP_{preempt}$ based on a 16 microsecond processing time.

Since this is the fully loaded link case, 1.5 links are fully loaded at switch 2. Therefore, we calculate the RPP for the preemptive switch to be:

$$RPP_{preempt} = \frac{\frac{200}{16}}{1.5} = 8.3 MIPS/port$$

Doing the standard comparison to RPP_{cell}, we see that $RPP_{preempt}$ has over 17 times smaller processing power than RPP_{cell}. In this case, only 17 ports could be supported with the same processing power using the preemptive architecture that the cell architecture uses for one port.

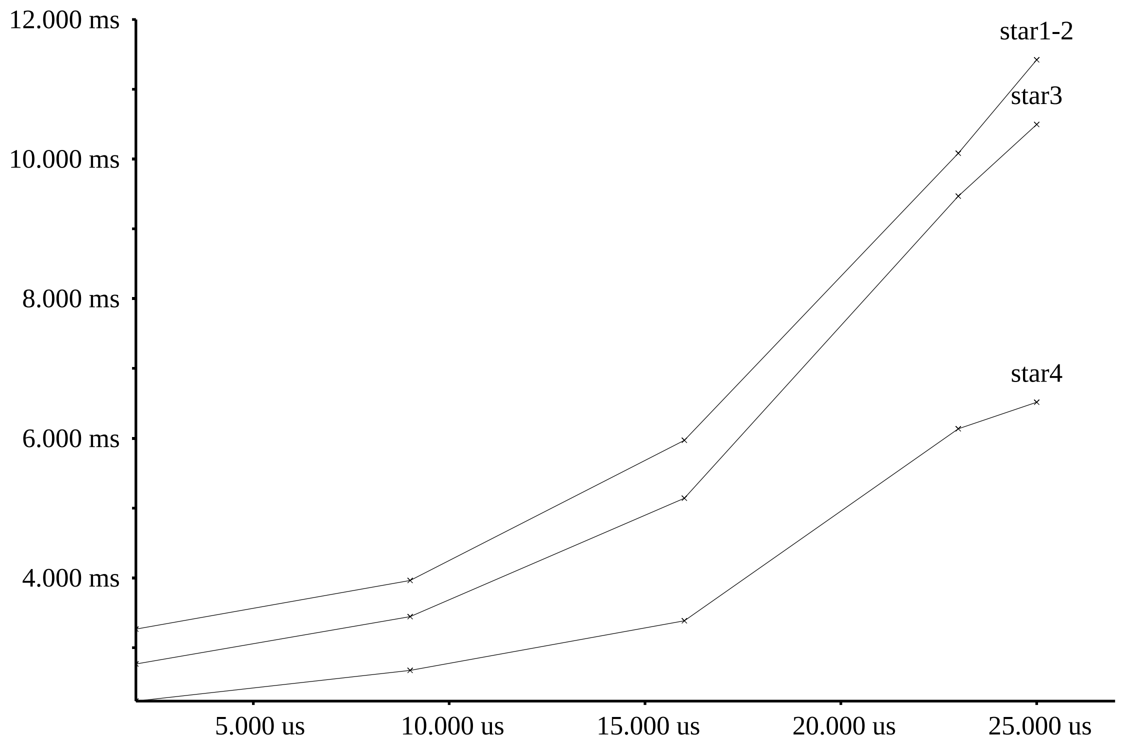

Figure 4-27: Average End-to-End Delay of VBR Traffic for TCP Link Loaded Topology

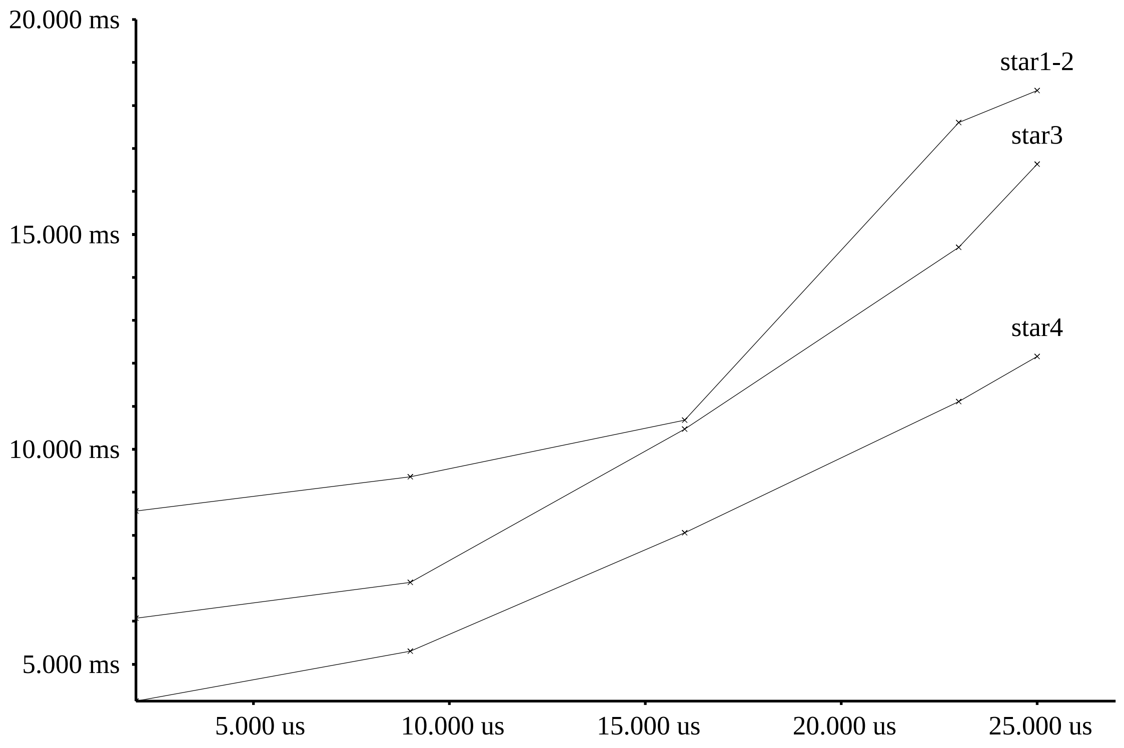

Figure 4-28: End-to-End Delay of VBR Traffic at the 99.9 Percentile for TCP Link Loaded Topology

NDU size

The NDU size for the link loaded topology has the same characteristic as the base topology for TCP bulk source and VBR video.

4.4.3 TCP Processor Loaded Topology

End-to-End Delay

Figure 4-29 shows the average end-to-end delay of the VBR traffic. The delay increases somewhat dramatically at 10 microseconds for sources one, two, and three. Also, in examining the 99.9 percentile plot, it was found that increases were upwards of 7 milliseconds from 6 microsecond processing time to 10 microsecond processing time. Thus, we choose the 6 microsecond processing time as being acceptable.

As in the poisson simulation, switch 3 is the heavily loaded switch. It has three fully loaded links. In doing the calculation and making the comparison it was found that $RPP_{preempt}$ was 13 times smaller than RPP_{cell}. So, 13 ports could be supported with the same processing power using the preemptive architecture that the cell architecture uses for one port.

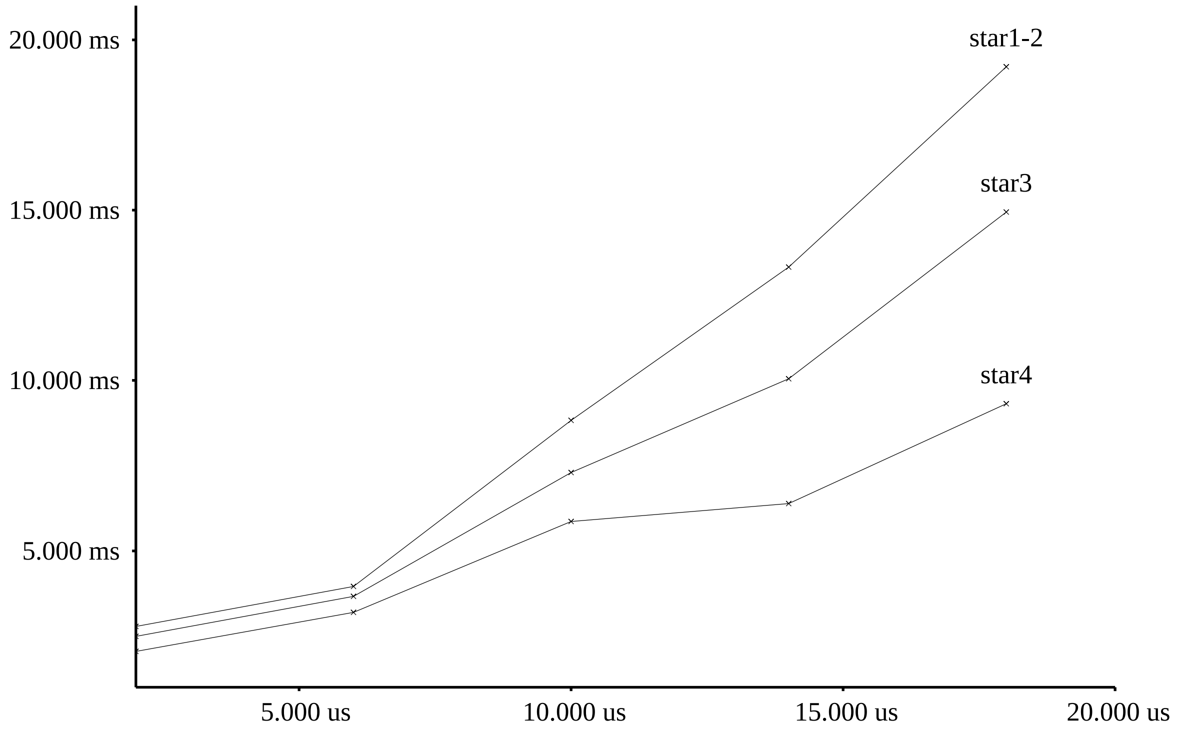

Figure 4-29: Average End-to-End Delay of VBR Traffic for TCP Processor Loaded Topology

NDU size

The NDU size for the processor loaded topology has the same characteristic as the base and link loaded topologies for TCP bulk source and VBR video.

Chapter 5

Conclusions

In this work, an alternative network and switch architecture was explored. The network architecture had the goals to provide:

- The capability to integrate services easily
- The capability to build switches cheaply
- The capability to transmit data efficiently

These goals then lead to a second set of goals for building the switch. The goals in building the switch were:

- Minimize the Switch Processing Power
- Provide a scalable architecture with respect to port size

From these goals a set of design principles were developed, modeled, and simulated. The results of the simulations were used to justify some unique problems and features of the combined architectures.

To achieve the network goals set out this report proposed using preemption as a mechanism to send and receive data. Data was classified into two types of transfer: the end-to-end unit, called the Application Data Unit and the local unit called the Network Data Unit. ADUs were the single chunks of data sent from the source and expected to be received as single chunks of data by the sink. NDUs were the chunks of data sent in between any two switches or host-switch pairs. Preemption helped by controlling the end-to-end delay of ADUs in the network.

Also the switch architecture was designed to use preemption as a justification for dumping large ADUs onto the network. The larger the ADUs, the less required processing power the switch needs. The switch architecture exploits a sharing of processing power among many ports in a switch. Processing was done at the ADU level, but ADUs broken into NDUs could cause a reprocessing to occur. The architecture was examined carefully to understand how the switch processing power could be minimized and therefore the scalability with respect to the port size could be maximized.

5.1 Comparison of Cell and Preemptive Architectures

The primary objective was to find the amount of SPP saved by using a preemptive architecture over a cell architecture. Results show that for this type of traffic mix, the amount saved by using the preemptive architecture is at minimum 30 times less than the cell architecture. Thus, all things being equal, if two switches were being built with the same port size and link bandwidths, the preemptive architecture would need 30 times less processing power. The savings in processing power implies a more scalable architecture with respect to port size and link bandwidths of a switch.

5.2 Comparison of Packet and Preemptive Architectures

As can be seen in the presentation of this report, no simulation or direct comparison was done between the Packet and Preemptive Architectures. From the explanations in Chapter 2, we know that the packet architecture has similar required processing power compared to the preemptive architecture. The preemptive architecture may have slightly smaller required processing powers, since the average ADU size can be larger than the average packet size without loss of performance. However, the expected difference between processing powers will be less than an order of magnitude.

The packet architecture, however, can only support certain traffic mixes and guarantee delay bounds at the same time. If the traffic mix for a network is known and the delay bounds can be guaranteed through either simulation or analytical techniques, the packet architecture should be picked. This is because it is a proven technology, and complexity for the preemptive architecture will probably be higher. If the traffic mix is not known or the mix is found to not be able to guarantee acceptable delay bounds, then an alternative architecture, such as the cell or the preemptive architectures, should be chosen.

5.3 Priority Games

For the initial set of simulations using poisson sources, priority was varied for the small ADU (or ack) traffic and processing time was also varied to see the effect on switch performance. It was found that increasing the priority of the small ADU traffic to be high decreased the processor sharing delay, but increased the link sharing delay. Since the small ADU traffic took a small proportion of the bandwidth of a link, the link sharing delay was only increased slightly. This advantage gained by the high priority small ADU was only useful when the switch was overloaded. The performance degraded more gracefully in this case and the switch processing power could be lower for such an environment. However, such an environment had the cost of causing the NDU size to decrease. This may have adverse effects on the handling and scheduling of the ADU as it flows through a switch, but no first hand effects were found to occur in this case.

The advantage gained by using such a scheme of giving high priority to smaller ADUs is useful only when the processor is the scarce resource in the network. Enforcing such a scheme to have small ADUs be a low load on the bandwidth of the link is a separate problem in and of itself. If such an enforcement was not made, then users would be encouraged to make ADUs small and let them be processed first. This problem was beyond the scope of this report.

5.4 TCP with Video in such a network

As a last set of experiments, simulations were run with TCP bulk transfer and VBR video trace sources. The number of ADUs being processed per second by a switch increased from 5,000 to 40,000. Also, the VBR video gave a flavor of the possible characteristic of real-time traffic in future networks. Results show that even with the order of magnitude increase in ADUs to process, the amount saved by using the preemptive architecture is at minimum 10 times less than the cell architecture. This result further supports the concept of a scalable architecture based on preemption.

5.5 Future Directions

No piece of work is complete without a list of open questions that should be answered for future research. Below is a list of potential problems that were not going to be resolved before completion of the report. These potential problems were not studied due to lack of time. The objective in this work was to take a look at just how good the advantages would be and to examine the one particular problem of fragmentation.

Errors Handling errors in the transmission of ADUs; should they be dropped, corrected, or controlled?

Buffers When buffering becomes full in a switch how do we determine which ADU(s) to drop? Do we drop all or part of an ADU?

Congestion How should congestion control algorithms be integrated into such a system. Can extra time be allotted for algorithms that take longer to process?

Flexibility It is conceivable that different algorithms will produce better results than others for only specific sets of traffic. Is the preemptive architecture able to handle different algorithms on a port by port basis to give us better performance?

Engineering There is a large gap between an architecture model and a implementation. A lot of engineering decisions will have to be made about format of ADU and NDU, link speeds, and the like. Will the effort to create a new network be worth the added gains discovered by this work?

The first two questions on error and buffering are very important and should be investigated before an implementation is proposed. Two more questions came to mind in the process of completing this report. The first was whether the switch architecture could be used in a non-preemptive environment and what are the implications of such an architecture? The second was should a balancing between link sharing and processor sharing be used in future networking? These two questions could be answered in a number of ways in research. Here is the take from the perspective of the work just completed.

Both the Internet Protocol(IP) and Asynchronous Transfer Mode(ATM), the two leading packet and cell architectures respectively, are flexible enough to use the switch architecture without preemption. In the case of IP, a mechanism exists called fragmentation/reassembly which provides the capability to segment a large packet into smaller pieces. While the network would have to make the fragments before starting to send the packet, an algorithm could be imagined which fragments the packet into small enough segments that each segment only takes up a short amount of time on the link. Since processing is done at the packet level, fragments are reassembled and refragmented as necessary at each switch in the path. The biggest problem with such a scheme is how to handle dynamic routing, a feature of the Internet specification. Such a scheme would not have to depend on the guarantee that all data follows one path. Also, while there is no explicit specification that fragments may not be reassembled in the network, generally no such mechanism exists. Adding such a mechanism should not be too difficult and could be done incrementally. Thus, with some effort a router (Internet switch) could be implemented to have the architecture described in this report.

The ATM architecture could also have such design of a switch. However, the ATM model is a little harder to twist to use the switch architecture. Certainly a preemption model exists in a cell network. The preemption model is that we have a choice to preempt every cell or continue sending the same end-to-end unit. One model is to have IP running over ATM and process ATM cells in groups which form the IP packets. Such a scheme would work nicely. However, other ATM cells which don't relate to an IP packet must have some form to be processed as blocks. ATM's adaptation layer might be used to process end-to-end units instead of every cell. The use of the adaptation layer does not solve the problem of stream connections, however. These are connections which don't have any end-to-end units, rather they expect a continuous stream of cells/data at the two end hosts.

Finally, we ask the question of whether finding the balance between link sharing and processor sharing is necessary. Certainly if processing power becomes a scarce resource or bandwidths increase to the point where processing power must be conserved then balancing will be useful. However, currently with growth trends for processing power doubling every year such an outlook is discouraging. The only argument in favor of such an architecture is cost. In that respect saving an order of magnitude in required processing power is significant since it could be used to make a switch more scalable or make it cheaper. Adding the requirement that we can do processor sharing is simple. Besides bandwidth being a parameter we specify in setting up a connection, we also specify the size of the end-to-end units being sent. This statistic should not be too hard to produce since many reservation schemes and algorithms already require a parameter to measure the burstiness of the traffic. Such a measurement should be close to the end-to-end unit size for the network architecture.

In summary this report presents an argument as to how processing power could be saved at the switch level using an alternative view of networking. Saving processing power requires some kind of sharing which results in an understanding of the traffic being sent through the network. Such an understanding is not far from the network designers grasp since many of the properties of the traffic are already determined by the congestion control algorithms proposed. While this report used preemption as the mechanism to provide such an architecture, other architectures could be modified to provide a "preemptive" like service. Such a mechanism must exist in some form for the sharing to occur. Otherwise, traditional thoughts of switch implementations must be used to provide service.

Bibliography

[1] Dimitri Bertsekas and Robert Gallager. *Data Networks, Second Edition*, chapter Three. Prentice Hall, 1992.

[2] Herwig Bruneel and Byung G. Kim. *Discrete-Time Models for Communication Systems Including ATM*, chapter Two. Kluwer Academic Publishers, 1993.

[3] David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM'90 Symposium*, 1990.

[4] M. W. Garrett and M. Vetterli. Congestion control strategies for packet video. In *Fourth International Workshop on Packet Video, Kyoto, Japan August 1991*, 1991.

[5] J. N. Giacopelli, W. D. Sincoskie, and M. Littlewood. Sunshine: A high performance self routing broadband packet switch architecture. In *Proc. of the Inernational Switching Symposium*, 1990.

[6] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88 Symposium*, 1988.

[7] Tony T. Lee. A modular architecture for very large packet switches. In *IEEE Transactions on Communications*, 1990.

[8] Jorg Liebeherr, Ian F. Akyildiz, and N. Tantawi Asser. An effective scheme for pre-emptive priorities in dual bus metropolitan area networks. Technical report, IBM Research Division, 1992.

[9] Sarit Mukherjee, Debanjan Saha, and Satish K Tripathi. Performance evaluation of a preemptive protocol for voice-data integration in ring-based lan/man. Technical report, University of Maryland, College Park, MD, 1993.

[10] Ronald W. Wolff. *Stochastic Modeling and the Theory of Queues*. Prentice Hall, 1989.

[11] Y. S. Yeh, M. G. Hluchyj, and A. S. Acampora. The knockout switch: A simple modular architecture for high performance packet switching. In *Proc. of the International Switching Symposium*, 1987.

[12] Lixia Zhang, Scott Shenker, and David D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *SIGCOMM '91 Conference on Communications, Architectures, and Protocols*, 1991.