Optimal Priority Packet Discarding for Queue Overload Control

David W. Petr

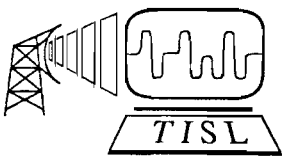
Victor S. Frost

TISL Technical Report TISL-6731-22

Prepared for

National Science Foundation
and
AT&T Information System

March 1990



Telecommunications and Information Sciences Laboratory
The University of Kansas Center for Research, Inc.
2291 Irving Hill Drive Lawence, Kansas 66045

Abstract

This dissertation deals with controlling overload in a finite queue of a packet switched communications network. We accomplish the overload control at each queuing point by selectively discarding packets based on a delivery priority associated with each packet. Our goal is to identify discarding policies, specifying when to begin discarding and which packets to discard, that are optimal with respect to an expected discarding cost performance metric. We develop a general model for the analysis of priority discarding systems and classify such systems into four basic structures. We develop more complete analysis models for two of these structures and analyze their behavior using the methods of stochastic dynamic programming. For systems that make discarding decisions for every Arrival interval and discard only Arriving packets (A/A systems), we derive computationally efficient recursive algorithms for finding the state distribution, relative values and expected discarding cost of any policy, including the default policy of discarding packets only if the queue is full. Combined with the policy improvement algorithm of dynamic programming, this yields an efficient method of finding the optimal discarding policy for any set of system parameters. We find that optimal A/A policies significantly outperform the default discarding policy over a wide range of system parameters. We also find that additional queue capacity can improve performance, but that the improvement diminishes considerably with increasing levels of overload. For systems that discard once every Fixed Service interval and are allowed to discard any Queued packets (FS/Q systems), we also derive a computationally efficient means of finding the relative values and expected discarding cost of a policy. For systems allowed to retain only one packet at each discarding decision, we identify a very small family of optimal policies, derive closed-form expressions for each member's optimality conditions and expected discarding cost, and show that one of the member policies must be optimal for any system parameters. We find that optimal FS/Q systems exhibit performance similar to optimal A/A systems. We conclude with a discussion of the implications of these results for the design of integrated packet networks.

Table of Contents

Chapter 1

Introduction

CONTENTS

Chapter 1

LIST OF FIGURES

Chapter 1

Introduction

Research on wide-area telecommunications networks is beginning to focus on architectures and capabilities which will supersede today's emerging Integrated Services Digital Network (ISDN). This successor network has been referred to as the integrated services *packet* network [Turn86] and Broadband ISDN (B-ISDN) [Hand89]. The economies and flexibility of an *integrated* network capable of transporting computer data, speech, graphics, image, video and other signals make such a network very attractive, and packet network architectures have the potential for realizing these advantages. However, the effective integration of these diverse traffic types into an Integrated Packet Network (IPN) presents many new problems and requires new thinking about network design priorities. An integrated packet network is illustrated in Figure 1-1.

For example, in the first packet networks such as ARPANET [Kahn71], the traffic consisted of relatively narrowband computer data which had to be received error-free. However, this type of traffic could tolerate considerable delays and could even be held at the source if the network was congested. Transmission link speeds of 1200, 2400 or even 9600 bps were slow by today's standards, and the nodal processing required for each packet was only of secondary concern. In this environment, link-by-link error and flow control protocols made engineering sense, and many clever protocols were devised. However, time-critical, broadband signals like those cited above will demand broadband transmission links using optical fiber technology and broadband packet switching. Of these two technologies, switching is likely to

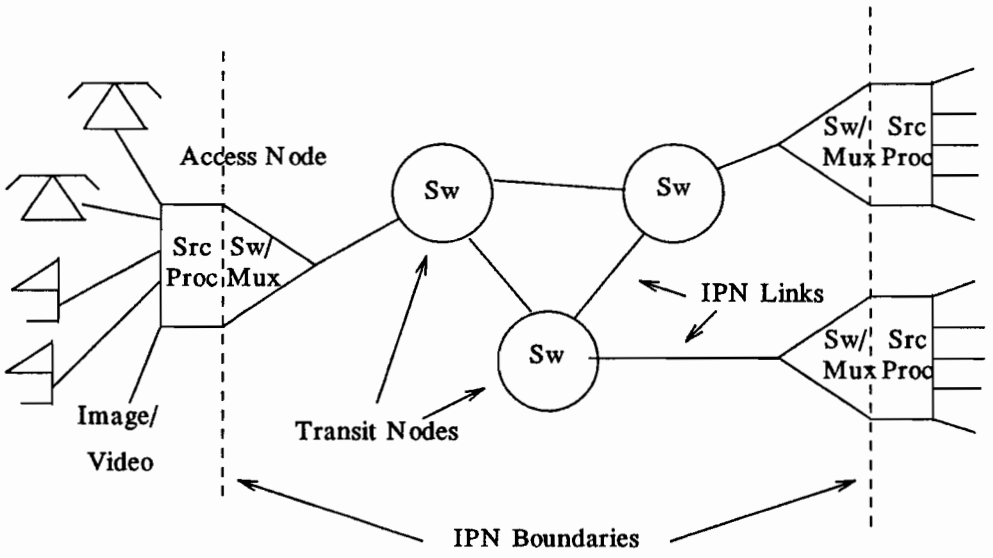


Figure 1-1. An Integrated Packet Network

be the bottleneck in future IPNs. Although processing speeds will continue to increase, it will also be necessary to minimize the nodal per-packet processing requirements imposed by the network design. Processing associated with individual sources, while important in these networks, is not likely to be a limiting factor in the network's speed or capacity. This is the motivation for new switching concepts like Fast Packet Switching [Muis86] and Asynchronous Transfer Mode (ATM) [Ride89], in which protocol processing within the network is kept to a minimum at the expense of increased source processing.

However, the presence of these new signals can also provide new flexibility which can be exploited in the network design. While computer data signals must generally be received error-free in order to be useful, the inherent structure of speech and image signals and the way in which they are perceived allows for some loss of information without significant quality impairment. This presents the possibility of *purposely* discarding limited information to achieve some other goal, such as the control of temporary congestion. This is indeed fortunate, since these signals are likely to exhibit extreme burstiness (large peak to average traffic ratios),

which is likely to cause short but perhaps frequent periods of extreme congestion.

1.1 Problem Statement

The research described in this dissertation is concerned primarily with controlling such short periods of congestion, or overload, in an IPN by discarding packets. It is motivated and guided by the three new principles for IPN design mentioned above: greater need for overload control, packet loss flexibility due to signal structure, and the need for minimal per-packet processing at network nodes.

We specifically wish to find packet discarding policies (or algorithms) which accomplish this overload control with the smallest possible service impact, *i.e.*, optimal policies. Unlike much previous work in network optimization, our objective is not to minimize delay or even to minimize packet losses, but to minimize the *effect* of packet losses. We show that all packets are not "created equal" in terms of loss sensitivity, so that some packets should receive a higher *delivery* priority (as opposed to service priority). The existence of several such delivery priorities, each with a different loss penalty in terms of received quality or some other service measure, forms the basis for our optimization.

1.2 Background

1.2.1 Overload Control in Integrated Packet Networks

1.2.1.1 Definitions

Overload in an IPN is any "short" period of time (on the order of tens to hundreds of milliseconds) during which the arrival rate at a queuing point exceeds the service rate, thus threatening overflow of the (finite) queue. Overload control can thus be contrasted with load management, the control of long-term average network load (*e.g.*, by call denial, throughput negotiations, etc.) so that queues are not *constantly* in overload. The contrast is in both scope and time frame. Overload control applies to individual queuing points in the network, whereas

load management applies to the network as a whole, or at least to subsets of network paths. In addition, the time frames (*e.g.*, for measurement, decision, and action) associated with overload control are orders of magnitude shorter than those for load management.

To get some mathematical definitions of overload we introduce the following notation, assuming here (and throughout this dissertation) that the queue consists of a finite number of packet buffers.

$\lambda(t)$:	arrival rate at time t (packets per second)
$\mu(t)$:	service rate at time t (packets per second)
$\rho(t)$:	normalized load at time t: $\lambda(t)/\mu(t)$
$n(t)$:	number of packets in queue (queue fill) at time t
N:	maximum queue capacity (packets)
τ:	a prediction horizon (order of ms or tens of ms)

The simplest possible definition for overload is any time such that

$$\rho(t) > 1. \tag{1.1}$$

This is essentially the definition used in [Yin87a]. However, if this condition is *very* short-lived, there may be no danger of queue overflow. So a slightly more sophisticated definition for overload could be any period in which overflow is imminent, for example, any period characterized by

$$n(t) + [\lambda(t) - \mu(t)]\cdot\tau > N. \tag{1.2}$$

This differs from condition (1.1) in that it incorporates queue fill and capacity in addition to arrival and service rates. We will find this latter definition useful in our subsequent discussion of overload control options. However, we will ultimately see that this definition is limited since it fails to incorporate information about the *types of packets* in the queue in addition to queue fill.

1.2.1.2 Significance

We emphasize here that overload is an unavoidable short-term phenomenon in any

statistically operated network. It is a fundamental result of queuing theory that probability of overload for a finite queue is always strictly positive (if arrival variance is nonzero) and increases as network efficiency (average load) is increased. This effect is illustrated by the lower curve in Figure 1-2 (taken from [Mura89]) labeled "M-stream Only" for a Poisson arrival process with fixed packet lengths.

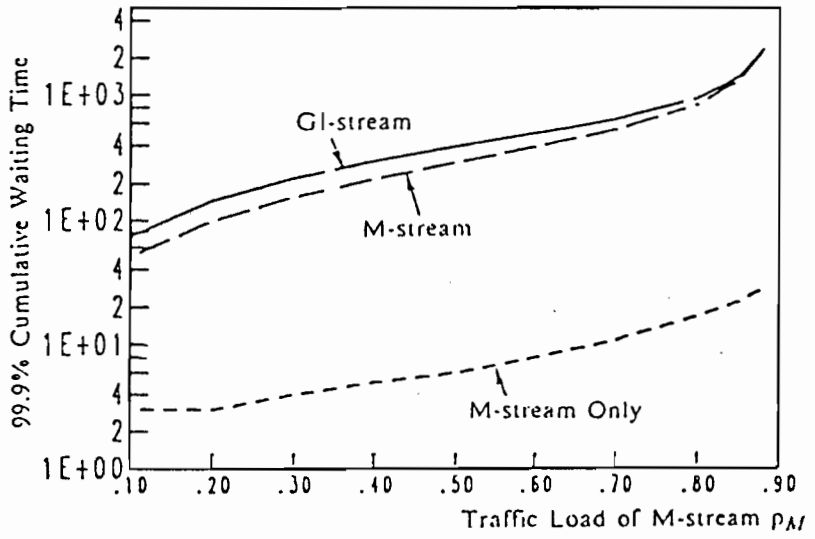


Figure 1-2. Effect of Traffic Burstiness on Queue Overload

This figure is for an infinite queue and plots the 99.9% cumulative waiting time (the minimum T which satisfies Prob[waiting time $\leq$ T] $\geq$ 0.999), that is, it gives an indication of the tail of the waiting time distribution. This is a measure for infinite queues that behaves similarly to overload probability for a finite queue. Because of this effect, network designers and operators are forced to strike a balance between the financial advantages of increased network load and the service impairment of overload. As stated in [Pouz81], "Taking no chance of overflow requires more resources than taking some chances $\cdots$ As a whole, there must be an acceptable balance taking into account all costs and resources."

Furthermore, the overload problem is *intensified* as the traffic burstiness increases. This is seen in the curves labeled "GI-stream" and "M-stream" in Figure 1-2. For these curves, the authors added a bursty traffic stream (the GI-stream) with average load less than 0.1 to the Poisson arrival stream (the M-stream). The GI-stream was an "on-off" stream, either generating packets continuously (on) or generating none (off), with the length of the on and off intervals geometrically distributed with means of 100 and 1000, respectively. As such, the GI-stream represents a very bursty arrival process added to the "smooth" Poisson process. The curves show that, even though the average load of the GI-stream is low, its burstiness significantly increases the cumulative waiting time for *both* streams. For a finite queue, this would translate into significantly increased overload probability.

It is significant, then, that future IPNs carrying speech, graphics, image and video traffic are likely to exhibit considerable burstiness. A document browsing service, in which one may examine an image of a document before ordering or printing it, is a perfect example. In order to be useful in such an application, the image needs to be of high quality and delivered very quickly (less than a second per scanned page), but there would be relatively long intervals between transmissions. For speech traffic, references [Heff86, Srir86] point out that the presence of multiple speech sources in a packet network tends to increase the the burstiness of the composite traffic. Many proposals for video coding for future IPNs (such as B-ISDN) call for variable-rate coding, which again produces inherently bursty traffic. Figure 1-3, taken from [Wood90] illustrates the range of traffic burstiness expected of these signals.

Of course, if the average load on the network is kept low enough, the probabilities of overflow at queuing points can be made arbitrarily small. The goal of intelligent overload control, however, should be to allow the *maximum* average network load, subject to some service quality constraints. Thus, we expect overload control to be especially important in future IPNs to help network designers and operators take greatest advantage of statistical

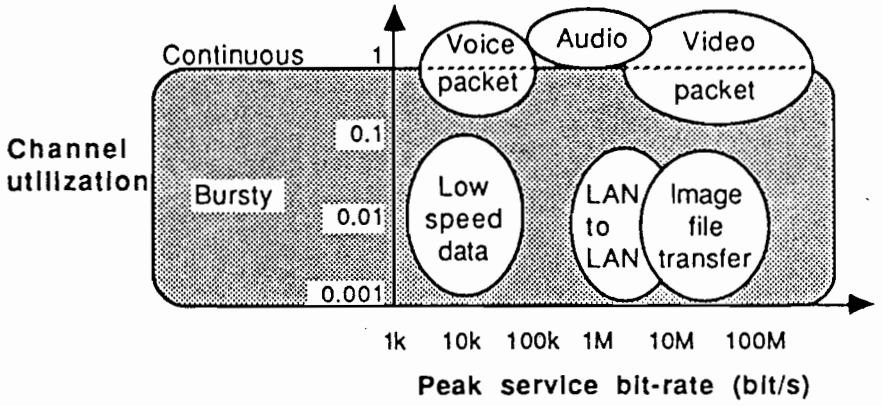


Figure 1-3. Expected Range of Traffic Burstiness

efficiencies even though the aggregate traffic exhibits considerable burstiness. The present research is especially timely since it addresses some of the [Minz89] "significant technical issues and concerns over the tradeoff between statistical gain and quality of service" associated with B-ISDN.

1.2.1.3 Overload Control Options

Because of the short overload control time frames, autonomous control mechanisms, in which each overload point acts independently, are more attractive than algorithms involving any sort of coordinated data exchange or handshaking. Minimal per-packet processing at each switching node is also desirable if switches are to operate under the low-delay, high-capacity constraints necessary for large IPNs. With these restrictions, plus the real-time nature of much of the expected traffic, it is clear that traditional window-based or stop-and-wait flow control techniques (either link-by-link or end-to-end) are undesirable as overload control mechanisms [Gers89]. Here we explore some other possibilities and discuss the applicability of each to traditional computer data and to the new wave of speech, image and video packets. For data packets, we will assume no *a priori* knowledge of the structure of the information which they contain. However, it may be possible to effectively exploit the inherent structure of

some of these newer signals.

Note that the only variables in equation (1.2) which can be manipulated dynamically for overload control are the effective arrival rate (λ), the effective service rate (μ) and the queue fill (n). Assuming that the bytes/second service rate (transmission rate of the outgoing link) is fixed, the following four approaches are possible for overload control.

1.2.1.3.1 Discard Arriving Packets When Queue Is Full

The default approach taken by most current networks is to discard arriving packets only when absolutely necessary, *i.e.,* when the queue is full. In this approach, τ (the prediction horizon) approaches zero and overload is controlled by reducing the effective arrival rate (λ). However, the only controlling parameter is queue capacity (N), which is limited by the delay requirements for speech and video. This technique is clearly applicable to packets carrying any kind of information: in fact, packets would be discarded without regard for the type of information carried.

This approach fails to recognize that some information may indeed be more important than other information, *i.e.,* that the type of information *should* be taken into account. In coded speech, for example, active speech is more important to the communication than background noise during pauses. In image coding, edge information is more important than shading. Some data packets may even be more important than others by virtue of a greater network "investment" in them (*e.g.,* number of nodes already traversed [Gerl80]). We will call this principle *priority discarding,* which we define as *the deliberate discarding of information, in priority order to reduce the effects of discarding, so that some aspect of network performance is enhanced.* An early example of priority discarding is found in the first digital multiplex systems (called T1 carrier systems), in which the *least* significant bit of the Pulse Code Modulated (PCM) speech samples is periodically discarded to provide signaling information.

The remaining three approaches to overload control incorporate this principle of priority discarding.

1.2.1.3.2 Reduce Packet Generation Rate

It is also possible to reduce the effective arrival rate (λ) by reducing the packet generation rate at the source. However, for this technique to be a useful for overload control, the overload point would have to be able to *directly* control the source generation rate. This is possible in local-area networks (for both speech and data packets), but not in more general multi-node networks.

Priority discarding using this technique was applied to a local area network in [Fros86], in which an *embedded coding* algorithm [Good80] was used to code the speech. The basic concept of embedded coding is to structure the code so that it produces "essential" and "enhancement" bits. A low quality (but continuous) signal will be generated if only the essential bits are received, with quality improvements for each additional level of enhancement bits received. Using this technique, the (fixed-length) packet generation rate was reduced during overload by discarding enhancement bits and including more samples in each packet. This was shown to be an effective overload technique in that it allowed significantly more speech sources (higher average load) for a given delay performance, with only slightly degraded speech during overload. A similar technique could be applied to systems with a speech activity detector by suppressing packets containing background noise *only* during overload periods.

We reiterate, however, that this technique is not applicable to general multi-node IPNs.

1.2.1.3.3 Reduce Packet Size

Packet size could also be reduced, in which case the effective packet/second service rate (μ in equation 1.2) is increased to reduce the overload. This technique is clearly not applicable

for data since any alteration of the packet would presumably void the packet's usefulness, but it could be applied to coded speech or image/video traffic. For example, in [Muis86] embedded coding was again used, but speech packets arriving at a queue were *shortened* in response to overload by discarding "enhancement" bits from each packet. Again, this technique was shown to significantly improve the performance of an IPN in terms of delay *vs.* load (number of simultaneous speech sources), with only a slight reduction in perceived quality.

Packet shortening is quite amenable to a general network environment, since each overload point can exercise control autonomously. However, it has the serious disadvantage of requiring network nodes to "know" the internal structure of the packet body and to manipulate packet contents. Besides going against established principles of protocol layering, this technique violates the important IPN design principle of minimal per-packet processing.

1.2.1.3.4 Discard Packets By Delivery Priority

The final approach we will consider is to discard packets according to delivery priority. This differs from the first (default) approach in that the prediction horizon does not approach zero (that is, packets may be discarded before the queue is full) and priority discarding is practiced. The simplest method to implement is to discard arriving packets and thereby control arrival rate (λ). However, more control flexibility is possible by allowing previously queued packets to be discarded, controlling queue fill ($n(t)$). We assume only that relative importance for all packets is indicated by a "delivery priority" in the *network* portion of the packet header [Ride89].

Priority discarding was applied to speech packets in [Yin87a, Yin87b, Bial80]. Methods of determining packet delivery priorities included: embedded coding, with the coding bits for a segment of speech segregated into *separate* packets according to their importance to the decoding process; even-odd samples (see discussion of [Jaya81] below); and multiple energy

detection thresholds (silence, semi-silence, active speech). Queuing points could then autonomously respond to overload (as measured by queue fill thresholds or number of active sources) by simply discarding entire packets in priority order. The performance of delay *vs.* load (number of speech sources) improved in varying degrees according to the particular combination of techniques.

1.2.1.3.5 Dealing With Packet Loss

When information is discarded by the network, the receiver is forced to deal with this loss of information. With computer data packets, the information can be exactly regenerated through standard error recovery protocols. Speech and image signals, however, have many more options since the information does not need to be *exactly* regenerated. Examples of speech regeneration include: replacing a lost packet with silence or the previous packet [Gold77, Cohe80]; ignoring the lost information, when embedded coding is used [Yin87b, Bial80]; sample interpolation, when samples are numbered with some modulus and samples with the same residue are grouped together in packets [Jaya81, Mats87]; and regeneration based on information contained in previously received packets, for example, pattern-matching, pitch-synchronous replacement [Good86] and model-based coding [DaSi89].

1.2.2 Priority Packet Discarding

Of the four options considered, priority packet discarding is clearly the preferred method of overload control for IPNs. But this only creates a whole new set of questions. How does the concept of priority packet discarding affect overall IPN design in general and the transmitters and receivers in particular? What classes of priority-based packet discarding algorithms are there? Precisely how is optimality for a packet discarding algorithm to be defined? What is the optimal control policy for a particular class of algorithm, and how does it depend on traffic parameters? Can the performance of each policy be characterized accurately with analysis?

What are the effects of more or less buffering? Which classes of algorithms perform better than others? Which can be more easily implemented? What is the performance penalty for a simpler implementation?

The next section answers the first question by presenting an example (original to this research) of how priority packet discarding can influence the design of an IPN carrying primarily speech. Previous studies by other researchers in this area are then reviewed briefly.

1.2.3 An Example IPN System Model

We will here give an example of how the existence of a priority packet discarding capability can influence overall network design. Although our example is concerned specifically with speech, the approach can be extended to other structured signals such as graphics, image and video signals. Our concern in the construction of the model is to optimize the entire IPN (transmitters, packet multiplexers, and receivers) for higher speeds and capacities. As such, we allow more complex processing at network edges (transmitters and receivers) in order to simplify the processing at network nodes.

1.2.3.1 Transmitter Subsystem

In our system model [Petr89], the transmitter (Figure 1-4) first classifies speech segments according to models of the speech production process (*e.g.*, voiced sounds, fricatives and plosives). This model-based classification is used to remove redundancy during coding, to assign delivery priorities, and to regenerate discarded speech packets.

Our model makes a one-to-one correspondence between contiguous speech segments and packets. That is, it does not distribute information about a single segment of speech into several (say N) packets [Yin87a, Jaya81, Mats87], even though such techniques could be used in conjunction with priority packet discarding. Such approaches have a fundamental N-to-1 disadvantage compared to our single packet per segment approach. They either produce N

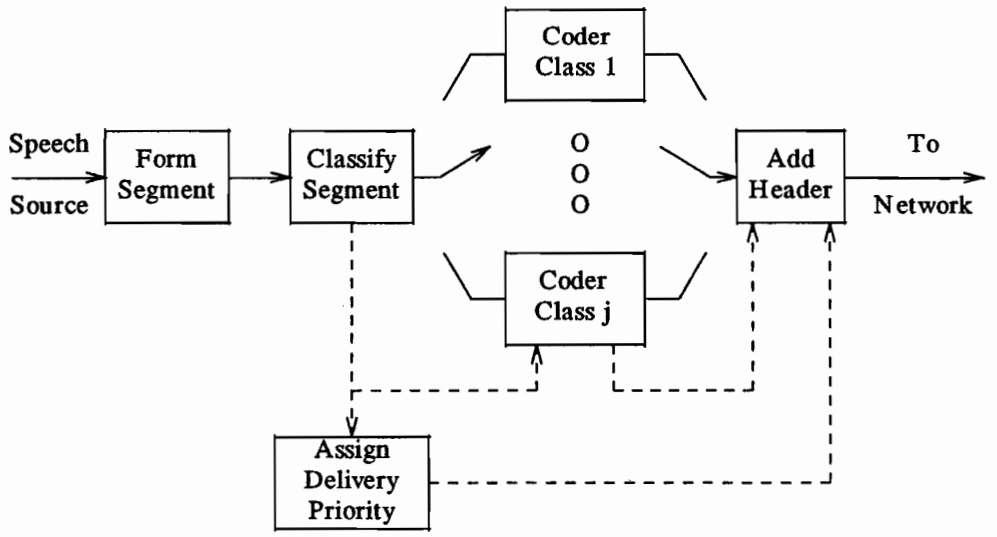


Figure 1-4. Transmitter Subsystem

times as many packets to be processed at each network node (if segment size is held constant) or N times as much packetization delay (if the number of packets produced is held constant). In addition, these approaches fail to take advantage of the relative importance of different segments of speech.

After classification the transmitter removes redundancy from the speech using a coding algorithm based on the determined model. For example, voiced sounds (*e.g.*, vowels) could be coded with a block-oriented pitch prediction coder. After coding, the transmitter assigns a delivery priority to each packet based on the expected distortion which will result if the packet is lost (discarded) by the network and then regenerated at the receiver. The assignment depends partly on the class of speech packets and is determined in some other fashion for data packets (perhaps by giving all data packets the same high delivery priority).

In forming packets from speech segments, the delivery priority would be included in the network portion of the packet header (for example, the ATM header [Ride89]) and the classification and any coding parameters would be included in the end-to-end portion of the

header (ATM adaptation header). In contrast to speech-detector-based transmitters used in all previously cited studies except for [Fros86], all segments of speech (even background noise) are "launched" into the network and only discarded by the network for overload control. We will expand on this point in the discussion of the receiver subsystem.

1.2.3.2 Packet Multiplexer Subsystem

Packet multiplexers (Figure 1-5) exist at each outgoing link of each network node as well as at each multiplexed network access point.

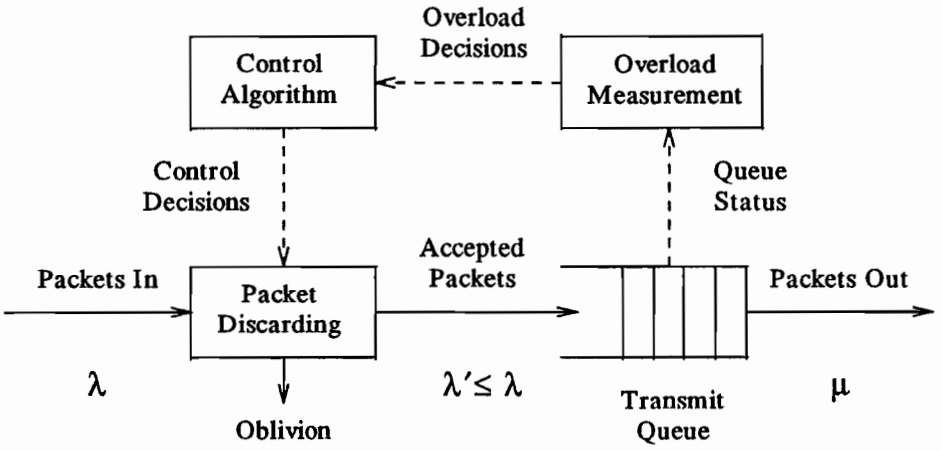


Figure 1-5. Packet Multiplexer Subsystem (Arriving Packets Discarded)

Each packet multiplexer monitors local overload and discards packets according to packet delivery priority (read from the network portion of the packet header) and some locally determined measure of overload level. Specific measures of overload and discarding algorithms are the subject succeeding chapters. Figure 1-5 illustrates the case of discarding only arriving packets, but it would also be possible to discard previously queued packets.

In addition, *if* error checking is performed by the nodes, *any* packet (data or speech) found to have an error is discarded. Some experimental IPNs (*e.g.*, [Muis86]) check for packet errors in data packets but only header errors in speech packets, assuming that it is better to

deliver an errored speech packet than to discard it. Our unified approach to error control is justified since a sophisticated speech packet regeneration mechanism is built into the receiver.

This model requires very little per-packet processing for overload control at the packet multiplexer and the processing (including optional error control) is uniform for all packets. This should result in significantly faster switching nodes relative to other approaches.

1.2.3.3 Receiver Subsystem

The receiver (Figure 1-6) decodes the samples in speech packets delivered to it (based on the classification and coding parameters contained in the end-to-end header) and determines the appropriate time to play them out. By choosing to launch all speech packets (including background noise) at the transmitter, the receiver synchronization problem requires only packet sequence numbers. This is a significant feature in light of the difficulties with the alternatives: global synchronization is administratively difficult and relative time stamps must be modified at each packet multiplexer, requiring additional per-packet processing [Mont83]. Furthermore, potential speech detector impairments (such as clipping) are eliminated whenever the network is not overloaded. Even during periods of considerable overload, the received quality may be better if at least a few "background noise" packets are delivered and then used to regenerate noise which is similar in character to the actual noise.

If a packet is lost for any reason (discarded by the network because of overload or errors, excessively delayed in the network, etc.) the receiver must first detect the loss by inspecting sequence numbers of those packets which are received. It must further make a determination of the class of each lost packet, so that the appropriate regeneration model can be applied using previous header and sample history. A correct class determination will be critical to accurately regenerating the lost information, but this is easily done with our model as follows. In a string of packets with the same class, we can virtually ensure that the *first* packet will be received by

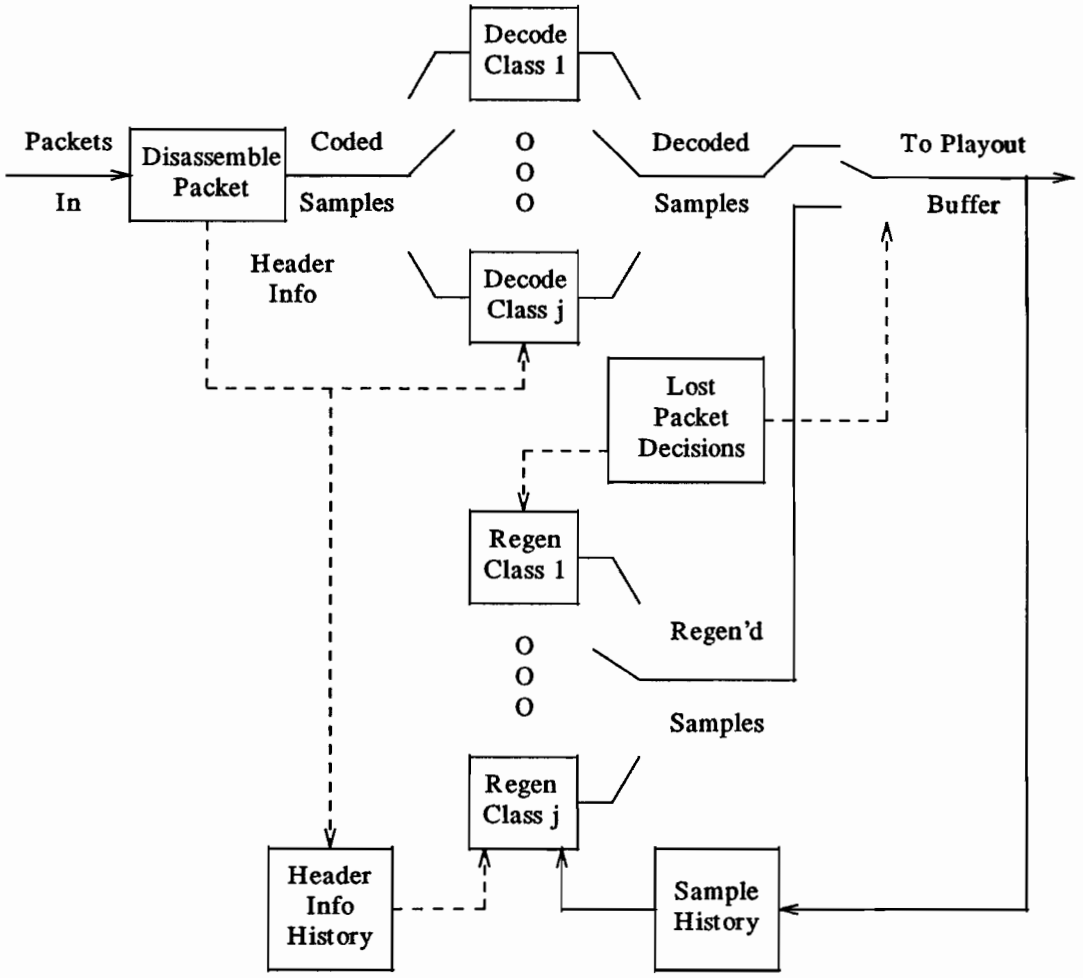


Figure 1-6. Receiver Subsystem

assigning it a high delivery priority. If we assume perfect delivery of these first packets, the class of any lost packet will match the class of the last received packet. Thus the receiver's class decisions can be virtually error-free.

1.2.3.4 Advantages

We now briefly summarize the advantages which we gain by incorporating the concept of priority packet discarding and then taking a total system approach to this problem. The model has the following characteristics:

1. Provides a powerful overload control mechanism.
2. Effectively exploits the signal structure.
3. Allows extremely simple per-packet processing for overload control.
4. Requires only one packet per signal segment.
5. Simplifies receiver playout synchronization.
6. Allows reduced per-packet error processing at packet multiplexers.

1.2.4 Previous Related Research

1.2.4.1 Excerpts Showing Significance

We begin this section with a collection of brief quotes from the recent literature, each of which mentions the importance of priority discarding. Many of these references (and others) are then summarized. Italics are the added by the present author for emphasis.

In [Chen88], the need for discarding mechanisms is identified:

> We speculate that congestion control can be implemented by simply *discarding packets*, in which case a mechanism for choosing *which packets to discard* might be needed.

In describing overload control, it is stated in [Li88] that:

> Such control is necessary and important in order to ··· protect *high priority messages* from low ones.

In discussing a priority field for ATM headers, [Ride89] states that it should be used to:

> ··· provide a real-time response mechanism for determining *which cells are discarded* in extreme congestion conditions.

In [Ohni88], traffic is divided into loss-sensitive and delay-sensitive classes, and a scheme is proposed in which:

> ··· loss-sensitive cells always have *[buffer input] priority*, and when required for the loss-sensitive class input, delay-sensitive class cells within the buffer are *discarded*.

In [Bial80], a congestion control method for speech is described:

> Our flow control method is based on a *priority-oriented* packet transmission format ··· [in which] lower priority packets may be

> *discarded* without affecting the continuity of speech, but voice quality will degrade as more data are omitted ···

In [Yin87b],

> ··· a family of congestion control schemes is proposed. They are all based on *selectively discarding packets* whose loss will produce the least degradation in quality of the reconstructed voice signal.

The authors in [Yin87a] describe one such voice packet priority system in which:

> ··· more significant information is carried by *high priority packets* and less significant information by low ones ··· [and] problems of congestion at network nodes are easily handled locally by *dropping low priority packets,* without having to resort to end-to-end control.

In discussing packet losses caused by the burstiness of packet video, [Nomu88] states:

> Degradation caused by this type of packet loss could be controlled by introducing a *"priority"* into networks ··· networks should keep the loss rate of [high] priority packets as low as possible.

1.2.4.2 General Analyses

We now begin a more thorough review of some of the recent research related to priority packet discarding for overload control, starting with general analyses.

In [Gerl80, Kamo80a] we find some possibilities for the structure of packet discarding algorithms. The authors describe several methods of flow control mechanisms which arbitrate between *classes* of traffic competing for a common buffer pool in each node. We need only generalize how the classes are determined (by delivery importance), the entity over which the pool is shared (an outgoing link *vs.* an entire node), and the motivation for the techniques (minimizing discarding effect *vs.* increasing throughput, preventing deadlocks and promoting fairness) to apply the family distinctions directly to our priority discarding problem. The methods include:

1. Complete Sharing (CS): all available buffers are shared between all classes on a first-come first-served basis. In an overload control context, this corresponds to the default

option of discarding packets if and only if the queue is full.

2. Complete Partitioning (CP): available buffers are partitioned among the classes, with no sharing of buffers between classes.

3. Sharing with Maximum Queues (SMXQ): available buffers are shared, but a limit is place on the number of buffers which can be used by each class at any time. The sum of these class limits may exceed the total number of buffers available (overselling).

4. Sharing with Minimum Allocation (SMA): a number of buffers are reserved for each class, and the remaining buffers are shared among all classes.

5. Combination of SMXQ and SMA: buffers are reserved for each class, but the total number of buffers used by each class at any time is also limited. This method was used in the well-known ARPANET.

6. Structured Buffer Pool (SBP): class i packets may use all buffers available to class $i-1$ packets, plus a few more. For a single FCFS server, this is equivalent to creating *nested thresholds* on a single queue, that is, allowing class i packets to join the queue if the total queue fill does not exceed threshold i. This technique (with two classes) combined with SMXQ was used in Canada's Datapac network [Spro81].

We note that each of these methods is applicable to priority discarding if more resources are allocated for higher priority packets. Furthermore, in the discussion of the SBP method, the authors provide an important example of how priorities may be assigned. In the SBP method, priorities are determined by the number of hops a packet has traversed since entering the network, reflecting the greater network investment in the packet.

However, we also note that in all these methods, only *arriving packets* can be discarded. Our intuition leads us to believe that better results could be obtained if queued packets could also be discarded. Finally, we note that the analyses for these methods have tended to focus on

throughput optimization, deadlock prevention and fairness, with a number of servers. Consequently, the results are not directly applicable to the problem of minimizing the effect of discarding with a single server.

The studies in [Lam79, Lam80] analyze congestion control via "input buffer limits." This is a two-priority, SMXQ system with the limit for input packets (those just entering the network) *less than* the total number of buffers and the limit for transit packets *equal to* the total number of buffers. The general goal is maximization of network throughput, and analytical results are obtained for the discarding probability for each of the two priorities for a single node. Some results are also obtained for networks (as opposed to isolated nodes) and validated with simulations. The authors also provide a "rule of thumb" for good values of the input buffer limit.

Limitations of these studies are that they focus on maximizing throughput as the objective; they use an M/M/m queuing analysis; they deal with only two priorities; they incorporate multiple servers; and they provide no formal optimization for the input buffer limits.

The studies in [Kamo80b, Kamo81] analyze a a two-priority structured buffer pool, a model which is very similar to the one in [Lam79]. They also derive expressions for discarding probability for each priority for a single node and obtain some network results for a specific class of networks. They find a throughput-optimal input threshold only by plotting throughput *vs.* load for various values of input threshold, and observe that the optimal value is a function of load.

Again, throughput is the objective, M/M/m analysis is used, only two priorities are treated, and no analytical optimization is attempted.

Another application of priority packet discarding is treated in [Luan88a, Luan88b]. Here the concern is with bandwidth management, that is, controlling a customer's input to protect a

network from congestion, while at the same time allowing idle network resources to be used in the absence of congestion. This is accomplished by creating two packet priorities at the network input ("negotiated" and "excess"), and discarding excess packets first in response to network congestion. The authors are (again) primarily concerned with throughput when a window-based flow control mechanism is operating end-to-end, and the focus is on finding appropriate window sizes. A simple nested threshold discarding policy is used only to illustrate how window sizes can be chosen. The present research would complement these studies by focusing on optimizing the discarding policy itself.

A new twist on two-priority nested threshold discarding is presented in [Li88], in which a hysteresis loop is introduced in the (single) threshold. The author shows that if the processes are Markovian birth-death (M/M/1/K) or quasi-birth-death, closed form solutions for the queue length distribution (and thus discarding probability for each priority) can be obtained. Results for packet discarding show that (predictably) high-priority packet loss decreases at the expense of low-priority packet loss as the threshold is decreased. However, there is no indication given of how to evaluate the relative weight of low and high priority losses so that an optimal value of threshold can be obtained. Thus this paper suffers from many of the same inadequacies (with respect to the present problem) as those previously discussed.

So far, the studies have been concerned only with discarding arriving packets. Discarding previously queued packets is treated much less often, so we consider the example in [Clar86], even though it differs in many ways from the problem at hand. The context is minimizing delay for time-critical information. Priority is based on time of arrival with most recent packets receiving the highest priority, so there are no predetermined priority classes as such. The analysis model differs from previous studies in that it is a discrete-time model with batch arrivals. This model has particular relevance to systems with fixed packet lengths (hence service intervals), and we will use a similar model for our analysis in later chapters. A

significant constraint of this study is that it includes only work conserving disciplines, *i.e.*, those which discard packets only when all buffer slots are full (and for which the server is never idle if there are packets waiting). Since there is nothing to design or optimize in such a system, the study is restricted to finding waiting time distributions.

1.2.4.3 B-ISDN/ATM Studies

It has been decided within international standards bodies [CCIT89] that Asynchronous Transfer Mode (ATM), a statistically multiplexed, fixed packet length transfer mechanism, will be the preferred mode of operation for Broadband ISDN (B-ISDN). Only a few papers have addressed the issue of congestion control in such a network.

The authors in [Gers89] propose and analyze a combination of resource reservation and source traffic throttling for congestion control. This differs from the bandwidth management proposals in that, although excess traffic is defined similarly, it is throttled at the source rather than discarded within the network when congestion occurs. As such, the technique is similar to the "reduce packet generation rate" overload control option discussed above and the analysis has little bearing on the packet discarding problem at hand. However, the authors do give a good explanation of the need within ATM networks for the general type of overload control considered in this research.

Another general discussion of congestion control for ATM networks is provided in [Wood88]. Many options are discussed, including packet discarding for bandwidth management, and the relationship between load management and overload control is recognized. However, no analysis is included.

Finally, two classes of ATM traffic are proposed in [Ohni88], a delay-sensitive class and a loss-sensitive class. The authors provide simulation results comparing several control mechanisms involving both input (discarding) and output (service) priorities. The input

priority scheme is especially interesting because it allows for the discarding of queued low priority (delay-sensitive) packets to allow admission of arriving (loss-sensitive) high-priority packets. This is again a work conserving discipline. Results are similar to those in [Li88]: the discard probability of the loss-sensitive class is significantly lowered, at the expense of higher discard probability for the delay-sensitive class. However, no analysis or optimization is attempted.

1.2.4.4 Packet Speech and Video

We conclude by considering several papers which have proposed and analyzed priority packet discarding specifically for speech or video signals.

A pioneering study was done in [Bial80], in which embedded Linear Predictive Coding (LPC) was used in conjunction with speech silence elimination (speech detection). The coded LPC frames were divided into separate packets of different priorities, the highest priority for the essential bits and several lower priorities for enhancement bits. A single priority threshold was established at each network node, so that packets of priority less than the threshold would be discarded on arrival. Note that this mechanism is similar to nested thresholds, yet quite distinct in that there are multiple priorities but only a single threshold. An interesting feature of the scheme is that the threshold is dynamically adapted in an attempt to maximize link utilization while keeping queue length below some limit. Note that maximizing rate roughly corresponds to minimizing speech distortion, so an attempt is being made to minimize the effect of the discarding subject to keeping within the queue limit.

A simulation study showed that the proposed adaptation performed reasonably well with the node functioning autonomously. The authors then proposed two end-to-end (receiver to transmitter) feedback schemes in an attempt to improve overall network utilization. This was successful, but we doubt that such feedback will be effective with the greater dynamics of

future IPNs.

The study in [Yin87b] concerned a two-priority, threshold-based packet voice system. The priorities for speech packets were determined by using embedded coding, even-odd sample numbering (with one arbitrarily given the higher priority), or from speech activity detector thresholds (active speech and semi-silence). The thresholds were either on number of active speakers or on buffer content (a simple nested threshold system). A fluid approximation analysis was carried out, yielding discarding probability and mean waiting time. However, the emphasis here was on comparing the various schemes in terms of mean waiting time, keeping discarding probability approximately the same. No attempt was made to characterize the effect of packet discarding on signal quality and optimize thresholds on that basis.

The results showed that all of the priority packet discarding systems performed better than a system without control and that the system using buffer content thresholds and embedded or even-odd sample priorities performed best. From informal subjective listening tests, the system with embedded coding performed best in terms of maximum percent packet loss for "acceptable" quality, followed by the even-odd sample priority, active/semi-silence priority, and no priorities.

Several papers discuss an embedded coding, priority voice packetization scheme in which the overload control is accomplished by shortening packets [Muis86, Srir88, Srir90a, Srir90b]. These results are interesting even in a packet discarding context since many of the results would remain valid if the embedded bits were grouped into separate packets and packet discarding was used. A system of nested thresholds is used, with the thresholds determined from voice traffic only in [Muis86, Srir88] and from a combined metric derived from voice queue length and data queue length in [Srir90a, Srir90b]. The latter two papers incorporate a priority servicing algorithm in addition to the packet shortening for congestion control.

In addition to showing that mean delay is reduced with the priority control [Muis86], it is also shown that the control is effective since it tends to reduce the burstiness caused by the correlations in the superposed voice packet streams [Srir88]. In the latter paper, it is also shown that quality (mean received bits per sample) decreases as the thresholds are decreased. However, the positive side of this effect (increased capability of carrying other data traffic) is not considered, nor is any way given for combining the two performance measures to determine optimal threshold values. In [Srir90a], the priority discarding and priority service algorithms are detailed, along with simulation results showing that some performance measures are highly dependent on system parameters, while others are not. However, no optimization is attempted. These results are supplemented in [Srir90b] with subjective results (including temporal variations in received bits per sample) indicating the superiority of the priority discarding technique with embedded coding relative to a system in which entire speech segments are discarded.

One final paper has treated priority packet discarding in the context of packet video [Nomu88]. The two priorities are based on discrete cosine transform coding of the video signal. The results show improved distortion *vs.* total packet loss rate performance for the priority discarding system. However, the discarding mechanism is not detailed and the results seem to be based on simulation experiments.

1.3 Dissertation Overview

1.3.1 Organization

This dissertation is organized into five chapters, including this introductory chapter. For each chapter, we provide a table of contents, list of figures, and list of tables immediately preceding the chapter and a list of references at the end of the chapter. Chapters 2 through 4 also include a summary of the results obtained in the chapter. We have gathered together all

of these summaries below. These summaries, together with the discussion of system design implications in the final chapter, provide a succinct encapsulation of the work and its significance.

1.3.2 Summary of Results

1.3.2.1 Chapter 1: Introduction

The new result in this chapter is the development of the example IPN system model described in section 1.2.3. This new system model is made possible by the assumed existence of a priority packet discarding capability in the network, and the model exhibits a number of significant system advantages as a result.

1.3.2.2 Chapter 2: Optimal Priority Packet Discarding: Preliminaries

In Chapter 2, we deal with some preliminaries to our actual analysis of optimal priority packet discarding systems. We obtain the following new results.

1. We develop a general model for the analysis of priority packet discarding algorithms, characterized by three size dimensions: the queue capacity N, the maximum number of arrivals in a fixed service interval L, and the number of packet delivery priorities D. The model dimensions allow one to compare the "size" of different systems. The model is particularly well-suited to future broadband packet switching techniques such as the Asynchronous Transfer Mode (ATM) recently adopted for Broadband ISDN.

2. We develop a classification system for discarding algorithms consistent with the general analysis model. The classification is in two dimensions: control or discarding intervals (either arrival intervals or fixed service intervals) and discarding set (either arriving packets only or any queued packets). This classification allows one to compare the discarding control structures of different systems which may have the same "size" dimensions.

3. We specify an appropriate optimality criterion (expected discarding cost) and show how this criterion is a generalization of the throughput criterion which has been used in many previous studies.

4. We show how stochastic dynamic programming can be used to analyze priority packet discarding systems.

1.3.2.3 Chapter 3: Optimal A/A Packet Discarding

We deal in Chapter 3 with the A/A priority discarding structural classification: systems that use Arrival intervals as the control intervals and that are allowed to discard only Arriving packets. The following list itemizes the new contributions presented in Chapter 3 and at the same time identifies some of the remaining issues.

1. We develop a 2-dimensional analysis model for queue fill A/A systems, that is, A/A systems for which the queue state information is restricted to total queue fill.

2. We derive sufficient conditions for the irreducibility of any policy under this model. By the results of Chapter 2, restricting our search to optimal policies satisfying these conditions guarantees that a stationary policy will be optimal.

3. We derive a simple recursive method for obtaining the state distribution and expected discarding cost for a given discarding policy under this model for any model dimensions and parameters. The computational complexity of this algorithm is linear in S, the number of states in a particular model. A more general solution for the state distribution would have complexity roughly proportional to S^3.

4. We derive a simple recursive method for obtaining the relative values and expected discarding cost for a given discarding policy for any model dimensions and parameters. These values then allow one to check for optimality for the given policy. The method could also be used as the value determination step in the policy improvement algorithm

for finding the optimal queue fill A/A policy for a given set of model dimensions and parameters. The recursive value determination algorithm again has complexity which is linear in S, comparing very favorably with the S^3 complexity of the more general solution method.

5. We hypothesize that all optimal policies are nested threshold policies, calculate the number of such policies, and sketch a possible approach to proving the hypothesis. The actual proof is left as an open item.

6. We demonstrate that the normalized expected discarding cost per arrival for the *default policy* of discarding packets if and only if the queue is full is independent of any priority information, as expected. We also derive a closed-form expression for this normalized cost for the default policy for the particular case of $L=2$. Since this default policy is almost universally applied today, this result gives a performance baseline for priority packet discarding systems.

7. We demonstrate that the default policy is an optimal queue fill A/A policy for $(N,L,D) = (3,2,2)$ only for low loads or small differences in the discarding costs. Again, since the default policy is almost universally used today, this result indicates that continuing to use the default policy as an overload control mechanism will often result in sub-optimal performance. Demonstrating this result in general is an open item.

8. We demonstrate that the performance advantage of optimal A/A discarding relative to default discarding is significant for large variations in model parameters. This result indicates that optimal discarding policies should be of practical as well as theoretical interest.

9. We demonstrate that there is always a performance advantage of using a larger capacity queue (larger N), but that this advantage diminishes with increasing load. So, for

heavily overloaded systems, increasing the queue capacity is of little value. However, at low overloads the default policy for a larger queue capacity may perform better than the optimal queue fill A/A discarding policy. In some middle region of overload (perhaps quite narrow), the system designer can choose between a larger system with the default discarding policy or a smaller system with an optimal queue fill A/A policy and get approximately the same performance.

1.3.2.4 Chapter 4: Optimal FS/Q Packet Discarding

In Chapter 4, we treat a different class of packet discarding systems: those that use the Fixed Service intervals as the control intervals and are allowed to discard any Queued packets, or FS/Q systems. The following list summarizes the new contributions presented in Chapter 4, along with remaining open issues.

1. We develop a specific analysis model for FS/Q discarding systems. For the two possible discarding instants for this model, we show how to modify the analysis for systems discarding after the service completion to obtain an analysis for systems discarding before the service completion.

2. We derive sufficient conditions for the irreducibility of any FS/Q policy. Restricting our search to such policies guarantees the existence of an optimal stationary policy.

3. We show that the policy universe may be reduced by considering only candidate policies, but that even this reduced universe is generally quite large for this model.

4. We identify the form of every distinct transition vector in the transition matrix of any FS/Q policy.

5. We derive a computationally efficient method for value determination for FS/Q systems, using the distinct transition vectors. The computational advantage of this method over the general method is substantial. This method can be used to speed up the policy

improvement algorithm for finding an optimal FS/Q policy for a given set of parameters.

6. We demonstrate that the performance of optimal FS/Q discarding policies is similar to the performance of the optimal queue fill A/A policies of Chapter 3, both of which significantly out-perform the default policy. We also hypothesize that A/Q policy structures should perform better than these and that FS/A policies should perform worse, but leave verification as an open issue.

7. We demonstrate that the performance improvement of increasing queue capacity diminishes significantly with increasing load, just as for A/A systems.

8. We perform a general analysis of FS/Q systems restricted to retain at most one packet. We identify a very small family of optimal policies, derive closed-form sufficient conditions for the optimality of each policy and closed-form expressions for the average discarding cost, and prove that the family is complete, that is, for any set of arrival statistics and discarding costs, one of the family members must be optimal.

9. We demonstrate that for systems with just two delivery priorities, it may be that a single specific policy is either optimal or very nearly optimal for every set of arrival statistics. The validity of this phenomenon for systems with more delivery priorities is left as an open issue.

1.3.2.5 Chapter 5: Conclusions and Future Directions

In Chapter 5, we discuss the implications of our results to the system design of Integrated Packet Networks and other systems. We also identify open questions from this research and indicate how some of the analysis techniques could be extended.

1.4 References

[Bial80] T. Bially, B. Gold, S. Seneff, "A Technique for Adaptive Flow Control in Integrated Packet Networks," *IEEE Transactions on Communications*, Vol. COM-

28, No. 3, March 1980.

[CCIT89] CCITT Recommendation I.121, "Broadband Aspects of ISDN," Blue Book, Geneva, Switzerland, June 1989.

[Chen88] T. Chen and D. G. Messerschmitt, "Integrated Voice/Data Switching," *IEEE Communications Magazine,* Vol. 26, No. 6, June 1988.

[Clar86] L. P. Clare and I. Rubin, "Preemptive Buffering Disciplines for Time-Critical Sensor Communications," Proceedings of the IEEE International Conference on Communications, Toronto, Canada, June 1986.

[Cohe80] D. Cohen, "On Packet Speech Communication," Proceedings of the Fifth International Conference on Computer Communication, Atlanta, October 1980.

[DaSi89] L. A. DaSilva, D. W. Petr, V. S. Frost, "A Class-Oriented Replacement Technique for Lost Speech Packets," *IEEE Transactions on Acoustics, Speech and Signal Processing,* Vol. ASSP-37, No. 10, October 1989.

[Fros86] V. S. Frost, E. M. Friedman, G. J. Minden, "Multirate Voice Coding for Load Control on CSMA/CD Local Computer Networks," *Computer Networks and ISDN Systems,* Vol. 11, 1986.

[Gerl80] M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey," *IEEE Transactions on Communications,* Vol. COM-28, No. 4, April 1980.

[Gers89] A. Gersht and K. J. Lee, "A Congestion Control Framework for ATM Networks," Proceedings of the IEEE Infocom Conference, Ottawa, Canada, April 1989.

[Gold77] B. Gold, "Digital Speech Networks," *Proceedings of the IEEE,* Vol. 65, No. 12, December 1977.

[Good80] D. J. Goodman, "Embedded DPCM for Variable Bit Rate Transmission," *IEEE Transactions on Communications,* Vol. COM-28, No. 7, July 1980.

[Good86] D. J. Goodman, "Waveform Substitution Techniques for Recovering Missing Speech Segments in Packet Voice Communications," *IEEE Transactions on Acoustics, Speech and Signal Processing,* Vol. ASSP-34, No. 6, December 1986.

[Hand89] R. Handel, "Evolution of ISDN Towards Broadband ISDN," *IEEE Network Magazine,* Vol. 3, No. 1, January, 1989.

[Heff86] H. Heffes and D. M. Lucantoni, "A Markov-Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performance," *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 6, September 1986.

[Jaya81] N. S. Jayant and S. W. Christensen, "Effects of Packet Losses in Waveform Coded Speech and Improvements Due to an Odd-Even Sample-Interpolation Procedure," *IEEE Transactions on Communications*, Vol. COM-29, No. 2, February 1981.

[Kahn71] R. E. Kahn and W. R. Crowther, "A Study of the ARPA Computer Network Design and Performance," Bolt Beranek and Newman, Inc., Tech. Rep. 2161, August 1971.

[Kamo80a] F. Kamoun and L. Kleinrock, "Analysis of Shared Finite Storage in a Computer Network Node Environment Under General Traffic Conditions," *IEEE Transactions on Communications*, Vol. COM-28, No. 7, July 1980.

[Kamo80b] F. Kamoun, A. Belguith, and J. L. Grange, "Congestion Control with a Buffer Management Strategy Based on Traffic Priorities," Proceedings of the IEEE International Conference on Computer Communications, Atlanta, October 1980.

[Kamo81] F. Kamoun, "A Drop and Throttle Flow Control Policy for Computer Networks," *IEEE Transactions on Communications*, Vol. COM-29, No. 4, April 1981.

[Lam79] S. S. Lam and M. Reiser, "Congestion Control of Store-and-Forward Networks by Input Buffer Limits -- An Analysis," *IEEE Transactions on Communications*, Vol. COM-27, No. 1, January 1979.

[Lam80] S. S. Lam and Y. L. Lien, "An Experimental Study of the Congestion Control of Packet Communications Networks," Proceedings of the IEEE International Conference on Computer Communications, Atlanta, October 1980.

[Li88] S. Q. Li, "Overload Control in a Finite Message Storage Buffer," Proceedings of the IEEE Infocom Conference, New Orleans, March 1988.

[Luan88a] D. T. Luan and D. M. Lucantoni, "Throughput Analysis of a Window-Based Flow Control Subject to Bandwidth Management," Proceedings of the IEEE Infocom Conference, New Orleans, March 1988.

[Luan88b] D. T. Luan and D. M. Lucantoni, "Throughput Analysis of an Adaptive Window-Based Flow Control Subject to Bandwidth Management," Proceedings of the 12th International Teletraffic Congress, Torino, Italy, June 1988.

[Mats87] N. Matsuo, M. Yuito, Y. Tokunga, "Packet Interleaving for Reducing Speech Quality Degradation in Packet Voice Communications," Proceedings of the IEEE

Globecom Conference, December, 1987.

[Minz89] S. E. Minzer, "Broadband ISDN and Asynchronous Transfer Mode (ATM)," *IEEE Communications Magazine,* Vol. 27, No. 9, September 1989.

[Mont83] W. A. Montgomery, "Techniques for Packet Voice Synchronization," *IEEE Journal on Selected Areas in Communications,* Vol. SAC-1, No. 6, December 1983.

[Muis86] R. W. Muise, T. J. Schoenfeld, G. H. Zimmerman, "Experiments with Wideband Packet Technology," Proceedings of the 1986 International Zurich Seminar on Digital Communications, March 1986.

[Mura89] M. Murata, Y. Oie, T. Suda, and H. Miyahara, "Analysis of a Discrete-Time Single-Server Queue with Bursty Inputs for Traffic Control in ATM Networks," Proceedings of the IEEE Globecom Conference, Dallas, November 1989.

[Nomu88] M. Nomura, T. Fujii and N. Ohta, "Layered Packet-Loss Protection for Variable Rate Video Coding Using DCT," Second International Packet Video Workshop, Torino, Italy, September 1988.

[Ohni88] H. Ohnishi, T. Okada and K. Nugushi, "Flow Control Schemes and Delay-Loss Tradeoff in ATM Networks," *IEEE Journal on Selected Areas in Communications,* Vol. 6, No. 9, December 1988.

[Petr89] D. W. Petr, L. A. DaSilva, V. S. Frost, "Priority Discarding of Speech in Integrated Packet Networks," *IEEE Journal on Selected Areas in Communications,* Vol. 7, No. 5, June 1989.

[Pouz81] L. Pouzin, "Methods, Tools, and Observations on Flow Control in Packet-Switched Data Networks," *IEEE Transactions on Communications,* Vol. COM-29, No. 4, April 1981.

[Ride89] M. J. Rider, "Protocols for ATM Access Networks," *IEEE Network Magazine,* Vol. 3, No. 1, January, 1989.

[Spro81] D. E. Sproule and F. Mellor, "Routing, Flow, and Congestion Control in the Datapac Network," *IEEE Transactions on Communications,* Vol. COM-29, No. 4, April 1981.

[Srir86] K. Sriram and W. Whitt, "Characterizing Superposition Arrival Processes in Packet Multiplexers for Voice and Data," *IEEE Journal on Selected Areas in Communications,* Vol. SAC-4, No. 6, September 1986.

[Srir88] K. Sriram and D. M. Lucantoni, "Traffic Smoothing Effects of Bit Dropping in a Packet Voice Multiplexer," Proceedings of the IEEE Infocom Conference, New Orleans, March 1988.

[Srir90a] K. Sriram, "Dynamic Bandwidth Allocation and Congestion Control Schemes for Voice and Data Multiplexing in Wideband Packet Technology," Submitted to IEEE Supercomm/ICC Conference, Atlanta, April 1990.

[Srir90b] K. Sriram, G. A. Mariano, D. O. Bowker, and A. U. Mac Rae, "An Integrated Access Terminal for Wideband Packet Networking: Design and Performance Overview." Submitted to 1990 International Switching Symposium, Stockholm, Sweden, June 1990.

[Turn86] J. S. Turner, "Design of an Integrated Services *Packet* Network," *IEEE Journal on Selected Areas in Communications,* Vol. SAC-4, No. 8, November 1986.

[Wood88] G. M. Woodruff, R. G. H. Rogers, and P. S. Richards, "A Congestion Control Framework for High-Speed Integrated Packetized Transport," Proceedings of the IEEE Globecom Conference, Hollywood, Florida, November 1988.

[Wood90] G. M. Woodruff and R. Kositpaiboon, "Multimedia Traffic Management Principles for Guaranteed ATM Network Performance," to appear in *IEEE Journal on Selected Areas in Communications,* April 1990.

[Yin87a] N. Yin, T. E. Stern and S. Q. Li, "Performance Analysis of a Priority-Oriented Packet Voice System," Proceedings of the IEEE Infocom Conference, San Francisco, March 1987.

[Yin87b] N. Yin, S. Q. Li and T. E. Stern, "Congestion Control for Packet Voice by Selective Packet Discarding," Proceedings of the IEEE Globecom Conference, December, 1987.

Chapter 2

Optimal Priority Packet Discarding:

Preliminaries

CONTENTS

Chapter 2

LIST OF FIGURES

LIST OF TABLES

Chapter 2

Optimal Priority Packet Discarding: Preliminaries

In this chapter we lay the groundwork for the analysis (in succeeding chapters) of priority packet discarding systems. We begin by developing a general model (section 2.1) for priority packet discarding systems which will be used in the subsequent analyses. The next section (2.2) presents a classification system for discarding algorithms which can be described by the model of section 2.1. Section 2.3 discusses the optimality criterion used for the analyses in subsequent chapters. Section 2.4 introduces dynamic programming, an analytical tool which is applicable to our study of optimality in priority discarding systems. The final section (2.5) summarizes the new results in this chapter.

2.1 A General Analysis Model

Mathematical modeling is the crucial first step in any optimization study, as pointed out in [Luss89]:

> The effective use of optimization techniques requires much more than the application of sophisticated algorithms to well-formulated problems. It involves, as well, the appropriate representation of a real-life situation within a mathematical model.

We thus begin our study with the development of a general model which provides a mathematical description of priority packet discarding systems. The model will be made more specific in later chapters to allow for the analysis of specific priority packet discarding systems. We have specifically tried to avoid "models that only faintly resemble real-life problems" [Luss89]. The model we have developed is particularly well-suited to high-speed IPNs of the future (such as Broadband ISDN), as we will show in the following presentation of the model.

We will also see that the "dimensions" of the model allow us to compare systems of the same "size" but different discarding structures.

2.1.1 Scope and Assumptions

We provide here a list of the characteristics of a general model for the analysis of priority packet discarding that bound its scope and define its applicability.

1. *Single Queue:* We will focus the research on a single queuing system with finite buffer space. The number of queuing positions (excluding the server) will be designated N. Analysis for entire networks is left for future research.

2. *Output Queuing:* We will assume that all node queuing occurs while waiting for transmission. That is, we will ignore secondary queuing in a network node, *e.g.*, queuing on the input side of a switch.

3. *Single Server:* We will assume that each transmission link has a queue dedicated to it. This one-to-one association between queue and transmission link differs from many previous studies [Lam79, Gerl80, Kam81] concerned with a pool of buffers shared by all links in a node.

4. *FCFS Service:* We are concerned primarily with discarding priority, as opposed to service priority, and will therefore assume a First-Come-First-Served (FCFS) queuing service discipline. That is, a packet must join the queue at the end, and no packet is allowed to exchange places in the queue with any other packet. This service discipline is followed in most connection-oriented networks to avoid packet re-ordering within the network. Of course, packets may be discarded, in which case they are not served at all.

5. *Fixed Service Intervals:* We will assume that a fixed amount of time is required to move a packet onto the transmission link. Since transmission links almost always have a fixed bit rate, this amounts to an assumption of fixed packet lengths. This assumption is

justified since high capacity packet switching systems are more easily implemented if the packet size is fixed. For this reason, international standards bodies have recently agreed [CCIT89] that the basic transport unit for Broadband ISDN (B-ISDN) will be of fixed length. The assumption of fixed packet lengths precludes the use of the M/M/1 continuous-time analysis method and leads naturally to discrete time analysis methods.

We also assume that these fixed service intervals are kept synchronized. That is, if the queue is ever empty when the server is ready for another packet, a "null" packet of the same length as all other packets will be transmitted.

6. *Bounded, Independent Arrivals:* The maximum number of arrivals during a single service interval is assumed to be limited to at most L. For a switching system in which all input and output links have the same transmission rate, this amounts to a maximum of L input links which can route to a given output. Furthermore, the intervals are assumed to be statistically independent.

7. *Discarding Priorities and Costs:* We will assume that each arrival has associated with it a discarding priority, of which there are D possible values, coded in the packet header. Furthermore, we assume the existence of a *discarding cost function,* a function of delivery priority which indicates the cost associated with discarding a packet of that priority.

8. *Steady-State Overload Analysis:* We will assume that for the period being analyzed the condition $\rho > 1$ holds on average, *i.e.,* that the system is overloaded as in equation (1.1). We will further assume that this overload period persists long enough *a*) to require packet discarding due to finite buffer size and *b*) for steady-state analysis to accurately

* The B-ISDN name for this basic unit is a *cell,* but we will use the more generic term *packet* in this research.

predict queue behavior. The steady-state assumption also implies that arrival statistics, parameters and the discarding cost function, all of which are assumed to be known, will also be considered time-invariant during the overload period. Extensions of the research to time-varying analysis are discussed in Chapter 5.

9. *Control Based on State:* In keeping with the principle of node-autonomous control, discard decisions are assumed to be made on the basis of instantaneous node state (as well as the discarding cost function mentioned above). Of course, as much of the past node history can be incorporated into this state as desired. In this dissertation, we will assume that node state information is restricted to queue state information (which includes some indirect information on past arrival and discarding history). Basing control decisions on queue state is more restrictive than basing them on a relation such as (1.2) in that projections of queue occupancy are not allowed. On the other hand, it is also more general in that all queue state information (not just occupancy) can be used.

10. *Non-Preemptive Discarding:* The packet which has already begun receiving service will not be be discarded. This assumption extends to "null" packets that are being transmitted to maintain service interval synchronization.

2.1.2 Model Dimensions: (N,L,D)

From the foregoing description of the model, we note that one way to characterize a particular system is by the three "dimensions" listed:

- N: the number of positions (excluding server) in the queue
- L: the maximum number of arrivals in a fixed service interval
- D: the number of delivery priorities

We will use the triple (N,L,D) as such a characterization throughout the remainder of this

dissertation to compare different systems. Such a description is shown graphically in Figure 2-1.

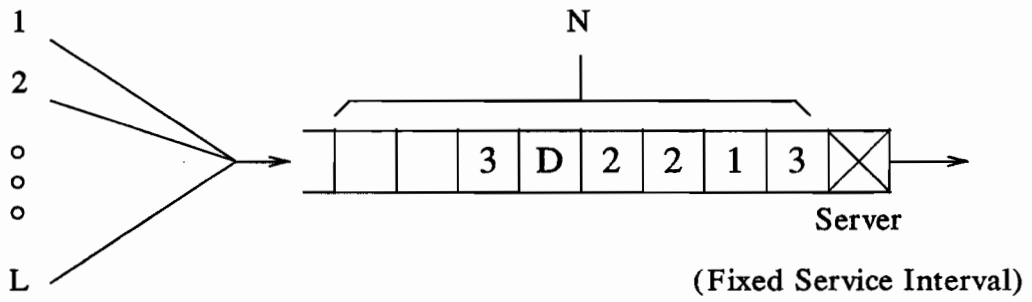


Figure 2-1. Priority Packet Discarding Model Summary

Priority discarding requires $D \geq 2$, a system in overload ($\rho > 1$) requires $L \geq 2$, and any decision to retain or discard requires at least one non-serving buffer position ($N \geq 1$), so the smallest non-trivial priority system has $(N,L,D) = (1,2,2)$.

2.2 Classification of Discarding Algorithms

The dimensions (N,L,D) of the previous section constitute a *quantitative* description of a system; that is, a description of its "size." In this section we develop a classification system which describes the *structure* of a discarding algorithm. Together, the model dimensions and structural classification provide a complete description of any priority packet discarding system. The two descriptions provide a useful means of comparison for systems which differ in various ways. For example, we may wish to compare the performance of two systems of the same structure, one of which has dimensions (N_1,L_1,D_1) and the other (N_2,L_2,D_2). Alternatively, we may wish to discover the performance difference between systems with different structural classifications, but the same dimensions. Such comparisons are a critical means of evaluating system design.

The two discriminating features we will use are the specification of *control intervals* and *discarding sets*. We will then designate a particular class of algorithm by a code of the form X/Y, where X indicates the control interval specification and Y indicates the discarding set specification.

2.2.1 Control Intervals

A fundamental feature of any discarding algorithm is the frequency with which discarding (control) decisions are made. We will refer to the intervals between discarding decisions as *control intervals* and base one part of our classification on how these intervals are specified.

The control interval influences the partitioning of the positions in the queue in the following way. We will wish to exercise complete control over any discarding from the queue in the sense that we do not want any packet to be discarded unless we have deliberately *decided* to discard it. This implies that some queue positions must be *reserved* for arrivals between control intervals. That is, every discarding decision must ensure that these reserved queue positions are empty after the control decision is implemented. In other words, these reserved queue positions can only be *temporarily* occupied, and so may be viewed as less "valuable" than the remainder of the queue positions, even though they must be included in the total queue capacity N. On the other hand, the reserved queue positions allow the discarding algorithm to *defer* discarding decisions until more information (about future arrivals) is available. Because of these conflicting effects, we cannot say unequivocally whether it is desirable to have a small or large number of reserved positions; that determination must be left to analyses of particular cases.

Since we assume that control decisions are based on state information, control intervals must be bounded by state changes. For a queue, changes of state (aside from packet discarding) occur only when there is an arrival or service completion.

2.2.1.1 Arrival Intervals

One possibility is to specify the control interval as the interval between arrivals, resulting in a discarding decision for every arrival. We will designate this control interval specification as A (Arrival).

Although we offer no formal proof, we make the following argument that the arrival interval is the smallest (on average) reasonable control interval for an overloaded system, *i.e.*, one in which the normalized utilization ρ exceeds unity (see the previous section). For such a system, the only interval which could possibly be smaller is the interval between *any* queue state change, including service completions. However, it is intuitive that "non-null" control decisions (those which actually discard a packet or packets) should only be triggered by one or more arrivals, not by departures. That is, if the state of the system after a control decision is σ, then the control decision corresponding to state σ', defined as state σ modified by the removal of a packet due to a service completion, should be (in any reasonable system) the null decision (discard no packets).

Using arrival intervals as the control intervals implies that we need only one reserved queue position. However, we must make a discarding decision every time a packet arrives without the benefit of information about future arrivals.

2.2.1.2 Service Intervals

Another possibility is to specify the control interval as the interval between service completions. By the reasoning above, we would expect only null control decisions if there are no arrivals in such an interval. However, if there are any arrivals in a service interval, these could trigger a non-null decision for that service interval. This specification is particularly appealing for $\rho > 1$, since then (on average) there will be more than one arrival per control decision and thus *fewer* control decisions must be made and implemented. Since we have

restricted ourselves to fixed service intervals, we will designate this control interval specification as FS (Fixed Service).

In this case, we will need L reserved queue positions,* but we can defer our discarding decisions until we have more complete information on arrivals to the queue. That is, if we were to compare systems with the same number of *non-reserved* queue positions, we would expect one using service interval as the control intervals to perform better than one using arrival intervals, all other things being equal. However, this would not be a "fair" comparison since the system using service intervals would have a larger total queue capacity N.

2.2.2 Discarding Sets

The other defining characteristic for discarding algorithms is the set of packets which may be discarded by any given discard decision. Since we restrict our algorithms from discarding the packet currently receiving service, there are two logical possibilities for such a set: *only* those packets which have arrived since the last decision or *any* currently queued (waiting) packets, including the most recent arrivals.

2.2.2.1 Discard Arriving Packets

If the discarding set is limited to arriving packets only, we will assign a code of A (Arriving) for the discarding set. Limiting the discarding set in this way could simplify hardware implementations since control would be limited to *admission* to the queue. That is, once a packet has joined the queue, it is guaranteed to be served. If in addition the control interval is specified as the interval between arrivals, resulting in an A/A class of discarding algorithm, the discarding decision is binary: either discard the (single) packet which has just arrived, or allow it to join the queue. For FS/A discarding algorithms (control interval =

* We will see in Chapter 4 that in some cases only $L-1$ reserved positions are required.

Fixed Service interval and discarding set = Arriving packets), the number of choices for a particular discarding decision is in general 2^a, where a is the number of arrivals since the last discarding decision. This variation in control choices complicates the analysis of FS/A algorithms compared to A/A algorithms. We note that all of the discarding structures listed in Chapter 1 under the discussion of [Gerl80] are specific members of the A/A class.

2.2.2.2 Discard Queued Packets

Although limiting the discarding set to arriving packets simplifies implementation, it is clear that allowing *any* queued packets to be discarded, including the most recent arrivals, affords greater control flexibility. We will designate this discarding set as Q (Queued packets).

To see the potential for performance advantage, we need only consider a general two-priority system in which a burst of low priority packets (some of which will gain access to the queue) is followed by a burst of high priority packets which cannot all be accommodated due to the finite queue size. In this case, any X/A algorithm (where X represents any discarding interval) will be forced to discard only high priority packets during the high priority burst, but the corresponding X/Q algorithm could reach into the queue to discard low priority packets instead, achieving better performance. The price of this performance advantage is complexity, both in implementation (though we make no attempt to justify this claim) and in analysis (as we shall see in subsequent chapters).

We summarize our classification notation in Table 2-1.

TABLE 2-1. Structural Classification of Discarding Algorithms

		Discarding Set	
		Only **Arriving** Packets	Any **Queued** Packets
Discarding Interval	**Arrival** Interval	A/A	A/Q
	Fixed Service Interval	FS/A	FS/Q

2.3 Optimality Criterion: Expected Discarding Cost

In order to analyze, compare and optimize discarding systems, we must define the performance measure. As discussed in Chapter 1, we are concerned here with minimizing the effect of packet discarding for an overloaded system. Hence we will use *expected discarding cost* as our basis for comparison. We will use the following notation to define this measure:

$E[C]$:	expected discarding cost
d:	delivery priority, $d \in \{1,2,...,D\}$
c_d:	cost of discarding priority d packet
p_d:	probability that an arriving packet is of priority d
δ_d:	probability that an arriving packet will be discarded before reaching the server, given that it has priority d

Then the expected discarding cost per arrival is defined as

$$E[C] = \sum_d \delta_d \cdot p_d \cdot c_d \tag{2.1}$$

Clearly we wish to minimize $E[C]$. We note that minimizing $E[C]$, the expected cost *per arrival,* is equivalent to minimizing the expected cost *per service completion,* which is just $\rho \cdot E[C]$, where ρ is again the normalized queue load (arrivals per service interval). We also note that minimizing $E[C]$ can be equivalently viewed as maximizing a normalized *weighted throughput* measure, defined as

$$\frac{\sum_d (1-\delta_d) \cdot p_d \cdot c_d}{\sum_d p_d \cdot c_d} = 1 - \frac{E[C]}{\sum_d p_d \cdot c_d} \tag{2.2}$$

In this context, $(1-\delta_d)$ is the probability that an arriving packet will be served (transmitted) given that it is of priority d, and the cost c_d should be viewed as a reward for transmitting a priority d packet. Thus, the present research can be seen as a generalization of previous research [Lam79, Kamo81, Gerl80, Luan88] concerned with maximizing *unweighted* throughput, *i.e.,* all c_d equal.

In performance comparisons in future chapters, we will refer to the second term on the right-hand side of (2.2) as the *normalized expected cost per arrival,* since the denominator $\sum_d p_d \cdot c_d$ is the expected cost of discarding *every* packet and is thus the maximum possible value of $E[C]$. The use of the normalized form of the expected cost allows us to focus on the relationship between the costs and eliminates unwanted effects due to the absolute cost values. For example, if one system with $D = 2$ has costs $c_1 = 1$ and $c_2 = 2$ and another has $c_1 = 10$ and $c_2 = 20$, but the systems are identical in all other respects, their performance will be identical if we use the normalized expected cost measure.

2.3.1 Discarding Cost Function

We may assume without loss of generality that the delivery priorities are ordered so that the discarding cost function $\{c_d : 1 \le d \le D\}$ is positive and *strictly* increasing;, that is, $c_{d+1} > c_d > 0$ for all $1 \le d \le D-1$.* With this ordering, priority D packets are the most important. Strict monotonicity is not limiting since, if two delivery priorities have the same discarding cost, they are indistinguishable and may be grouped into a single priority.

We now show some examples of how the discarding cost function can be derived for packets under different circumstances. The number of ways of assigning the cost function is virtually unlimited, so these examples should be viewed as purely representative.

In any network, if we are concerned about total *network* throughput regardless of the type of traffic, we might assign a discarding cost function which is monotonically increasing as a function of the number of hops a given packet has already traversed. This is a generalization of the priority assignment for the structured buffer pool (SBP) discussed in Chapter 1 in

* We will sometimes allow d to take the value 0, representing a "null" packet. In these cases, the associated discarding cost is $c_0=0$, and if the function is strictly increasing, it will also be positive.

connection with [Gerl80] in that it relates the value of the packet to the "investment" that the network has made in it.

A related example can be constructed for a network with small packet sizes (such as the "cells" in B-ISDN [CCIT89]). In such a network, endpoints may wish to perform end-to-end error control for computer data on "super-packets" formed from a number of consecutive packets. In such a case, the loss of an individual packet would effectively void the entire super-packet. If the super-packet size is not always the same, we could then reasonably assign to each packet a discarding cost which is proportional to the size of its super-packet.

We have already seen in Chapter 1 many examples of how speech packets may be grouped into different priorities. The assignment of a discarding cost to each priority could then be made based on the expected subjective effect of the loss of a packet of that priority. For example, in [Petr89] speech packets were classified by the underlying speech production model. We showed that these different classes had different levels of maximum packet loss which would result in a subjectively indistinguishable difference from a signal with no packet losses. In this case, the discarding cost for each class could be made inversely proportional to these maximum packet loss rates. Similar techniques could be used for other perceptually received signals such as image and video signals [Nomu88].

2.3.2 Secondary Considerations

We will see in subsequent chapters that there may well be several specific discarding policies for a given system which have identical values of $E[C]$. In such cases it would be possible to distinguish among them by resorting to secondary considerations, such as a desire to minimize the average delay of delivered packets. In this research, however, we will focus exclusively on the minimization of $E[C]$ without any concern for secondary considerations.

2.4 Analysis Method: Stochastic Dynamic Programming

The model described in section 2.1 lends itself to analysis via the discrete-time, sequential decision method of dynamic programming. In this section, we will introduce basic dynamic programming terminology and notation and show how the technique can be applied to the present problem. We will also review some established results of dynamic programming theory which will be useful in subsequent analyses. Material in this section (except for section 2.4.3) is an amalgamation of material from [Howa60, Boud71, Ross83, Kuma86].

2.4.1 Dynamic Programming Tutorial Example

Dynamic programming is a technique for analyzing and optimizing sequential decision processes. We will use an extremely simple example to introduce its essential features. Although the example (borrowed from [Boud71]) is unrelated to packet discarding, we will quickly move to show the applicability to our present problem.

Suppose that a smuggler operates between three countries labeled 1, 2 and 3. If she is in country x (her *current state*), she may transport contraband to any country y (her *decision*), thereby receiving payment of $r(x,y)$ (her *reward*) and finding herself in country y as her next starting point (her *next state*). Table 2-2 shows the rewards $r(x,y)$ for this example.

TABLE 2-2. Smuggler's Rewards

		Decision y		
		1	2	3
State x	1	0	10	7
	2	1	0	4
	3	8	2	0

The problem is to find the sequence of K decisions from starting state x which *optimizes* (that is, maximizes) the smuggler's *total return*. We will denote this *optimal* total return as v_x^K. The length of the sequence of decisions K is called the *horizon* of the problem.

The problem can be solved in a recursive manner as follows. We first consider the case of $K=1$, where the maximum reward is clearly:

$$v_x^1 = \max_y [r(x,y)] \tag{2.3}$$

and the optimal decision y_x^1 is the one which yields this maximum, that is:

$$y_x^1 = arg \max_y [r(x,y)]. \tag{2.4}$$

For example, if the smuggler starts in country 1, we see directly from Table 2-2 that the best *single-step* decision is to travel to country 2, obtaining reward of 10. This is illustrated in Figure 2-2 below for all initial states, where the boxes represent states x, arrows represent optimal decisions y_x^1, and the number in each box represents the optimal return v_x^K for the given starting state x.

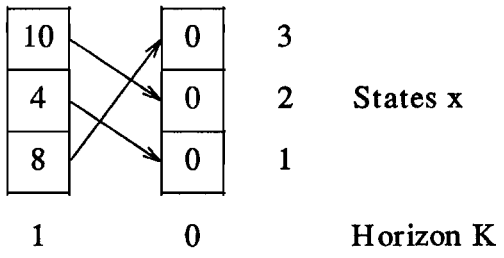


Figure 2-2. Smuggler Solution For K=1

Now we consider the case of $K=2$. We immediately note that, *after the first decision is made,* we are faced with the identical $K=1$ problem that we just solved! So if the first decision (of two) is to travel to country 1, we know from the solution of the $K=1$ problem (Figure 2-2) that the best second decision (of two) is to travel from there to country 2. Since we wish to optimize the *total* two-step reward, our solution will then be:

$$v_x^2 = \max_y [r(x,y) + v_x^1]. \tag{2.5}$$

For example, if the smuggler starts in country 1, we see from Table 2-2 and Figure 2-2 that the best *two-step* decision is to travel first to country 3 (obtaining *present* reward of 7) and then to country 1 (obtaining *future* reward of 8) for a total reward of 15. Note for this example that the optimal smuggler sacrifices present reward (by going to country 3 instead of 1) because the future reward will more than offset his present sacrifice (the total reward for the path 1→2→3 is only 10+4 = 14). The solution for every initial state for the $K=2$ problem is illustrated in Figure 2-3 below.

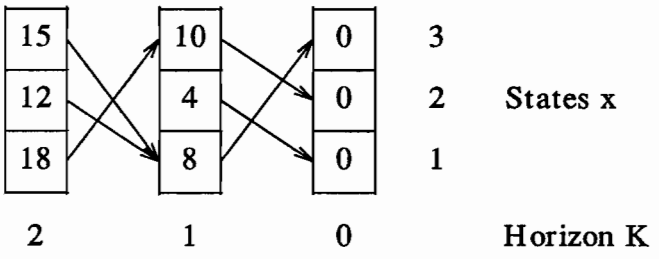


Figure 2-3. Smuggler Solution For K=2

Carrying on the recursion, we see that for general K we have:

$$v_x^K = \max_y [r(x,y) + v_y^{K-1}]. \tag{2.6}$$

The general recursive relation (2.6) is known as the *optimality equation,* and is the foundation of the dynamic programming method. It essentially says that to find an *optimal* solution for K steps, we need consider only the *optimal* solutions for $K-1$ steps. So the solution to our problem for any K is solved by working *backwards,* finding first the best one-step solution, then the best two-step solution, etc. The results of this recursion for $K=5$ are shown in Figure 2-4 below.

We see from this simple example several important aspects of any dynamic programming problem. First, we note that time is discrete and the system is described by a state at any

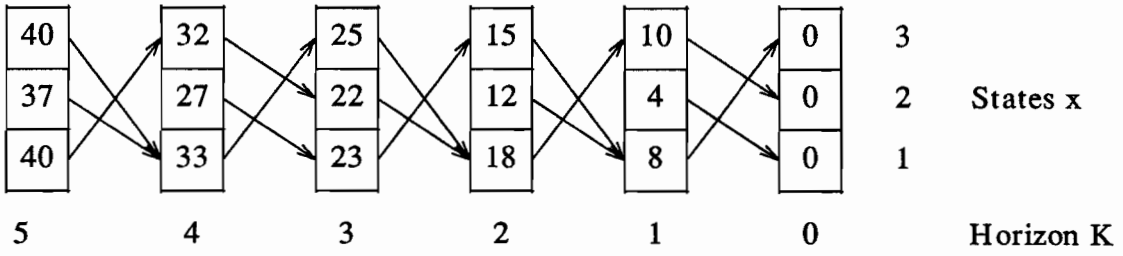


Figure 2-4. Smuggler Solution For K=5

discrete time instant. Also, we see that control decisions are a function of state, and possibly of time as well. We note that the present reward and the next state are both functions of the current state and the control decision. We also see that the effect of future rewards, which are a function of the next state, plays a major part in the decisions. Finally, note that in solving a problem for a specific horizon K and initial state x, we have in fact solved the much larger problem of determining the optimal sequence of decisions for *any* horizon k with $1 \leq k \leq K$ and for *any* initial state. This technique is known as invariant embedding of the specific problem [Boud71].

However, this example is overly simplistic in two important ways. First, the relationship between the decision and the next state is too direct: the decision *is* the next state. In a more general system, the next state will be a function of three factors: the current state, the control decision, and some external input or inputs. Secondly, there is no provision for chance or uncertainty in this example, which is especially unrealistic in the life of a smuggler! This leads us to a discussion of stochastic dynamic programming [Ross83, Kuma86].

2.4.2 Stochastic Dynamic Programming

We will now consider the class of problems of the following general description. Suppose that time is again discrete and numbered backwards from some end time 0 to a starting time K. Suppose also that the system can be described at each time instant k by a state σ^k with $\sigma^k \in \{1,2,\ldots,S\}$. At each time instant, we are allowed to make a control decision $g_{\sigma}{}^{k}$ which

may depend on both the current state σ and the time instant k. We will refer to any set of control decisions for all states $\{g_\sigma{}^k : 1 \le \sigma \le S\}$ as a *control set* at time k and denote it G^k.

We will find it more natural in our application to think in terms of penalties or costs associated with control decisions, rather than rewards as in the smuggler example. So we assume that associated with every control g_σ is a *present cost* $\xi(\sigma, g_\sigma)$.* We further assume that, once a control $g_\sigma{}^k$ for a particular state σ has been specified, the next state is a *probabilistic* function of the current state, and that states prior to the current state have no influence on the next state except through the current state. That is, the sequence of states forms a *discrete-time Markov chain.* If the policies vary with time, then the Markov chain will also be time-varying. The uncertainty of the next state can be due to the influence of uncertain inputs or any other uncertain behavior of the system. The state transition probabilities at time k can be consolidated into a *transition* or *stochastic matrix* $\mathbf{P}^{G^k} = (P^{g^k}_{\sigma,\sigma'})$ where $P^{g^k}_{\sigma,\sigma'} = \Pr[\text{next state is } \sigma' \mid \text{current state is } \sigma \text{ and the control is } g_\sigma{}^k]$. We will use $\mathbf{P}_\sigma{}^{g^k}$ to represent the *row* of $\mathbf{P}^{G^k}$ corresponding to state σ and control $g_\sigma{}^k$, that is, the *state transition vector* corresponding to state σ and control $g_\sigma{}^k$.

We are concerned with the problem of determining the optimal sequence of K control sets. We define a *control policy* G as a sequence of control sets $\{G^k : 1 \le k \le K\}$. Once a policy G is specified, its *expected (total) cost* is defined as:

$$J^G = E\left[\sum_{k=K}^{1} \xi(\sigma^k, g_\sigma{}^k)\right] \tag{2.7}$$

where E denotes expectation and we have emphasized that time K is the starting time by making it the "lower" limit of the summation. We then define an *optimal control policy* G^* as

* In the most general case, this cost may be a function of time as well as state and control, but this is an unnecessary complication at this point.

one which minimizes the expected cost, that is:

$$J^{G^*} = J^* = \min_G J^G. \tag{2.8}$$

The quantity J^* is then the optimal expected cost.

We are now in a position to state the primary result of stochastic dynamic programming for finite horizon K. Recursively define* the values $v_\sigma{}^k$ as:

$$v_\sigma{}^k := \min_{g_\sigma{}^k} \left[\xi_{\sigma g^k} + \sum_{\sigma'} P^{g^k}_{\sigma,\sigma'} \cdot v^{k-1}_{\sigma'} \right] \tag{2.9}$$

for $1 \le \sigma \le S$ and $1 \le k \le K$, with $v_\sigma{}^0 := 0$. This can be represented more compactly in matrix form as follows. Let ξ^{G^k} be a vector of length S which represents the present costs for control set G^k, that is:

$$\xi^{G^k} := \begin{bmatrix} \xi(1, g_1^k) \\ \xi(2, g_2^k) \\ \cdot \\ \cdot \\ \cdot \\ \xi(S, g_S^k) \end{bmatrix}. \tag{2.10}$$

Then recursively define the length S vectors v^k as:

$$\mathbf{v}^k := \min_{G^k} \left[\xi^{G^k} + \mathbf{P}^{G^k} \cdot \mathbf{v}^{k-1} \right] \tag{2.11}$$

for $1 \le k \le K$, where $\mathbf{v}^0 := \mathbf{0}$ and the minimization is understood to be performed individually for each state in the vector $\mathbf{v}^k$. Then a control policy G^* is optimal if and only if each control set in G^* minimizes (2.11) for the appropriate k. Also, the optimal expected cost is $J^* = \mathbf{p} \cdot \mathbf{v}^K$, where $\mathbf{p}$ is the row vector of initial probabilities associated with the S states. Equation (2.9) or

* We will use the notation $x := y$ to mean "x is defined as y."

(2.11) is referred to as the optimality equation for the finite horizon stochastic dynamic programming problem.

We see from (2.11) that at each time k, each element $v_\sigma{}^k$ of $\mathbf{v}^k$ is the optimal expected cost for the k-step dynamic programming problem, given that the initial state is σ. In finding this optimal cost, we must consider the sum of the *present cost* of the control decision $\xi(\sigma, g_\sigma{}^k)$ and the quantity $\mathbf{P}_\sigma{}^{g^k} \cdot \mathbf{v}^{k-1}$, the *expected future cost* at time k for state σ and control $g_\sigma{}^k$. We conclude that, just as in the simple smuggler example, any specific problem is solved by solving a much more general problem, and at each step we must consider both the present and future costs of our decision.

Note that the expected future cost is derived from $\mathbf{v}^{k-1}$, the optimal expected costs at time $k-1$. Also, suppose that the next state of the system is independent of the control decision for some set of possible control decisions; that is, the state transition vector is the same for all options in the set. Then the *expected future costs* of the decisions in the set are identical for every time k, and the best decision *in this set* at every time k is the one with minimum *present cost*. This principle will be useful for quickly identifying control decisions which can never be optimal.

2.4.3 Present Application

Before continuing to present further results from the theory of stochastic dynamic programming, it is appropriate to pause here to begin to relate the present problem of priority packet discarding to the general problem description of the previous section.

The discrete time interval corresponds to the decision interval in the structural classification of section 2.2. We will see that this can always be made to represent a fixed interval of time. According to the general model, the state of the queue can be used to describe the system at any time instant. The control decisions are obviously packet discarding decisions, which

certainly depend on the state of the system and could also depend on the time instant. The cost of any decision is just the sum of the individual discarding costs associated with any discarded packets. The uncertainty associated with the next state is due to arrivals which may occur in the next decision interval. Both the number and delivery priorities of these arrivals are unknown, but will influence the next state. The expected discarding cost $E[C]$ defined in section 2.3 and the expected cost from the dynamic programming problem are obviously of the same nature, but to see an exact correspondence requires a particular packet discarding model and an extension of dynamic programming to the infinite horizon case, treated in the next section.

2.4.4 Infinite Horizon, Average Cost Case

So far we have considered dynamic programming problems for which the horizon is both known and finite. But what if, as in our packet discarding problem, the horizon is of unknown duration or is quite long relative to the discrete time intervals? For example, in [Kami89] it was shown that an overload period may last several hundred milliseconds for a system with speech traffic, which would be orders of magnitude larger than a packet service time if packets are on the order of a hundred bytes and the transmission speed is on the order of 100 Mbps (service time on the order of 10 microseconds). We may then wish to assume that the horizon is infinite and modify the dynamic programming problem accordingly.

The first problem we encounter is in the performance metric. If the horizon is infinite and all discarding costs are non-negative, then defining expected total cost as in (2.7) with $K \to \infty$ results in infinite expected cost for *all* policies. This problem can be handled either by defining a discounted expected cost or an average expected cost per step. We will choose the latter method, defining the average expected cost per step of a policy G as:

$$J^G = \lim_{K\to\infty} \frac{1}{K} \sum_{k=K}^{1} E\left[\xi(\sigma^k, g_\sigma{}^k)\right] \tag{2.12}$$

where E again denotes expectation and we have again made the first term in the sum correspond to the starting time K. We will often refer to J^G as simply the average cost of policy G.

2.4.4.1 Stationary Policies

We have seen that optimal policies in finite horizon dynamic programming may consist of control sets which vary with time. We now define a *stationary* or time-invariant policy as one for which the control sets G^k are identical for all k and equal to G. In such cases, we will continue to use G to represent the resulting stationary policy.

As a general principle, the fewer system variables a control algorithm depends on, the simpler it is. Removing the time dependence is then valuable to reduce implementation complexity. For example, suppose the optimal control policy was periodic in time. This would necessitate not only the ability to change the control decisions at each time instant, but would necessitate time synchronization as well. We would much rather deal with stationary policies.

Another advantage of stationary policies is that the number of such policies will be finite for our problem in which both the state space and the number of control actions for each state will be finite. This introduces the possibility of exhaustive search through all policies to find the optimal policy. For a particular analysis model, then, we would like to know how many stationary policies are possible. We will call the set of all possible *distinguishable* stationary policies the *policy universe*. We limit the set to distinguishable policies since, if two control (discarding) options result in the same next state (*i.e.*, have the same set of state transition probabilities) and have the same present cost, they are equivalent (or indistinguishable) with regard to our performance metric of expected discarding cost. For the remainder of this dissertation, we will omit this unwieldy adjective with the understanding that any reference to

possible discarding options or policies should be understood to be a reference to distinguishable options or policies.

Since a stationary policy is a specific control (discarding) decision for each state (independent of the time step), we need to know how many discarding options are possible for each state σ. Let $O(\sigma)$ represent this number. Then the number of policies in the policy universe is:

$$U = \prod_{\sigma} O(\sigma). \tag{2.13}$$

This expression will be useful for determining the size of the policy universe for the analysis models in subsequent chapters. We will see that the universe is often too large for exhaustive search to be practical.

A major result of infinite horizon dynamic programming is a simple sufficient condition that ensures the existence of a stationary policy which is optimal.

2.4.4.2 Existence of Optimal Stationary Policy

We first obtain a simple formula for the average cost for any stationary policy. Let G be any *stationary* policy, $\mathbf{P}^G$ be its associated transition matrix, and ξ^G its associated vector of present costs. Then we have:

$$\begin{aligned} J^G &= \lim_{K\to\infty} \frac{1}{K} \cdot \sum_{k=K}^{1} E\left[\xi(\sigma^k, g_{\sigma}{}^k)\right] \\ &= \lim_{K\to\infty} \frac{1}{K} \cdot \sum_{k=K}^{1} \left[\mathbf{p}\cdot\mathbf{P}^{K-k}\right]\cdot\xi^G \\ &= \mathbf{p}\cdot\left[\lim_{K\to\infty} \frac{1}{K} \cdot \sum_{k=0}^{K-1} \mathbf{P}^k\right]\cdot\xi^G \end{aligned} \tag{2.14}$$

where $\mathbf{p}$ is the vector of initial state probabilities and to simplify notation we have let $\mathbf{P}$ replace $\mathbf{P}^G$ so that $\mathbf{P}^k$ now represents the k^{th} power of $\mathbf{P}$. The limiting summation of powers of $\mathbf{P}$ is

known as the Cesaro limit (call it Π) and is guaranteed to exist for any stochastic matrix $\mathbf{P}$ [Kuma86]. Furthermore, if $\mathbf{P}$ is irreducible, the rows of Π are all identical so that $\mathbf{p}\cdot\Pi = \boldsymbol{\pi}$ where $\boldsymbol{\pi}$ is any row of Π and is the unique solution of $\boldsymbol{\pi} = \boldsymbol{\pi}\cdot\mathbf{P}$ subject to $\boldsymbol{\pi}\cdot\mathbf{e} = 1$ where $\mathbf{e} = (1\ 1\ \cdots\ 1)^T$. Thus, if G is any stationary policy and $\mathbf{P}^G$ is irreducible, the average cost is independent of the initial state distribution and we have:

$$J^G = \boldsymbol{\pi}\cdot\boldsymbol{\xi}^G. \tag{2.15}$$

Define an *irreducible policy* (not necessarily stationary) as one for which every transition matrix (that is, the transition matrix for every control set G^k) is irreducible. We can then say that (2.15) holds for *irreducible stationary policies.*

We now state a result concerning the existence of a stationary policy which is optimal under the average cost criterion. Let G^* be any best *stationary* policy, *i.e.,* any stationary policy with the lowest average cost among all stationary policies. If G^* is irreducible, that is, if $\mathbf{P}^{G^*}$ is irreducible, then there exist numbers J^* and v_σ for all σ satisfying:

$$J^* + v_\sigma = \min_{g_\sigma} \left[\xi_\sigma^{g} + \sum_{\sigma'} P^{g}_{\sigma,\sigma'} \cdot v_{\sigma'} \right]. \tag{2.16}$$

where we now represent the present discarding cost of state σ for policy G as ξ^g_σ rather than $\xi(\sigma, g_\sigma)$. This can be put in a more compact vector form as:

$$J^*\cdot\mathbf{e} + \mathbf{v} = \min_{G} \left[\boldsymbol{\xi}^G + \mathbf{P}^G\cdot\mathbf{v} \right] \tag{2.17}$$

where the minimization is again understood to be performed state by state. Furthermore, if the stationary policy G^* is chosen to minimize (2.16) for all states σ, then G^* is an optimal policy and J^* is the optimal average cost. Equation (2.16) (equivalently (2.17)) is the optimality equation for infinite horizon dynamic programming under the average cost criterion.

We have reached the following important conclusion.

> *For the problem of finding an optimal policy for the average cost criterion, if the best stationary policy is irreducible, we may restrict our attention to stationary policies.*

This conclusion may be applied in several ways. If we can show that *every* stationary policy is irreducible, the hypothesis is obviously satisfied, so we may search only among stationary policies. Unfortunately, we will see that this is not true for the systems we will analyze in subsequent chapters. Failing that, we could satisfy the hypothesis clause by showing that, for every stationary *reducible* policy, there exists a stationary *irreducible* policy with lower average cost. This is also generally very difficult.

But even if both of these approaches fail, we may still apply the result in the following way. The conclusion above guarantees the existence of a stationary (irreducible) policy which is optimal *within the class* of all irreducible policies. In subsequent chapters, we will restrict ourselves to irreducible policies in this way, so we will be interested in finding sufficient conditions for the irreducibility of a policy. Figure 2-5 is a graphical depiction of the universe of all possible discarding policies, divided into the four structural classes shown in Table 2-1 with the irreducible policies shown as a subset.

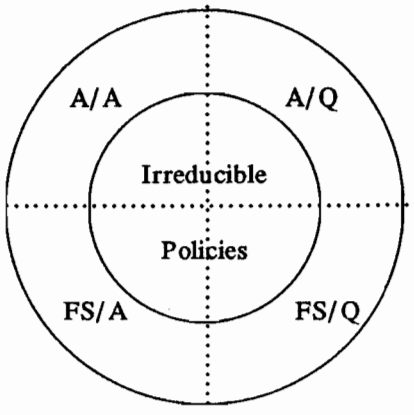


Figure 2-5. Subsets of the Universe of Discarding Policies

The numbers v_σ in (2.16) or (2.17) have the following interpretation. If we consider a finite horizon problem and calculate the optimal expected costs $v_\sigma{}^k$ at time k recursively for all

σ and k according to the optimality equation (2.9), then:

$$v_\sigma{}^K = K \cdot J^* + v_\sigma + o(1) \tag{2.18}$$

where the notation $o(1)$ means that $o(1) \to 0$ as $K \to \infty$. So v_σ is the deviation of $v_\sigma{}^k$ from $K \cdot J^*$ as $K \to \infty$. However, noting that the minimization in (2.16) is unaffected if each v_σ is replaced by $v_\sigma + \alpha$ for some constant α, we can more properly interpret the difference $v_\sigma - v_{\sigma'}$ as the relative advantage of starting in state σ instead of σ' as $K \to \infty$. The values v_σ are thus often referred to as the *relative values* of the optimal policy.

2.4.4.3 Policy Improvement Algorithm

The final result that we will review here gives us a fast-converging iterative algorithm for finding a stationary policy which is optimal under the average cost criterion. We first claim that for any irreducible stationary policy G, there exists a vector $\mathbf{v}^G$ such that:

$$J^G \cdot \mathbf{e} + \mathbf{v}^G = \boldsymbol{\xi}^G + \mathbf{P}^G \cdot \mathbf{v}^G. \tag{2.19}$$

To see this we first note that, since $\mathbf{P}$ is irreducible, $J^G = \boldsymbol{\pi}^G \cdot \boldsymbol{\xi}^G$; that is, $\boldsymbol{\pi}^G \cdot (J^G \cdot \mathbf{e} - \boldsymbol{\xi}^G) = \mathbf{0}$. But since $\boldsymbol{\pi}^G$ is the *unique* solution of $\boldsymbol{\pi}^G \cdot (\mathbf{P}^G - \mathbf{I}) = \mathbf{0}$, $\boldsymbol{\pi}^G$ spans the nullspace of $\mathbf{P}^G - \mathbf{I}$, so $(J^G \cdot \mathbf{e} - \boldsymbol{\xi}^G)$ lies in the range space of $[\mathbf{P}^G - \mathbf{I}]$. This assures the existence of a vector $\mathbf{v}^G$ such that:

$$J^G \cdot \mathbf{e} - \boldsymbol{\xi}^G = \left[\mathbf{P}^G - \mathbf{I}\right] \cdot \mathbf{v}^G \tag{2.20}$$

which is just another form of (2.19) above. We note that (2.19) represents a set of S linear equations in $S+1$ unknowns, and we may arbitrarily assign a value (which may as well be 0) to one of the $v_\sigma{}^G$, and then find the unique solution for the remaining S variables. The $v_\sigma{}^G$ have the same interpretation as before: $v_\sigma{}^G - v_{\sigma'}^G$ is the relative advantage of starting in state σ instead of σ' as $K \to \infty$. So the $v_\sigma{}^G$ are called the *relative values* for policy G.

Now suppose that policy G' is optimal. By the optimality equation (2.17) this implies:

$$J^{G'}\cdot\mathbf{e} + \mathbf{v}^{G'} = \min_{G}\left[\boldsymbol{\xi}^{G} + \mathbf{P}^{G}\cdot\mathbf{v}^{G'}\right]. \tag{2.21}$$

If G' is not optimal, we may choose a policy G'' which attains the minimum in $\min_{G}[\boldsymbol{\xi}^{G} + \mathbf{P}^{G}\cdot\mathbf{v}^{G'}]$. That is, for each state σ, we choose the control g_σ'' that minimizes $[\xi_\sigma^{g} + \mathbf{P}_\sigma^{g}\cdot\mathbf{v}^{G'}]$ over all possible g_σ, where $\mathbf{P}_\sigma^{g}$ is the transition vector for state σ and control g_σ. Recall from our previous definition that there are $O(\sigma)$ possibilities for g_σ. Then we have $J^{G'}\cdot\mathbf{e} + \mathbf{v}^{G'} > \boldsymbol{\xi}^{G''} + \mathbf{P}^{G''}\cdot\mathbf{v}^{G'}$, where the vector inequality $\mathbf{x} < \mathbf{y}$ means that $x_i \le y_i$ for all i and there exists at least one i for which strict inequality holds. Then, multiplying both sides by $\boldsymbol{\pi}^{G''}$ and simplifying, we see that $J^{G'} > J^{G''}$, so that G'' is a better policy than G'.

The algorithm for finding the optimal policy is now apparent. We may begin with any arbitrary irreducible stationary policy $G^{(0)}$ and find $J^{G^{(0)}}$ and $\mathbf{v}^{G^{(0)}}$ from the S linear simultaneous equations represented by (2.19). We then choose the policy $G^{(1)}$ that achieves the minimum in $\min_{G}[\boldsymbol{\xi}^{G} + \mathbf{P}^{G}\cdot\mathbf{v}^{G^{(0)}}]$. If $G^{(1)}$ is the same as $G^{(0)}$, we have found the optimal policy since this policy then satisfies the optimality equation (2.16). If not, $G^{(1)}$ is guaranteed to be a better policy than $G^{(0)}$, and we may again find $J^{G^{(1)}}$ and $\mathbf{v}^{G^{(1)}}$. We continue choosing $G^{(n+1)}$ from the $J^{G^{(n)}}$ and $\mathbf{v}^{G^{(n)}}$ until we find an optimal policy, that is, until the policy is unchanged in successive steps.

This algorithm is called the *policy improvement* algorithm, and it generally converges quite rapidly. We see that there are two primary steps in the iteration. The first is the determination of the relative values, which involves solving S simultaneous linear equations. This step is sometimes called the *value determination* step. The second step is often called the *policy improvement* step. This step requires an exhaustive search through $O(\sigma)$ possibilities for each state σ in order to find a better control decision for that state, but simple matrix multiplication is all that is required for each possible control.

If the number of states S is large, but there are only a few possible control options for each state ($O(\sigma)$ is small), the value determination step may dominate the computation (hence time) required to produce an optimal policy. On the other hand, if the number of states S is small, but the number of control options for each state is large, the policy improvement step may dominate. A significant result of each of the next two chapters is the derivation of a computationally efficient method for the value determination step for each system model considered.

2.5 Summary of Results

In this chapter we have obtained the following new results.

1. We developed a general model for the analysis of priority packet discarding algorithms, characterized by three size dimensions: the queue capacity N, the maximum number of arrivals in a fixed service interval L, and the number of packet delivery priorities D. The model dimensions allow one to compare the "size" of different systems. The model is particularly well-suited to future broadband packet switching techniques such as the Asynchronous Transfer Mode (ATM) recently adopted for Broadband ISDN.

2. We developed a classification system for discarding algorithms consistent with the general analysis model. The classification is in two dimensions: control or discarding intervals (either arrival intervals or fixed service intervals) and discarding set (either arriving packets only or any queued packets). This classification allows one to compare the discarding control structures of different systems which may have the same "size" dimensions.

3. We specified an appropriate optimality criterion (expected discarding cost) and showed how this criterion is a generalization of the throughput criterion which has been used in many previous studies.

4. We showed how stochastic dynamic programming can be used to analyze priority packet discarding systems.

2.6 References

[Boud71] R. Boudarel, J. Delmas, and P. Guichet, *Dynamic Programming and Its Application To Optimal Control,* Academic Press, New York, 1971.

[CCIT89] CCITT Recommendation I.121, "Broadband Aspects of ISDN," Blue Book, Geneva, Switzerland, June 1989.

[Gerl80] M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey," *IEEE Transactions on Communications,* Vol. COM-28, No. 4, April 1980.

[Howa60] R. Howard, *Dynamic Programming and Markov Processes,* MIT Press, Cambridge, 1960.

[Kami89] T. Kamitake and T. Suda, "Evaluation of an Admission Control Scheme for an ATM Network Considering Fluctuations in Cell Loss Rate," Proceedings of the IEEE Globecom Conference, Dallas, November 1989.

[Kamo81] F. Kamoun, "A Drop and Throttle Flow Control Policy for Computer Networks," *IEEE Transactions on Communications,* Vol. COM-29, No. 4, April 1981.

[Kuma86] P. R. Kumar and P. Varaiya, *Stochastic Systems: Estimation, Identification, and Adaptive Control,* Prentice Hall, Englewood Cliffs, NJ, 1986.

[Lam79] S. S. Lam and M. Reiser, "Congestion Control of Store-and-Forward Networks by Input Buffer Limits -- An Analysis," *IEEE Transactions on Communications,* Vol. COM-27, No. 1, January 1979.

[Luan88] D. T. Luan and D. M. Lucantoni, "Throughput Analysis of a Window-Based Flow Control Subject to Bandwidth Management," Proceedings of the IEEE Infocom Conference, New Orleans, March 1988.

[Luss89] H. Luss, "Optimization: Methodology, Algorithms, and Applications," *AT&T Technical Journal,* Vol. 68, No. 3, May/June 1989.

[Nomu88] M. Nomura, T. Fujii and N. Ohta, "Layered Packet-Loss Protection for Variable Rate Video Coding Using DCT," Second International Packet Video Workshop, Torino, Italy, September 1988.

[Petr89] D. W. Petr, L. A. DaSilva, V. S. Frost, "Priority Discarding of Speech in Integrated Packet Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 5, June 1989.

[Ross83] S. Ross, *Introduction to Stochastic Dynamic Programming*, Academic Press, New York, 1983.

Chapter 3

Optimal A/A Packet Discarding

CONTENTS

Chapter 3

LIST OF FIGURES

LIST OF TABLES

Chapter 3

Optimal A/A Packet Discarding

This chapter is devoted to the analysis of systems that come under the A/A classification of section 2.2, that is, systems for which the discarding interval is the *Arrival* interval and the discarding set is *Arriving* packets. We begin with a discussion of the importance of A/A systems (section 3.1), providing the motivation for the analysis of their behavior. We then develop a specific analysis model (section 3.2), consistent with the general model of section 2.1, for the large sub-class of A/A systems in which queue *state* information is limited to the *number* of packets queued (excluding the just-arrived packet) so that delivery priorities are considered only for arriving packets. We then (section 3.3) use the model and dynamic programming (reviewed in Chapter 2) to obtain some general analytical results. Specifically, we derive computationally efficient methods for finding the state distribution and expected cost for any given policy. We also derive a similarly efficient method of performing the value iteration step of the policy improvement algorithm used to find the optimal policy for a given set of model parameters. We then apply these techniques specifically to the default policy of discarding packets if and only if the queue is full. We also hypothesize that all optimal policies for this model must have a specific form which we call *nested queue thresholds* and sketch an approach to a possible proof. In section 3.4, we present some examples illustrating the significance of the performance advantage of optimal discarding relative to the default policy. We close the chapter with a summary of results.

3.1 Motivation for A/A Discarding Analysis

We have already seen in Chapter 1 that many previous researchers [Lam79, Gerl80, Kamo81] have studied A/A systems. An important reason for this interest is that, of the four structural classifications given in section 2.2, A/A systems are probably the simplest: they represent simple admission control to an otherwise "standard" queuing system; they require only one reserved storage position for arrivals between decisions; and all decisions are binary (admit or discard the arriving packet). This simplicity is important both for analysis and implementation of the systems. Particularly appealing for their simplicity of implementation are systems characterized by a series of *nested queue thresholds* such as the Structure Buffer Pool system described in [Gerl80] and the speech packet shortening system of [Muis86]. With nested queue thresholds, a priority d packet is admitted to the queue only if the current queue fill does not exceed a threshold associated with priority d packets, where the thresholds are larger for higher priorities.

Unfortunately, as detailed in Chapter 1, none of the previous studies can be directly applied to the present priority packet discarding problem, especially for the general system model of section 2.1. In particular, none of the previous studies has addressed the question of optimality of specific control decisions. It is for these reasons that we proceed to undertake the analysis of A/A systems under the model of section 2.1, with particular emphasis on optimality of control decisions.

3.2 Analysis Model for Queue Fill A/A Systems

In this section we extend the model of section 2.1 specifically for a large sub-class of A/A systems, namely ones that restrict queue state information to the *number* of packets queued (excluding the just-arrived packet), so that only the delivery priority of the just-arrived packet is considered. We will call this subclass of systems *queue fill A/A* systems. This specific model

will then be used in conjunction with dynamic programming in the analysis that follows. The model is applicable to a large sub-class of A/A systems, namely ones that restrict queue state information to the number of packets queued, without regard for priority.

3.2.1 Initial Model and Assumptions

3.2.1.1 General Description

Our starting point is the general model of section 2.1, which is characterized by three "size" dimensions: the queue capacity (N), the maximum number of arrivals per fixed service interval (L), and the number of delivery priorities (D). We now augment the model for the analysis of A/A discarding systems. We first divide the (fixed) service interval into L equal subintervals as in Figure 3-1.

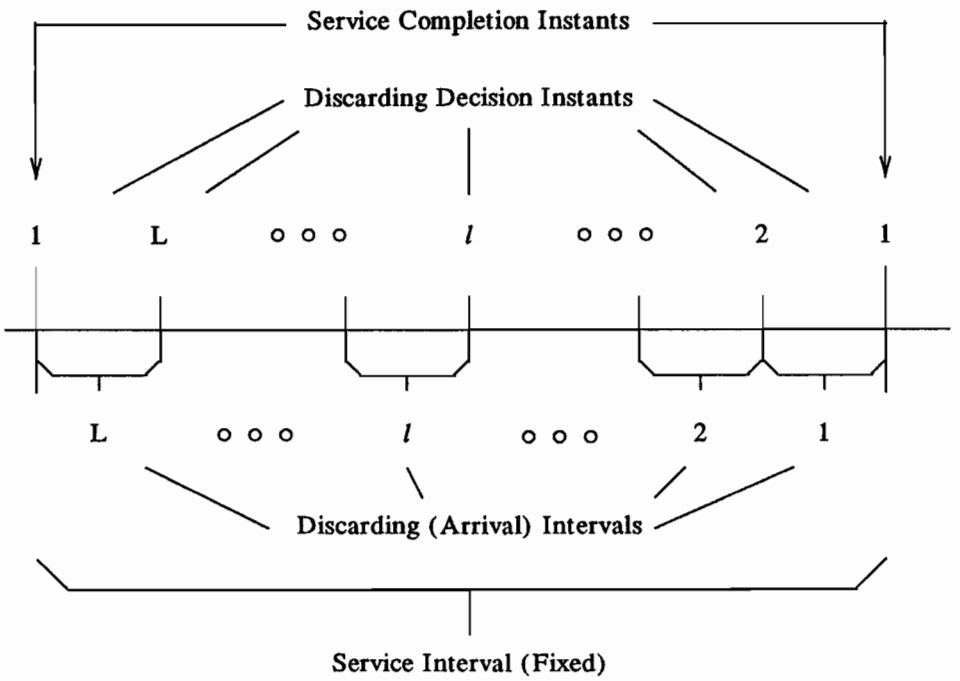


Figure 3-1. Partition of Service Interval into Arrival Intervals

We assume that at most one packet can arrive in a subinterval, so the subintervals can be used as the discarding (arrival) intervals. The discarding decision instants are synchronized with the

service completion instants as shown in Figure 3-1. If a packet arrives in a subinterval, a binary decision is made at the discarding instant: either admit the arrived packet into the queue or discard it. We will see that each arrival interval will correspond to a dynamic programming step. For the subinterval just before a service completion, the discarding decision is made immediately prior to the service completion.

3.2.1.2 Model Parameters

We first turn our attention to describing the parameters which model the arrival process. As in previous chapters, we let λ be the average arrival rate (during the overload period being analyzed), but we measure it in units of arrivals per service interval. Thus $\lambda = \rho$, where ρ is the normalized average queue load. We then assume that arrivals are independent, that the probability of a single arrival in any given subinterval is λ/L, and that the probability of multiple arrivals in any subinterval is zero. The probability distribution of the number of arrivals in a service interval (L subintervals) is then a binomial distribution with parameters L and λ/L, for which the expected value is $(L)(\lambda/L) = \lambda$. We note as an aside that letting $L \rightarrow \infty$ would result in a Poisson arrival process.

Also in Chapter 2, we introduced the the conditional distribution of the delivery priority of an arrived packet (given that an arrival has occurred) and called this probability distribution $\{p_d\}$ for $d \in \{1,2,\ldots,D\}$. For convenience we will completely specify the arrival statistics with a single probability distribution $\{\phi_d\}$ for $d \in \{0,1,\ldots,D\}$ as follows:

$$\phi_d = \begin{cases} 1-(\lambda/L) & \text{for } d=0 \\ (\lambda/L)\cdot p_d & \text{for } d \in \{1,2,,\ldots,D\} \end{cases} \tag{3.1}$$

so that ϕ_0 is the probability of no arrival (a null packet arrival) in a given arrival subinterval and ϕ_d for $1 \le d \le D$ is the (unconditioned) probability of a priority d arrival in a given subinterval.

As mentioned in Chapter 2, we will assume that the distribution of discarding costs $\{ c_d \}$ is strictly increasing in d. The two functions $\{ \phi_d \}$ and $\{c_d\}$ form a complete description of the model parameters.

3.2.1.3 Model State Space: A 3-Dimensional Representation

Discarding decisions are made on the basis of the *state* of the system at the discarding instant. We choose to view the state space of this model as existing in three dimensions, with the dimensions corresponding roughly to the general model dimensions (N,L,D). The first dimension of the overall system state encompasses information about the queue *prior to* the arrival. We will call this dimension the *queue state* and will defer for the moment our discussion of how it is specified. The second dimension is the particular subinterval within a service interval for which a discarding decision is being made. We will refer to this dimension as the *phase* and represent it as l with $l \in \{1,2,\ldots,L\}$. The final dimension indicates the delivery priority of the arrived packet *(arrival priority)* and will be represented as d with $d \in \{0,1,\ldots,D\}$ and $d=0$ representing no arrival (that is, a "null" packet).

3.2.1.3.1 Specifying Queue State

For our analysis, in which expected discarding cost is the performance metric of interest, the most complete description of queue state would be an ordered set of delivery priorities representing the queued packets. However, other less complete descriptions are also possible. For example, we could specify only the *number* of queued packets in *each* delivery priority. This would be appropriate for discarding policies with structures such as Complete Partitioning (CP), Sharing with Maximum Queues (SMXQ), or Sharing with Minimum Allocation (SMA) as described in Chapter 1. For the present model, we choose to restrict the queue state information to the *total number* of queued packets (excluding the one just arrived and the one in service), *without* regard for their delivery priorities. In this case, delivery priorities are

considered only in the arrival priority dimension of the state. We will refer to the resulting state dimension as the *queue fill* and represent it as n with $n \in \{0,1,\ldots,N-1\}$. The upper limit of n is $N-1$ since, as discussed in section 2.2, one queue position must be *reserved* for the arriving packet. We can now specify the state of this *queue fill A/A model* with the triple (n,l,d). Figure 3-2 shows that we are considering a subset of the A/A "quadrant" of the discarding policy universe by using this model.

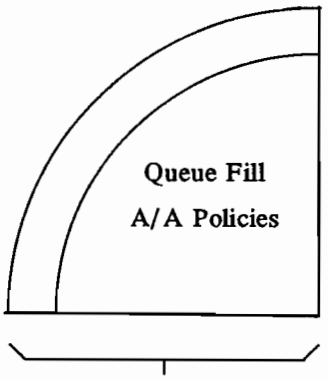


Figure 3-2. The Queue Fill Model as an A/A Policy Subset

3.2.1.4 Policy Universe

We recall from our discussion of dynamic programming in section 2.4 that a *stationary discarding policy* is defined as a particular discarding decision for each state that is independent of time step. For our queue fill A/A model, the number of distinct states is $S = N \cdot L \cdot D$. Since the discarding decision for each state is binary (either admit or discard the arrived packet), the number of possible control options for state σ is $O(\sigma)=2$ for all σ. Thus, the number of policies in the policy universe (set of all possible policies) is $U = \prod_{\sigma} O(\sigma) = 2^S = 2^{N \cdot L \cdot D}$. We will see shortly that we can, with very little effort, declare many of these policies to be suboptimal, thereby effectively shrinking the policy universe.

3.2.1.5 Allowed State Transitions

From the form of the model (Figure 3-1), we immediately see that from one decision instant to the next, the phase always decrements (modulo L):

$$l \rightarrow \begin{cases} l-1 & \text{for } l > 1 \\ L & \text{for } l = 1 \end{cases} \tag{3.2}$$

and the arrival priority is replaced by the new arrival priority:

$$d \rightarrow d' \tag{3.3}$$

where d' is the priority of the new arrival. The queue fill either increments, stays the same, or decrements, depending on the phase. If the phase is not associated with a service completion (that is, if $l > 1$), the queue fill remains the same if the arriving packet ($d > 0$) is discarded and increments if the packet is admitted to the queue:

$$n \rightarrow \left\{ \begin{array}{ll} n & \text{if } n < N-1 \text{ and } \textit{discard} \\ n+1 & \text{if } n < N-1 \text{ and } \textit{admit} \\ N-1 & \text{if } n = N-1 \end{array} \right\} \text{for } l > 1 \text{ and } d > 0 \tag{3.4}$$

since discarding is required if $n = N-1$. If the phase is associated with a service completion ($l=1$), the queue fill decrements if the arriving packet is discarded and stays the same if the packet is admitted to the queue:

$$n \rightarrow \left\{ \begin{array}{ll} n-1 & \text{if } n > 0 \text{ and } \textit{discard} \\ n & \text{if } n > 0 \text{ and } \textit{admit} \\ 0 & \text{if } n = 0 \end{array} \right\} \text{for } l = 1 \text{ and } d > 0 \tag{3.5}$$

since, for $n=0$, it does not matter whether the packet is discarded or admitted. If there is no arrival ($d=0$), then there is usually no decision to be made (equivalently, the null packet is discarded). However, if the queue is empty and a service completion is imminent, we will consider that the null packet is admitted so that it will go into the server, keeping the system

synchronized:

$$n \rightarrow \left\{ \begin{array}{lll} n-1 & \text{if} & n>0 \text{ and } l=1 \\ n & \text{if} & n>0 \text{ and } l>1 \\ 0 & \text{if} & n=0 \end{array} \right\} \text{for } d=0. \tag{3.6}$$

3.2.1.6 Present Discarding Costs

If at any time the decision is to discard the arrived packet (of priority d), the *present discarding cost* under this model is clearly the cost c_d of discarding a priority d packet. If the decision is to accept the packet, the present discarding cost is 0.

We see from relations (3.5) and (3.6) that there are two special cases to consider when $d>0$, that is, when a non-null packet has arrived. If $l>1$ and $n = N-1$, the arriving packet is *always* discarded since there must always be a queue position available for the next arrival. For $l=1$ and $n=0$, we have seen that the *next state,* and hence the *expected future cost,* does not depend on the discarding decision. We saw in section 2.4 that in such cases the optimal decision will minimize the present discarding cost, so we will *never* discard the arrived packet.

If there is no arrival ($d=0$), the present discarding cost is always 0 whether we consider the null packet to be discarded or not.

3.2.2 An Equivalent 2-Dimensional State-Space Model

We note from relation (3.3) that the next arrival priority is independent of the current arrival priority. This leads us to explore the possibility of excluding this dimension from the state and treating it as an input instead. If we are to be able to do so, we must be able to identify the form of the decisions (based on the resulting 2-dimensional state) and the present discarding costs.

We begin by showing that optimal policies must exist in a subset of the policy universe for the 3-dimensional state model. Keeping with the dynamic programming notation introduced in

section 2.4, we will denote the optimal total expected cost to time 0 at time k for a state σ as $v_\sigma{}^k$, or if the state is specified as the triple (n,l,d), as $v^k(n,l,d)$. Now consider any state σ at time $k+1$ with given n, $l>1$ and d^{k+1} (the delivery priority of the arrival at time $k+1$) so that σ can be represented as (n,l,d^{k+1}). We first note that if $d^{k+1}=0$ there is really no decision to be made, the next state is $(n,l-1,d^k)$, and we may say that this null packet is always discarded. If $d^{k+1}>0$ and we accept the just-arrived packet, the total cost (sum of present and expected future costs) will be $0+\sum_{d^k} p_{d^k}\cdot v^k(n+1,l-1,d^k)$ where d^k represents the *next* arriving packet. On the other hand, if we discard the just-arrived packet, the total cost will be $c_{d^k}+\sum_{d^k} p_{d^k}\cdot v^k(n,l-1,d^k)$. However, due to the strict monotonicity of the discarding cost function, there must exist a $d_\sigma\in\{0,1,\ldots,D\}$ such that the total cost for retaining any arrived packet lies between the total cost of discarding a packet of priority d_σ and the total cost of discarding a packet of priority $d_\sigma+1$. That is:

$$c_{d_\sigma}+\sum_{d^k} p_{d^k}\cdot v^k(n,l-1,d^k) \le \sum_{d^k} p_{d^k}\cdot v^k(n+1,l-1,d^k) < c_{d_\sigma+1}+\sum_{d^k} p_{d^k}\cdot v^k(n,l-1,d^k) \tag{3.7}$$

where we have temporarily (for relation 4.7 *only*) defined $c_0=-\infty$ and $c_{D+1}=\infty$. We will call d_σ the *priority threshold* for state σ. Note that d_σ does *not* depend on d_{k+1}. Combining this with the strict monotonicity of the discarding cost function, we see that an optimal decision for this state σ at time $k+1$ is governed by:

If $d^{k+1}\le d_\sigma$, *then discard*; *otherwise admit*. (3.8)

We conclude that for the D states in the 3-dimensional model characterized by a given n and $l>0$, there are only $D+1$ decisions which could *possibly* be optimal (*i.e.*, which are not guaranteed to be suboptimal by their form) out of the 2^D possible decisions. We will refer to these decisions as *candidate decisions* and the resulting control policies as *candidate policies*.

Similar arguments lead to the same conclusion for the case of $l=1$. For $n=N-1$ and $l>1$, we have already seen that discarding is always necessary, so we have $d_\sigma=D$ for these states. The states characterized by $n=0$ and $l=1$ are a special case since these are the only states for which we admit a null packet. For consistency of notation, we will say that the priority threshold for these states is $d_\sigma=-1$.

We also conclude that we may use a 2-dimensional model which includes only the queue fill (n) and phase (l) dimensions. This is because we have just shown that the *form* of an optimal discarding decision (for the 3-dimensional model) is always to specify a *priority value* d_σ, and this decision specification depends *only* on queue fill (n) and phase (l). Thus we can now treat the arriving packet as an input rather than part of the state, and it would be would then be discarded or admitted according to relation (3.8).

We have arrived at a 2-dimensional model with $S=NL$ states, each of which can be characterized by the pair $\sigma=(n,l)$. A (stationary) policy for the 2-dimensional model is then a specific priority value d_σ for each state σ. As we have seen above, this priority value is fixed for L of the NL total states. So there are $O(\sigma)=D+1$ possible discarding options for each state, and the total number of possible policies for the 2-dimensional model is reduced to $U = \prod_\sigma O(\sigma) = (D+1)^{NL-L}$. For fairly small systems, this policy universe might be small enough for exhaustive search to be a reasonable method for finding an optimal policy, if a computationally efficient means of determining the average cost for each policy could be found. We will show in subsequent sections that there are two such efficient algorithms for finding the average cost of a policy. The first is a recursive algorithm for finding the state distribution of a policy, from which the average discarding cost immediately follows. The second is a recursive algorithm for determining the relative values and average cost of a policy, so this second algorithm would also be useful as part of the policy improvement algorithm for finding the optimal policy for a given set of model parameters.

3.2.2.1 State Transitions and Discarding Costs

The allowed state transitions for the 2-dimensional model are identical to those for the 3-dimensional model with the arriving priority dimension eliminated. From (3.8) we see that the probability of discarding an arriving packet is the probability that its priority is d_σ or less. We introduce the following shorthand notation for this probability:

$$\overset{d_\sigma}{\Phi} := \sum_{d=0}^{d_\sigma} \phi_d \qquad d_\sigma \in \{0,1,\ldots,D\} \tag{3.9}$$

where $\overset{d_\sigma}{\Phi} := 0$ for the special case of $d_\sigma = -1$ for $\sigma=(n,l)=(0,1)$. We see that $\overset{d_\sigma}{\Phi}$ is just the cumulative distribution function of the probability distribution ϕ_d. We have represented it in this particular form because we will use the general notation $\overset{y}{X}$ throughout this dissertation to represent a summation of (possibly weighted) discrete probabilities from the lower probability index to index y.

We will choose to let α_σ represent the probability that an arriving packet is admitted when the system is in state σ , so that:

$$\alpha_\sigma = 1 - \overset{d_\sigma}{\Phi}. \tag{3.10}$$

The allowed state transitions may now be easily represented in a state transition matrix by indexing the states according to $\sigma = (n \cdot L)+l$ as in the table below.

State Index σ	(Queue Fill n, Phase l)
1	(0,1)
2	(0,2)
.	.
.	.
.	.
L	(0,L)
L+1	(1,1)
.	.
.	.
.	.
NL	(N-1,L)

We then have the following form for the transition probabilities:

$$Pr(\sigma \rightarrow \sigma') = \begin{cases} 1-\alpha_\sigma & \text{for} \quad \sigma' = \sigma - 1 \\ \alpha_\sigma & \text{for} \quad \sigma' = \sigma + (L-1) \\ 0 & \text{for} \quad \textit{all other } \sigma' \end{cases} \tag{3.11}$$

We note that we always have $\alpha_1 = 1$ and $\alpha_{NL} = \alpha_{NL-1} = \ldots = \alpha_{NL-(L-2)} = 0$. For example, the form of the resulting state transition matrix $\mathbf{P}^G$ for $N = 3$ and $L = 3$ and any policy G is then:

$$\mathbf{P}^G = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1-\alpha_2 & 0 & 0 & \alpha_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1-\alpha_3 & 0 & 0 & \alpha_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1-\alpha_4 & 0 & 0 & \alpha_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-\alpha_5 & 0 & 0 & \alpha_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1-\alpha_6 & 0 & 0 & \alpha_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1-\alpha_7 & 0 & 0 & \alpha_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.12}$$

In general, the first row and last $L-1$ rows contain a single unity value and the other rows contain two non-zero values separated by $L-1$ zeros. Note that $\mathbf{P}^G$ depends on the specific policy G because the probabilities α_σ depend on the policy (3.9 and 3.10).

When we try to identify the present cost of a discarding decision for this 2-dimensional model, we quickly see that we must think in terms of an *expected* present cost, rather than a deterministic one. This is a departure from the classical dynamic programming formulation, but poses no special problems. We introduce the following notation for this expected present cost for a state $\sigma = (n,l)$ and associated control decision g_σ (that is, priority value d_σ):

$$\xi_\sigma{}^g = \overset{d_\sigma}{\chi} := \sum_{d=0}^{d_\sigma} \phi_d \cdot c_d \tag{3.13}$$

Note that since $d_1 = -1$ and $d_\sigma = D$ for $NL-(L-2) \le \sigma \le NL$, we have $\xi_1 = 0$ and $\xi_\sigma = \overset{D}{\chi}$ for $NL-(L-2) \le \sigma \le NL$.

The expected future cost at time $k+1$ is given by:

$$(1-\alpha_\sigma)\cdot v_\sigma{}^k + \alpha_\sigma \cdot v_{\sigma+L-1}^k \tag{3.14}$$

where $v_\sigma{}^k := 0$ for $\sigma < 0$ or $\sigma > NL$.

3.3 Analysis for Given Policy

We assume in this section that a particular irreducible stationary policy G is specified, and then use the dynamic programming techniques from section 2.4 to derive results for its behavior and optimality.

3.3.1 State Distribution and Expected Cost

Due to the form of the transition matrix, the limiting state probabilities, and hence the average expected cost per arrival, are easily found. We begin by recalling from section 2.4 that if $\mathbf{P}^G$, the transition matrix resulting from stationary policy G, is irreducible, then the average expected cost per step is given by:

$$J^G = \boldsymbol{\pi}^G \cdot \boldsymbol{\xi}^G \tag{3.15}$$

where $\boldsymbol{\xi}^G$ is the (column) vector of present discarding costs and $\boldsymbol{\pi}^G$ is the (row) vector representing the unique steady state probability distribution satisfying:

$$\boldsymbol{\pi}^G = \boldsymbol{\pi}^G \cdot \mathbf{P}^G, \qquad \boldsymbol{\pi}^G \cdot \mathbf{e} = 1 \tag{3.16}$$

and $\mathbf{e} := (1 \cdots 1)^T$. The following two conditions, taken together, are sufficient for $\mathbf{P}$ to be irreducible.

1. The probability of zero arrivals in an arrival interval is strictly positive ($q_l > 0$). That is, λ is strictly less than L.

2. For all states σ such that $L \le \sigma \le NL$, we do not allow $d_{\sigma-1} = d_{\sigma-2} = \cdots = d_{\sigma-(L-1)} = D$. That is, we do not allow

$$\alpha_{\sigma-1} = \alpha_{\sigma-2} = \cdots = \alpha_{\sigma-(L-1)} = 0.$$

Condition 1 restricts the arrival parameters, while condition 2 restricts the policy definition by prohibiting any $L-1$ consecutive states (except the last $L-1$) from discarding all arriving packets. Roughly speaking, this second restriction ensures that the larger queue fill states can be reached by preventing the policy from discarding all packets at some partial queue fill.

We first show that condition 2 (by itself) is a necessary condition for irreducibility. This is clear from the form of the transition matrix $\mathbf{P}^G$ (3.12), in which the only possible transitions *to* a state σ are from state $\sigma+1$ or $\sigma-(L-1)$ and the only transitions *from* a state σ are to state $\sigma-1$ or $\sigma+(L-1)$. Now suppose there exists a state σ such that $\alpha_{\sigma-1}=\alpha_{\sigma-2}=\cdots=\alpha_{\sigma-(L-1)}=0$ and consider the set of states $\{\sigma, \sigma+1, \ldots, NL\}$. It is clear that these states are all transient: if the system starts in any state in this set, it will eventually transition to state $\sigma-1$ and can then never return to any state in the set. Thus the system could be reduced by eliminating these states.

We may demonstrate irreducibility by showing that, starting from some initial state σ_0, there exists a set of transitions, each of which has non-zero probability, which takes us through all states and back to the initial state σ_0. We will use state NL as our initial state. Condition 1 ensures that $1-\alpha_\sigma$ is non-zero for all σ in the range from 2 to NL, which in turn ensures that we can transition from state NL to state 1, passing through all states in between in sequence. We must now show that we may return to state NL. Conditions 1 and 2 together ensure this. We will always transition from state 1 to state $L-1$. Now suppose that $d_{L-1} = D$, or equivalently that $\alpha_{L-1} = 0$. Condition 1 ensures that we can transition "up" to a state σ_1 with $\alpha_{\sigma 1} > 0$ so that we may transition from state σ_1 to state σ_1+L-1. Condition 2 ensures that $\sigma_1 \geq 2$ and that $\alpha_{\sigma_1+L} > 0$. In this way we have managed to get "past" state $L-1$. We may continue to transition "down" in this way, repeating the "backing up" procedure as

necessary, until we get to one of the last L states. Once in this set, we will transition (with probability 1) to state $NL-(L-1)$. Condition 2 ensures that $\alpha_{NL-(L-1)} > 0$, so we may at last transition to state NL, and the cycle is complete.

With these restrictions to assure irreducibility, we can now show that minimizing J^G is identical to minimizing $E[C]$ as given in section 2.3 (equation 2.1). We first note that:

$$J^G = \pi^G \cdot \xi^G = \sum_{\sigma} \pi_\sigma \cdot \chi^{d_\sigma} = \sum_{\sigma} \pi_\sigma \cdot \left[\sum_{d=0}^{d_\sigma} \phi_d \cdot c_d \right]. \tag{3.17}$$

We can sum first over d and then over σ if we are careful to include only those σ for which $d_\sigma \geq d$, so we have:

$$J^G = \sum_{d} \phi_d \cdot c_d \cdot \sum_{\sigma : d \leq d_\sigma} \pi_\sigma \tag{3.18}$$

where the second summation is over all σ such that $d \leq d_\sigma$. But this second sum is just δ_d, the probability of discarding an arriving priority d packet. Recalling the definition of ϕ_d from (3.1), we finally have:

$$J^G = \sum_{d} \phi_d \cdot c_d \cdot \delta_d = \frac{\lambda}{L} \cdot \sum_{d} p_d \cdot c_d \cdot \delta_d = \frac{\lambda}{L} \cdot E[C]. \tag{3.19}$$

Since J^G and $E[C]$ for a given policy are the same within a constant factor, by minimizing J_G with respect to all policies G, we are minimizing $E[C]$ as well.

We now show that, again subject to the irreducibility requirement, we can find π^G for a given policy G with a simple iterative algorithm. Referring to the example matrix (3.12) we see from the matrix equation (3.16) that for any $2 \leq \sigma \leq NL$:

$$\pi_{\sigma-1} = \alpha_{\sigma-L} \cdot \pi_{\sigma-L} + (1-\alpha_\sigma) \cdot \pi_\sigma \tag{3.20}$$

where $\alpha_1 = 1$, $\alpha_{N \cdot L} = \alpha_{N \cdot L-1} = \cdots = \alpha_{N \cdot L-(L-2)} = 0$, and $\alpha_\sigma = \pi_\sigma = 0$ for $\sigma < 1$. Rearranging, we have:

$$\pi_\sigma = \frac{\pi_{\sigma-1} - \alpha_{\sigma-L} \cdot \pi_{\sigma-L}}{(1-\alpha_\sigma)}. \tag{3.21}$$

The algorithm for determining π is now apparent. We let $\pi_1' = 1$ (an arbitrary value), calculate π_σ' for $\sigma = 2,3, \ldots, NL$ in order according to (3.21), and then let:

$$\pi_\sigma = \frac{\pi_\sigma'}{\sum_{i=1}^{NL} \pi_\sigma'} \tag{3.22}$$

for all σ. This yields π^G, and then J^G follows quickly from (3.15).

So, for our queue fill A/A model we have derived a relatively simple means of obtaining the state distribution and expected cost of a policy. We can find these values by recursively solving the $S-1$ *independent* linear equations (3.21) and then normalizing according to (3.22), so that the computational burden grows linearly with S. Compare this to the general method of solving a system of S simultaneous linear equations, for which the complexity is roughly proportional to M^3 [Pres86]. We see that this recursive method is relatively very efficient, and for small systems, we could use it to find the expected discarding cost of *every* possible policy, and then find the optimal policy by exhaustive search.

3.3.2 Computationally Efficient Value Determination

Recall that finding the relative values v_σ of a given policy, called the value determination step, is an integral part of the policy improvement algorithm for finding a stationary optimal policy for a given set of model parameters. But in order to determine the relative values of a policy, the policy must be irreducible, and we saw in section 3.3.1 that not all stationary policies for this model are irreducible. So in keeping with the conclusions of Chapter 2 regarding the search for an optimal policy, we will restrict ourselves to searching for an optimal irreducible policy. In fact, to satisfy the sufficient conditions for irreducibility of $\mathbf{P}^G$ given in section 3.3.1, we will only consider systems satisfying $\lambda > L$ and policies restricted as follows:

for each control set G^k at each time k and for every state σ with $L \le \sigma \le NL$, we do not have $\alpha_{\sigma-1} = \alpha_{\sigma-2} = \cdots = \alpha_{\sigma-(L-1)} = 0$. This restriction to irreducible queue fill A/A policies is represented in Figure 3-3.

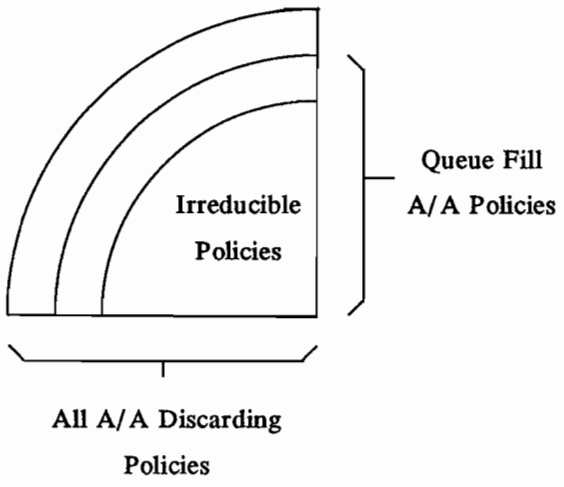


Figure 3-3. Irreducible Queue Fill A/A Policies

From our discussion of dynamic programming theory in Chapter 2, we are assured that the optimal policy in the innermost set is a stationary policy.

Recall from section 2.4 that the relative values and average cost per step for a policy G' are the elements of the vector $\mathbf{v}^{G'}$ and the value $J^{G'}$ satisfying:

$$J^{G'}\cdot\mathbf{e} + \mathbf{v}^{G'} = \boldsymbol{\xi}^{G'} + \mathbf{P}^{G'}\cdot\mathbf{v}^{G'}. \tag{3.23}$$

Then G' is optimal if and only if:

$$J^{G'}\cdot\mathbf{e} + \mathbf{v}^{G'} = \min_{G}\left[\boldsymbol{\xi}^{G} + \mathbf{P}^{G}\cdot\mathbf{v}^{G'}\right]. \tag{3.24}$$

As noted in section 2.4, the vector equation (3.23) has one degree of freedom in the relative values v_σ. So we may fix any one of these relative values at an arbitrary value (say 0) and then solve the set of $S=NL$ linear equations in the $S=NL$ remaining unknowns $J^{G'}$ and v_2 through v_{NL} by any convenient method. If a tractable closed form solution could be obtained,

we could try to find necessary or sufficient conditions for the optimality of G' in terms of the arrival statistics $\{\phi_d\}$ and costs $\{c_d\}$. We show next that there is a simple iterative solution for $\mathbf{v}^G$, but a tractable closed form solution has eluded us so far.

In what follows, we will omit the policy superscript from the variables with the understanding that a given policy has been specified. Examination of the vector equation (3.23) and the form of the transition matrix **P** (3.12) shows that we may write each of the $S=NL$ scalar equations as:

$$J + v_\sigma = \xi_\sigma + (1-\alpha_\sigma)\cdot v_{\sigma-1} + \alpha_\sigma \cdot v_{\sigma-1+L} \tag{3.25}$$

where $v_\sigma = 0$ for $\sigma < 1$ and $\sigma > NL$. Rearranging, we have:

$$v_{\sigma-1} = \frac{J + v_\sigma - \alpha_\sigma \cdot v_{\sigma-1+L} - \xi_\sigma}{1-\alpha_\sigma} \tag{3.26}$$

or, shifting the state index:

$$v_\sigma = \frac{J + v_{\sigma+1} - \alpha_{\sigma+1} \cdot v_{\sigma+L} - \xi_{\sigma+1}}{1-\alpha_{\sigma+1}}. \tag{3.27}$$

We can thus recursively calculate each relative value v_σ as a linear function of the two values $v_{\sigma+1}$ and $v_{\sigma+L}$. Also, for the last $L-1$ equations, this simplifies to:

$$v_\sigma = J + v_{\sigma+1} - \xi_{\sigma+1} \qquad NL-(L-1) \le \sigma \le NL-1. \tag{3.28}$$

If we now exercise our one degree of freedom by setting $v_{NL} = 0$, we see that v_σ is a linear function of J and only J for $NL-(L-1) \le \sigma \le NL-1$. This result then carries backward to smaller state values through the recursion equation (3.27) for $1 \le \sigma \le NL-L$, so that we have:

$$v_\sigma = a_\sigma \cdot J + b_\sigma \tag{3.29}$$

for some coefficients a_σ and b_σ for each σ from 1 to NL. Furthermore, it is easily verified from (3.27) and (3.28) that these coefficients can be found using the following recursions:

$$a_\sigma = \begin{cases} 0 & \text{for} \quad \sigma = NL \\ 1+a_{\sigma+1} & \text{for} \quad NL-(L-1) \le \sigma \le NL-1 \\ \dfrac{1+a_{\sigma+1}-\alpha_{\sigma+1}\cdot a_{\sigma+L}}{1-\alpha_{\sigma+1}} & \text{for} \quad 1 \le \sigma \le NL-L \end{cases} \tag{3.30}$$

and

$$b_\sigma = \begin{cases} 0 & \text{for} \quad \sigma = NL \\ b_{\sigma+1}-\xi_{\sigma+1} & \text{for} \quad NL-(L-1) \le \sigma \le NL-1. \\ \dfrac{b_{\sigma+1}-\alpha_{\sigma+1}\cdot b_{\sigma+L}-\xi_{\sigma+1}}{1-\alpha_{\sigma+1}} & \text{for} \quad 1 \le \sigma \le NL-L \end{cases} \tag{3.31}$$

However, we have one more equation which has not yet been used, namely the first equation of the entire linear system: $J+v_1 = \xi_1+v_L$. Recalling that ξ_1 is always 0 (we never discard when the queue is empty and a service completion is imminent), we have:

$$J = v_L - v_1 = (a_L - a_1)\cdot J + (b_L - b_1) \tag{3.32}$$

which yields:

$$J = \frac{b_L - b_1}{1 - (a_L - a_1)}. \tag{3.33}$$

Having found J, we can then use "back substitution" to find $v_\sigma = a_\sigma \cdot J + b_\sigma$.

This procedure for finding the average discarding cost J and relative values v_σ for a given policy may now be summarized.

1. Starting with $a_{NL} = b_{NL} = 0$, work backwards using recursions (3.30) and (3.31) to find a_σ and b_σ for all σ .
2. Find the value for J from (3.33).
3. For each σ, compute $v_\sigma = a_\sigma \cdot J + b_\sigma$.

This determination of relative values and average cost could then be used to test a given

policy for optimality from equation (3.24) or as the value determination step of the policy iteration algorithm for finding the optimal policy for a given set of model parameters. Also, just as with the algorithm for finding the state distribution, this is a recursive algorithm whose complexity is proportional to S, the number of states in the particular model. As such, it is much more efficient than general value determination step, which requires the solution of S simultaneous linear equations, a process whose complexity grows approximately as S^3.

3.3.3 Nested Threshold Policies

We now wish to consider the policy structure for a given phase l, so we will for the present identify each state σ or (n,l) by its queue fill n only, where $0 \le n \le N-1$. Using this simplified notation, we have the following policy structure for a given phase l: we specify a priority value d_n for each value of buffer fill n with the interpretation that if the queue fill is n, any arriving packet with delivery priority $d \le d_n$ will be discarded.

3.3.3.1 Nested Thresholds

Now suppose that these N priority values are non-decreasing in n, *i.e.*, that $d_n \le d_{n+1}$ for $0 \le n \le N-2$. We may express this monotonicity by saying that the priority values form a set of *nested priority thresholds*. Furthermore, we may then *equivalently* represent the discarding policy (for the given l) as a set of *nested queue fill thresholds* $\{n_\Delta: 0 \le \Delta \le D\}$, where n_Δ is defined as the smallest value of n for which $d_n \ge \Delta$. The interpretation of these new thresholds is that if the queue fill n at any arrival instant is n_Δ or more, we discard any arriving packet with delivery priority Δ or less. More precisely, representing the delivery priority of the arriving packet as d:

If there exists a Δ *such that* $n \ge n_\Delta$ and $d \le \Delta$, (3.34)

then discard the arriving packet.

Otherwise, admit the arriving packet.

This correspondence between d_n and n_Δ is illustrated below for a queue with size $N = 6$ and $D = 4$ different delivery priority values.

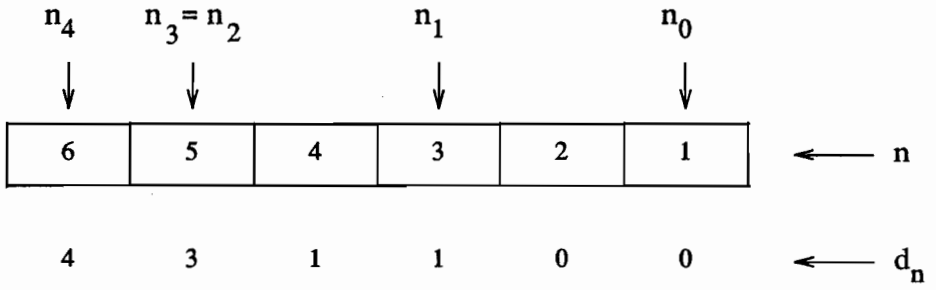


Figure 3-4. Example Priority Thresholds and Queue Fill Thresholds

Representation by queue fill thresholds is more convenient for visualization, but less convenient for implementation since (3.34) requires a search for an appropriate Δ.

Our intuition leads us to believe that any *optimal* policy should have this form for each phase l, just as we know that the optimal irreducible policy must be stationary. After all (we could say), how could we possibly gain by admitting lower priority packets as the queue becomes more full? We will define a *candidate policy* as one which meets one or more necessary conditions for optimality. Using this definition, stationary policies are a class of candidate policies, and our hypothesis is that nested threshold policies form another class of candidate policies. Note that restricting our search for an optimal policy to such candidate policies does not restrict the *problem*, only the region in which the solution to the problem must be found. We thus represent these nested threshold policies in Figure 3-5 as an oval within the irreducible policy region (rather than another arc) to emphasize this hypothesis that the optimal policy in the *entire* irreducible region (our search region) must be a nested threshold policy.*

* We could add another (intersecting) oval in the irreducible region to represent stationary policies. If nested threshold policies are indeed candidate policies, the optimal policy would lie in the intersecting region.

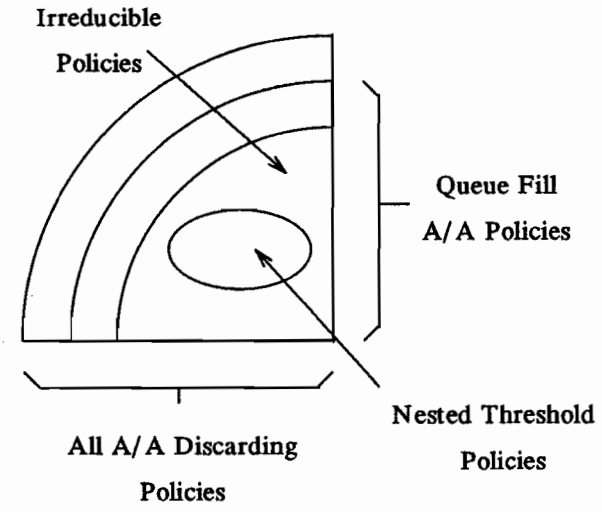


Figure 3-5. Nested Threshold Policies as Candidate Policies

Unfortunately, we have as yet been unable to verify our intuition with a mathematical proof. We will presently sketch out a possible approach to such a proof, but we first wish to briefly consider how many *candidate policies* (those not guaranteed to be sub-optimal) would exist if this hypothesis were true.

3.3.3.2 Number of Nested Threshold Policies

Since $d_{N-1} = D$ for $l > 1$ and $d_0 = -1$ for $l = 1$, we can choose a priority value for only $N-1$ of the N values of queue fill for any given value of phase l. Each of these chosen priority values is an integer in the range from 0 to D, and they must be non-decreasing to form a set of nested thresholds. So the number of nested threshold sub-policies (that is, policies for given l) for queue size N and number of priorities D (excluding the null or 0 priority) is the number of $(N-1)$-tuples, each element drawn from the set of $D+1$ elements $\{0, 1, \ldots, D\}$, such that the elements of the $(N-1)$-tuple are non-decreasing. We will call this value $Z(N-1, D+1)$. We first derive expressions for $Z(M,E)$, where M is any non-negative integer and E is any positive integer. We then show that the number of nested threshold policies is considerably smaller than the total number of possible policies for the 2-dimensional, queue fill A/A model.

To compute $Z(M,E)$, we consider the M choices in turn. For the first choice e_1, we may choose any value in the set $\{0, 1, \ldots, E-1\}$. Having made that choice, we may choose the second element from the set $\{0, 1, \ldots, e_1\}$. The third element then must be drawn from $\{0, 1, \ldots, e_2\}$, and so on until we get to the M^{th} element, which may be drawn from $\{0, 1, \ldots, e_{M-1}\}$, so that there are e_{M-1} possibilities for the last choice. This yields:

$$Z(M,E) = \sum_{e_1=0}^{E-1} \sum_{e_2=1}^{e_1} \cdots \sum_{e_{M-1}=1}^{e_{M-2}} e_{M-1}. \tag{3.35}$$

We can also easily see two recursive relationships. Suppose we make our first choice e_1 from $\{0, 1, \ldots, E-1\}$. Then the total number of possibilities for the last $M-1$ choices is just $Z(M-1,e_1)$. So we have:

$$Z(M,E) = \sum_{e_1=0}^{E-1} Z(M-1,e_1). \tag{3.36}$$

Now suppose that we know $Z(M,E-1)$ and wish to find $Z(M,E)$. The only choice for e_1 not included in $Z(M,E-1)$ is $e_1 = E-1$ which allows a total of $Z(M-1,E)$ choices for the remaining $M-1$ elements. So $Z(M,E) = Z(M,E-1) + Z(M-1,E)$. But we may apply the same reasoning to $Z(M-1,E)$ to get $Z(M,E) = Z(M,E-1) + Z(M-1,E-1) + Z(M-2,E)$. Continuing, we finally must consider $Z(1,E)$, which is obviously equal to E. So we can define $Z(0,E) := 1$ to get $Z(1,E) = Z(1,E-1) + Z(0,E)$ and:

$$Z(M,E) = \sum_{n=0}^{M} Z(n,E-1). \tag{3.37}$$

A comparison of the two recursions shows the symmetric nature of $Z(M,E)$, namely $Z(M,E) = Z(E-1,M+1)$. This symmetry is also apparent in the following tabulation of values of $Z(M,E)$.

TABLE 3-1. Values for Z(M,E)

	E							
M	1	2	3	4	5	6	7	8
0	1	1	1	1	1	1	1	1
1	1	2	3	4	5	6	7	8
2	1	3	6	10	15	21	28	36
3	1	4	10	20	35	56	84	120
4	1	5	15	35	70	126	210	330
5	1	6	21	56	126	252	462	792
6	1	7	28	84	210	462	924	1716
7	1	8	36	120	330	792	1716	3432
8	1	9	45	165	495	1287	3003	6435

This implies that $Z(D,N)$ is an equivalent representation for $Z(N-1,D+1)$, the number of sub-policies for a *given* phase for a system with queue size N and D priorities. So, the total number of possible nested threshold policies (ignoring for the moment the irreducibility requirement) is $Z(D,N)^L$. This is considerably smaller than $(D+1)^{NL-L}$, the total number of possible 2-dimensional model policies, but can still be quite large. For example, for $(N,L,D) = (3,3,3)$ we have $(D+1)^{NL-L} = 4^6 = 4096$ and $Z(D,N)^L = Z(3,3)^3 = 10^3 = 1000$.

3.3.3.3 Proving the Optimality of Nested Thresholds

We will now consider a possible approach to proving that any optimal policy for the 2-dimensional A/A packet discarding model must have the nested threshold form, that is, that the priority values d_n for a given phase l must be non-decreasing in queue fill n.

We begin by fixing a policy G' and considering two adjacent priority values with the same value of l. In terms of our overall state index $\sigma=nL+l$, these adjacent priority values for the same l can be labeled d_σ and $d_\sigma+L$. We will also use the specific form of the present discarding costs and the state transitions: $\xi_\sigma = \overset{d_\sigma}{\chi}$ and $\alpha_\sigma = 1-\overset{d_\sigma}{\Phi}$. Thus by fixing the policy, we have (according to the scalar form of equation 3.23 for this particular system):

$$J + v_\sigma = \overset{d_\sigma}{\chi} + \overset{d_\sigma}{\Phi}\cdot v_{\sigma-1} + (1-\overset{d_\sigma}{\Phi})\cdot v_{\sigma+L-1} \tag{3.38}$$

and

$$J + v_{\sigma+L} = \overset{d_{\sigma+L}}{\chi} + \overset{d_{\sigma+L}}{\Phi} \cdot v_{\sigma+L-1} + (1- \overset{d_{\sigma+L}}{\Phi}) \cdot v_{\sigma+2L-1} \tag{3.39}$$

for the two states under consideration.

One possible approach is proof by contradiction. Accordingly, we assume that the policy G' is optimal and that these two priority values are decreasing ($d_\sigma > d_{\sigma+L}$). The optimality assumption implies (from a scalar form of equation 3.23 with substitutions from equations 3.38 and 3.39):

$$\overset{d_\sigma}{\chi} + \overset{d_\sigma}{\Phi} \cdot v_{\sigma-1} + (1- \overset{d_\sigma}{\Phi}) \cdot v_{\sigma+L-1} = \min_{d \in \{0,1,\ldots,D\}} \left[\overset{d}{\chi} + \overset{d}{\Phi} v_{\sigma-1} + (1- \overset{d}{\Phi}) \cdot v_{\sigma+L-1} \right] \tag{3.40}$$

and

$$\overset{d_{\sigma+L}}{\chi} + \overset{d_{\sigma+L}}{\Phi} \cdot v_{\sigma+L-1} + (1- \overset{d_{\sigma+L}}{\Phi}) \cdot v_{\sigma+2L-1} = \tag{3.41}$$

$$\min_{d \in \{0,1,\ldots,D\}} \left[\overset{d}{\chi} + \overset{d}{\Phi} v_{\sigma+L-1} + (1- \overset{d}{\Phi}) \cdot v_{\sigma+2L-1} \right].$$

In particular, this implies:

$$\overset{d_\sigma}{\chi} + \overset{d_\sigma}{\Phi} \cdot v_{\sigma-1} + (1- \overset{d_\sigma}{\Phi}) \cdot v_{\sigma+L-1} \leq \overset{d_{\sigma+L}}{\chi} + \overset{d_{\sigma+L}}{\Phi} \cdot v_{\sigma-1} + (1- \overset{d_{\sigma+L}}{\Phi}) \cdot v_{\sigma+L-1} \tag{3.43}$$

and

$$\overset{d_{\sigma+L}}{\chi} + \overset{d_{\sigma+L}}{\Phi} \cdot v_{\sigma+L-1} + (1- \overset{d_{\sigma+L}}{\Phi}) \cdot v_{\sigma+2L-1} \leq \overset{d_\sigma}{\chi} + \overset{d_\sigma}{\Phi} v_{\sigma+L-1} + (1- \overset{d_\sigma}{\Phi}) \cdot v_{\sigma+2L-1}. \tag{3.44}$$

We now invoke the assumption that $d_\sigma > d_{\sigma+L}$, which implies that $\overset{d_\sigma}{\Phi} > \overset{d_{\sigma+L}}{\Phi}$. We can then rearrange (3.43) and (3.44) and combine them to get:

$$v_{\sigma+2L-1} - v_{\sigma+L-1} \leq \frac{\overset{d_\sigma}{\chi} - \overset{d_{\sigma+L}}{\chi}}{\overset{d_\sigma}{\Phi} - \overset{d_{\sigma+L}}{\Phi}} \leq v_{\sigma+L-1} - v_{\sigma-1}. \tag{3.45}$$

The idea would now be to show, perhaps using $v_\sigma = a_\sigma \cdot J + b_\sigma$ and the recursions for a_σ and b_σ, that this relationship (3.45) is impossible under the assumption of $d_\sigma > d_{\sigma+L}$. We could then conclude that if a given policy is optimal, we must have $d_\sigma \leq d_{\sigma+L}$ for all $\sigma \leq NL-L$, *i.e.*, that every optimal policy must be a nested threshold policy.

Unfortunately, it *is* possible to simultaneously satisfy (3.45) and $d_\sigma > d_{\sigma+L}$ with a sub-optimal policy. Consider the following counter-example, a system with specification:

$$(N,L,D) = (3,3,3)$$

$$\lambda = 1.5$$

$$(p_1,p_2,p_3) = (0.3,\ 0.6,\ 0.1)$$

$$(c_1,c_2,c_3) = (1,\ 2,\ 3)$$

$$(d_1,d_4,d_7) = (-1,\ 3,\ 1)$$
$$(d_2,d_5,d_8) = (1,\ 2,\ 3)$$
$$(d_3,d_6,d_9) = (1,\ 0,\ 3)$$

Using our recursive value determination algorithm, we find that this system has expected cost $J = 0.348$ and relative values (shifted to make $v_1 = 0$):

$$\begin{array}{lll} v_1 = 0.000 & v_4 = 0.900 & v_7 = 1.778 \\ v_2 = 0.117 & v_5 = 1.346 & v_8 = 2.330 \\ v_3 = 0.348 & v_6 = 1.489 & v_9 = 2.881 \end{array}$$

Then considering state $\sigma = 3$, we have $d_3=1$ and $d_6=0$ so that $d_\sigma > d_{\sigma+L}$. Also, we have $v_8 - v_5 = 0.984$, $v_5 - v_2 = 1.229$, and $\frac{\chi^1 - \chi^0}{\Phi^1 - \Phi^0} = c_1 = 1.0$, so that (3.45) is also satisfied.

Furthermore, the optimal policy for these model dimensions and parameters is:

$$(d_1,d_4,d_7) = (-1,\ 1,\ 1)$$
$$(d_2,d_5,d_8) = (0,\ 1,\ 3)$$
$$(d_3,d_6,d_9) = (0,\ 1,\ 3)$$

with optimal average cost $J^* = 0.251$. So, it is possible to have decreasing thresholds and still satisfy (3.45) for a sub-optimal policy.

We note, however, that in this policy there exists another pair of decreasing priority values $d_4 = 3$ and $d_7 = 1$ so that $d_\sigma > d_{\sigma+L}$ for σ=4. For this value of σ, inequality (3.45) does *not* hold. *

Thus we might want to try to prove that *if* inequality (3.45) holds for some pair of decreasing priority values $d_\sigma > d_{\sigma+L}$ in some policy G', then there must exist some *other* pair of decreasing priority values $d_{\sigma'} > d_{\sigma'+L}$ in G' for which (3.45) does *not* hold. This would be a more complicated proof and we have not yet been able to accomplish it. Of course, there are also many other possible approaches to proving that all optimal policies are nested threshold policies, and we remain hopeful that our future efforts will yield the desired proof.

3.3.4 The Default Policy

We now apply some of the analysis techniques detailed above to find the state distribution and expected cost for the *default policy* of discarding packets if and only if the queue is full. We will label this policy G_0. The policy G_0 corresponds to all priority thresholds (except the first one and the last $L-1$, which are fixed by the model) having value 0 (discard no packets), so we have:

$$\alpha_\sigma = \begin{cases} 1 & \text{for} \quad \sigma = 1 \\ 1-\overset{0}{\Phi} = 1-\phi_0 = \lambda/L & \text{for} \quad 2 \le \sigma \le NL-(L-1) \\ 0 & \text{for} \quad NL-(L-2) \le \sigma \le NL \end{cases} \tag{3.46}$$

* Incidentally, this is also the *only* policy for the specified system dimensions and parameters for which inequality (3.45) holds for a pair of decreasing priority values.

$$\xi_\sigma = \begin{cases} 0 & \text{for } 1 \le \sigma \le NL-(L-1) \\ \overset{D}{\chi} & \text{for } NL-(L-2) \le \sigma \le NL \end{cases} . \tag{3.47}$$

So the form of the average cost for the default policy is:

$$J^{G_0} = \overset{D}{\chi} \cdot \sum_{\sigma = NL-(L-2)}^{NL} \pi_\sigma . \tag{3.48}$$

Using the relation between J and $E[C]$ from (3.19) and the fact that $\overset{D}{\chi} = \sum \phi_d \cdot c_d = (\lambda/L) \cdot \sum p_d \cdot c_d$, we can express this result in terms of the normalized expected cost per arrival:

$$\frac{E[C]}{\sum_{d=1}^{D} p_d \cdot c_d} = \sum_{\sigma = NL-(L-2)}^{NL} \pi_\sigma . \tag{3.49}$$

Since the state distribution equations are functions only of λ/L, we conclude that *the normalized expected cost per arrival for the default policy depends only on load* λ, *queue capacity* N, *and maximum arrivals per service interval* L; *it is independent of all priority-related parameters*. We would be surprised if this were not the case, since the default policy totally disregards delivery priority information.

Unfortunately, even though all state equations are functions of the single parameter λ/L, the simple recursive formula (3.21) does not appear to yield a simple closed-form solution for the state distribution for general N and L. However, for the specific case of $L=2$ we do have the following simple relations:

$$\pi_\sigma' = \begin{cases} 1 & \text{for } \sigma = 1 \\ \dfrac{2}{2-\lambda} \cdot \left[\dfrac{\lambda}{2-\lambda} \right]^{\sigma-2} & \text{for } 1 < \sigma \le 2N-1 \\ \left[\dfrac{\lambda}{2-\lambda} \right]^{2N-2} & \text{for } \sigma = 2N \end{cases} \tag{3.50}$$

We can use simple induction to prove the relations (3.50). First note that (3.50) clearly holds for $\sigma'=1$ through $\sigma'=3$, since from (3.21) and (3.50):

$$\pi_2' = \frac{1}{1-\alpha_2}\cdot\pi_1' = \frac{2}{2-\lambda} \tag{3.51}$$

and

$$\pi_3' = \frac{1}{1-\alpha_3}\cdot\pi_2' - \frac{\alpha_1}{1-\alpha_3}\cdot\pi_1' = \frac{2}{2-\lambda}\cdot\left[\frac{2}{2-\lambda} - 1\right] = \frac{2}{2-\lambda}\cdot\frac{\lambda}{2-\lambda}. \tag{3.52}$$

We then let $4 \le \sigma < 2N$, assume that the equality holds for $\sigma-1$ and $\sigma-2$ and show that it holds for σ. Then from (3.21), (3.50) and (3.51) we have:

$$\begin{aligned}\pi_\sigma' &= \frac{1}{1-\alpha_\sigma}\cdot\pi_{\sigma-1}' - \frac{\alpha_{\sigma-2}}{1-\alpha_\sigma}\cdot\pi_{\sigma-2}' \\ &= \frac{2}{2-\lambda}\cdot\left[\frac{2}{2-\lambda}\cdot\left[\frac{\lambda}{2-\lambda}\right]^{\sigma-3}\right] - \frac{\lambda}{2-\lambda}\cdot\left[\frac{2}{2-\lambda}\cdot\left[\frac{\lambda}{2-\lambda}\right]^{\sigma-4}\right] \\ &= \frac{2}{2-\lambda}\cdot\left[\frac{\lambda}{2-\lambda}\right]^{\sigma-3}\cdot\left[\frac{2}{2-\lambda} - 1\right] \\ &= \frac{2}{2-\lambda}\cdot\left[\frac{\lambda}{2-\lambda}\right]^{\sigma-2}.\end{aligned} \tag{3.53}$$

Finally, for the last state we have:

$$\begin{aligned}\pi_{2N}' &= \frac{1}{1-\alpha_{2N}}\cdot\pi_{2N-1}' - \frac{\alpha_{2N-2}}{1-\alpha_{2N}}\cdot\pi_{2N-2}' \\ &= \frac{2}{2-\lambda}\cdot\left[\frac{\lambda}{2-\lambda}\right]^{2N-3} - \frac{\lambda}{2}\cdot\left[\frac{2}{2-\lambda}\cdot\left[\frac{\lambda}{2-\lambda}\right]^{2N-4}\right] \\ &= \left[\frac{\lambda}{2-\lambda}\right]^{2N-3}\cdot\left[\frac{2}{2-\lambda} - 1\right] = \left[\frac{\lambda}{2-\lambda}\right]^{2N-2}.\end{aligned} \tag{3.54}$$

We note that for $L=2$, equation (3.49) reduces to $\frac{E[C]}{\sum_d p_d \cdot p_d} = \pi_{2N}$. We can get an expression for this by obtaining a closed-form expression for the normalizing sum (3.22). We

first examine the partial sum:

$$\sum_{\sigma=2}^{2N-1} \pi_\sigma{}' = \frac{2}{2-\lambda} \cdot \sum_{\sigma=2}^{2N-1} \left[\frac{\lambda}{2-\lambda}\right]^{\sigma-2} = \frac{2}{2-\lambda} \cdot \sum_{j=0}^{2N-3} \left[\frac{\lambda}{2-\lambda}\right]^{j}. \tag{3.55}$$

If $\lambda = 1$, $\lambda/(2-\lambda) = 1$, this partial sum is just $4N-4$, $\pi'_{2N} = 1$, the total normalizing sum is $4N-4+2 = 4N-2$, and the normalized expected cost per arrival (equivalent to π_{2N}) is $1/(4N-2)$.

If $\lambda \neq 1$, we have:

$$\sum_{\sigma=2}^{2N-1} \pi_\sigma{}' = \frac{2}{2-\lambda} \cdot \frac{\left[\frac{\lambda}{2-\lambda}\right]^{2N-2} - 1}{\frac{\lambda}{2-\lambda} - 1} = \frac{2}{2(\lambda-1)} \cdot \left[\left[\frac{\lambda}{2-\lambda}\right]^{2N-2} - 1\right]. \tag{3.56}$$

We then have:

$$\begin{aligned} \sum_{\sigma=1}^{NL} \pi_\sigma{}' &= 1 + \left[\frac{\lambda}{2-\lambda}\right]^{2N-2} + \frac{2}{2(\lambda-1)} \cdot \left[\left[\frac{\lambda}{2-\lambda}\right]^{2N-2} - 1\right] \\ &= \frac{2(\lambda-2)}{2(\lambda-1)} + \frac{2\lambda}{2(\lambda-1)} \cdot \left[\frac{\lambda}{2-\lambda}\right]^{2N-2} \\ &= \frac{1}{\lambda-1} \cdot \left[\lambda - 2 + \lambda \left[\frac{\lambda}{2-\lambda}\right]^{2N-2}\right]. \end{aligned} \tag{3.57}$$

So we have the following expression for the normalized expected cost per arrival for the default policy G_0 for $L = 2$.

$$\frac{E[C]}{\sum_d p_d \cdot c_d} = \pi_{2N} = \begin{cases} \dfrac{1}{4N-2} & \text{for } \lambda = 1 \\ \dfrac{(\lambda-1) \cdot \left[\frac{\lambda}{2-\lambda}\right]^{2N-2}}{\lambda + \lambda \left[\frac{\lambda}{2-\lambda}\right]^{2N-2} - 2} & \text{for } \lambda \neq 1 \end{cases} \tag{3.58}$$

We would also like to know if and when the default policy is optimal among all policies of

the 2-dimensional queue fill A/A model being considered in this chapter. However, the optimality of a policy is difficult to characterize analytically unless a reasonably simple closed form expression for the relative values v_σ and average cost J can be found. It would suffice to find such a simple closed form expression for the a_σ and b_σ values in $v_\sigma = a_\sigma \cdot J + b_\sigma$, rather than the recursive relations (3.30) and (3.31). We have as yet been unable to find such simple expressions.

However, we do know that the default policy is sometimes optimal among the policies considered in this chapter. Note that the default policy fits the description of a nested threshold policy, so this result does not contradict the hypothesis that all optimal policies are nested threshold policies. For example, for $(N,L,D) = (3,2,2)$, we have found the optimal policy (by exhaustive search) for each set of parameters for which λ is in the range from 1.0 to 1.9 in 0.1 increments, p_1 is in the range from 0.1 to 0.9 in 0.1 increments, and c_1 is in the range from 1.0 to 9.0 in 1.0 increments, holding c_2 at 10.0. Of the 810 such parameter combinations, the default policy is optimal for 67 of them. However, the largest value of λ for which the default policy is optimal is $\lambda = 1.4$. Furthermore, with $\lambda = 1.4$ there is only one combination for which the default policy is optimal, namely $p_1 = 0.9$ and $c_1 = 9.0$. Based on this example, we might speculate that the default policy is optimal only for "small" loads or "small" cost differences. We leave as an open item the task of verifying and quantifying this speculation.

3.4 Performance Advantage of Optimal Discarding Policies

Optimal discarding policies are of practical interest only if they perform significantly better than sub-optimal policies. We illustrate here the superiority of optimal discarding for queue fill A/A systems for some simple cases.

3.4.1 A Speech Traffic Example

We first consider an example motivated by the work in [Petr89], which also provides a good illustration of the assignment of discarding costs. We consider a network carrying primarily speech traffic which has been separated into four different classes according to the basic speech production model for the given segment of speech: background noise, fricatives (noise-like sounds), voiced sounds (*e.g.*, vowels), and "other." We showed in [Petr89] that, with the proper processing at the transmitter and receiver, these four classes exhibit different sensitivities to packet loss, judged by the subjective quality of the received speech. Specifically, when losses were restricted to be from a single class only, the received speech was indistinguishable from the source speech for packet losses of roughly 50% for background noise (priority 1), 8% for fricatives (priority 2), 6% for voiced (priority 3) and 4% for other (priority 4).* On this basis, we can now assign a discarding cost to each class (delivery priority) which is inversely proportional to the maximum packet loss as shown in Table 3-2. Both a high and low background noise level were tested, with the priority distributions as shown in Table 3-2.

TABLE 3-2. Cost and Arrival Distributions for a Speech System

	Priority d			
	1	2	3	4
c_d	50.0	12.5	16.7	25.0
p_d (high noise)	0.38	0.09	0.35	0.18
p_d (low noise)	0.24	0.20	0.36	0.20

Using these distributions, we now consider a system with $N = 3$ and $L = 2$. For such a system, Figure 3-6 shows the advantage of priority discarding using an optimal queue fill A/A system, in terms of normalized expected discarding cost per arrival, over the default discarding

* The tolerance for voiced speech was actually less in [Petr89], but subsequent work leads us to believe that 6% is achievable.

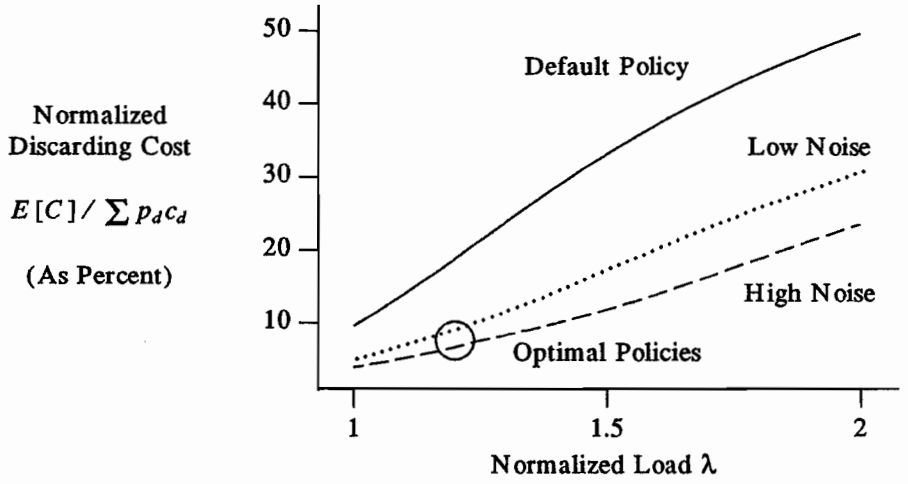


Figure 3-6. Performance Advantage for a Speech System

policy for the first queue encountered by this speech traffic[+] as a function of load λ.

In this figure, the discarding policy has been optimized for each point, that is, each combination of load and arrival distribution. We note as an aside that every optimal policy in this and every other example in this chapter is a nested threshold policy. We see that the performance penalty (discarding cost) of the optimal policy is approximately half that of the default policy at all loads, a clear indication of the superiority of optimal discarding.

3.4.2 Robustness to Parameter Values

To illustrate that the advantage of priority discarding using an optimized queue fill A/A policy is robust across different arrival and cost distributions, we now consider a slightly simpler system with $(N,L,D) = (3,2,3)$. We first consider the skewed cost function $(c_1,c_2,c_3) = (1,2,10)$. We further consider two skewed arrival distributions, the first having $(p_1,p_2,p_3) = (0.1, 0.3, 0.6)$ and the second having $(p_1,p_2,p_3) = (0.6, 0.3, 0.1)$. We plot the

+ Subsequent queues would have different arrival probabilities depending on the discarding at the first queue.

normalized expected discarding cost per arrival as a function of load for the default and optimal queue fill A/A policies for each case in Figure 3-7.

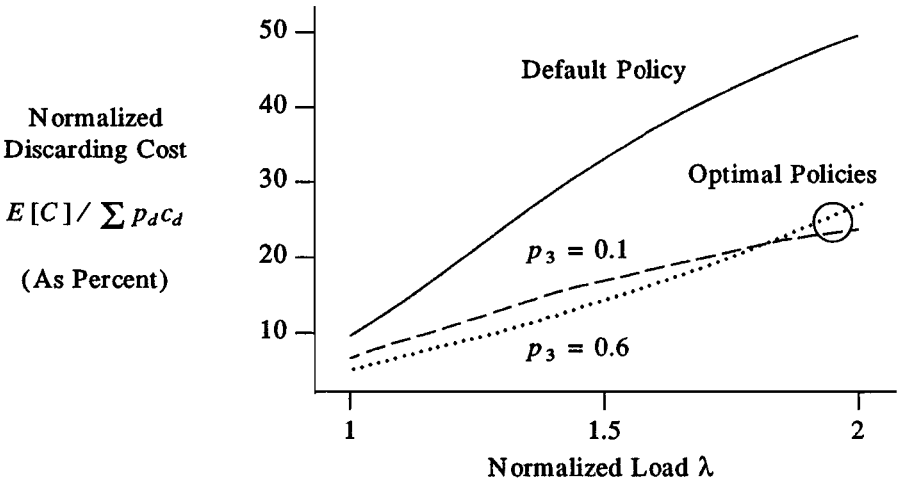


Figure 3-7. Performance Advantage for $(c_1,c_2,c_3) = (1, 2, 10)$

As in Figure 3-6, the optimization has been performed for every combination of load and arrival distribution. Note that, as predicted by equations (3.21), (3.46) and (3.49), the curve for the default policy does not depend on the priority parameters, and in fact is unchanged from from Figure 3-6. Again, the performance advantage of optimal discarding is roughly 2-to-1 across all loads. Figures 3-8 and 3-9 are similar plots, but for the linear cost function $(c_1,c_2,c_3) = (1,2,3)$ and the oppositely skewed cost function $(c_1,c_2,c_3) = (1,9,10)$, respectively.

We conclude from these last two plots that there may be some variation in the magnitude of the performance advantage with arrival and cost distributions, but that the advantage is always significant.

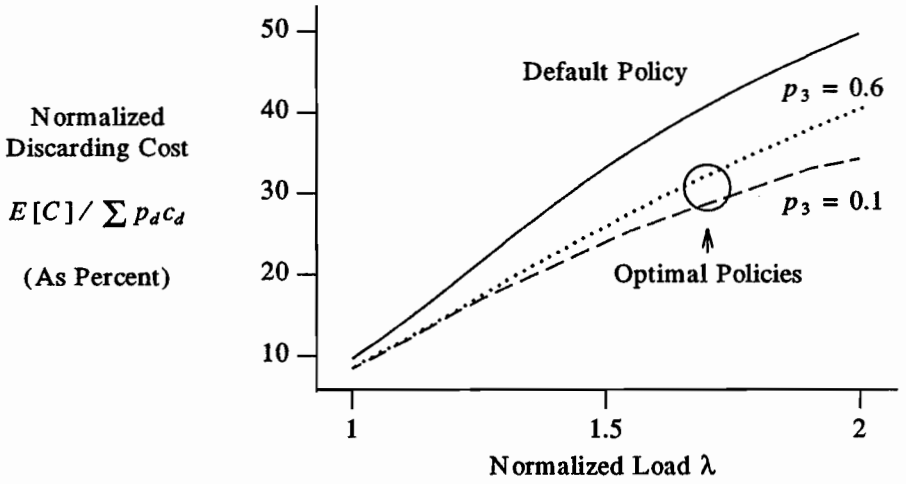


Figure 3-8. Performance Advantage for $(c_1,c_2,c_3) = (1, 2, 3)$

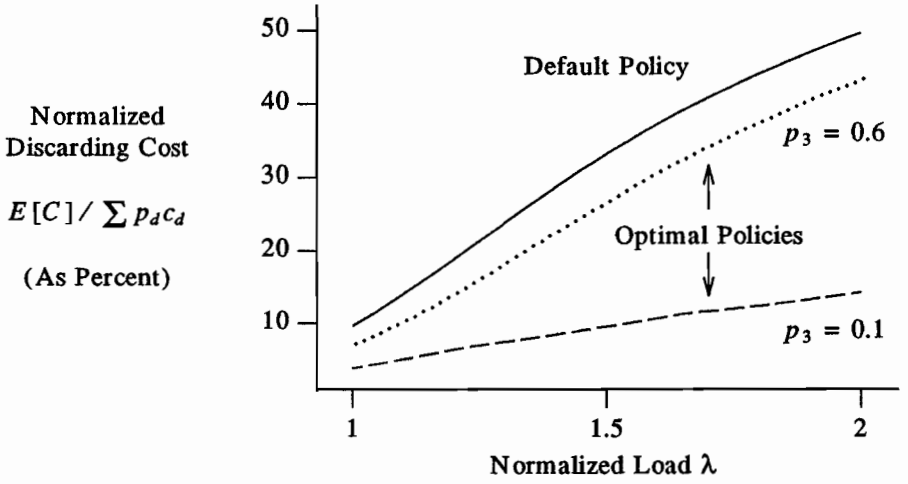


Figure 3-9. Performance Advantage for $(c_1,c_2,c_3) = (1, 9, 10)$

3.4.3 Effect of Queue Capacity N

We next consider the effect of changing the queue capacity N, keeping all other dimensions and parameters fixed. The queue capacity is often the only system dimension or parameter that is under the control of a system designer.

We consider a simple system with a maximum of two arrivals per service interval ($L=2$)

and with two delivery priorities ($D=2$). We choose a "balanced" set of parameters $(p_1,p_2) = (0.5, 0.5)$ and $(c_1,c_2) = (1,2)$. In Figure 3-10, we plot the normalized expected cost versus load for the default and optimal queue fill A/A discarding policies for two values of queue capacity: $N=2$ and $N=5$.

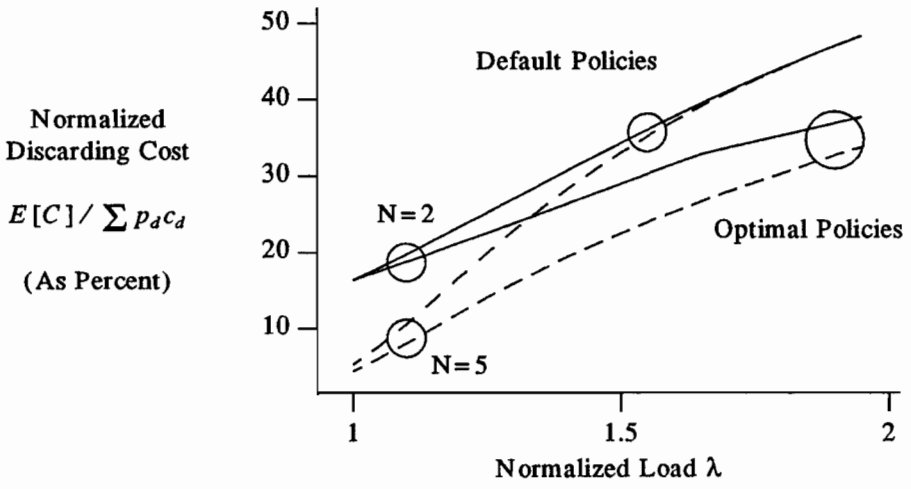


Figure 3-10. Effect of Queue Capacity N

We see from the figure that there is always an advantage in using the larger queue, but this advantage diminishes as the load increases. This result is consistent with the results in [Li89], in which it was concluded that queuing is most effective for buffering *transient* overload conditions and relatively ineffective for sustained periods of high overload. We note here, however, that the advantage diminishes *less* quickly with load if the optimal policy is used instead of the default policy. Finally, we note that the default policy for $N=5$ can perform better than the optimal policy for $N=2$, but only for low loads.

3.5 Summary of Results

We conclude with a list of the new contributions presented in this chapter, at the same time identifying some of the remaining open issues.

1. We developed a 2-dimensional analysis model for queue fill A/A systems, that is, A/A systems for which the queue state information is restricted to total queue fill.

2. We derived sufficient conditions for the irreducibility of any policy under this model. By the results of Chapter 2, restricting our search to optimal policies satisfying these conditions guarantees that a stationary policy will be optimal.

3. We derived a simple recursive method for obtaining the state distribution and expected discarding cost for a given discarding policy for any model dimensions and parameters. The computational complexity of this algorithm is linear in S, the number of states in a particular model. A more general solution for the state distribution would have complexity roughly proportional to S^3.

4. We derived a simple recursive method for obtaining the relative values and expected discarding cost for a given discarding policy for any model dimensions and parameters. These values then allow one to check for optimality for the given policy. The method could also be used as the value determination step in the policy improvement algorithm for finding the optimal policy for a given set of model dimensions and parameters. The recursive value determination algorithm again has complexity which is linear in S, comparing very favorably with the S^3 complexity of the more general solution method.

5. We hypothesized that all optimal policies are nested threshold policies, calculated the number of such policies, and sketched a possible approach to proving the hypothesis. The actual proof is left as an open item.

6. We demonstrated that the normalized expected discarding cost per arrival for the *default policy* of discarding packets if and only if the queue is full is independent of any priority information, as expected. We also derived a closed-form expression for this normalized cost for the default policy for the particular case of $L=2$. Since this default policy is

almost universally applied today, this result gives a performance baseline for priority packet discarding systems.

7. We demonstrated that the default policy is an optimal queue fill A/A policy for $(N,L,D) = (3,2,2)$ only for low loads or small differences in the discarding costs. Again, since the default policy is almost universally used today, this result indicates that continuing to use the default policy as an overload control mechanism will often result in sub-optimal performance. Demonstrating this result in general is an open item.

8. We demonstrated that the performance advantage of optimal A/A discarding relative to default discarding is significant for large variations in model parameters. This result indicates that optimal discarding policies should be of practical as well as theoretical interest.

9. We demonstrated that there is always a performance advantage of using a larger capacity queue (larger N), but that this advantage diminishes with increasing load. So, for heavily overloaded systems, increasing the queue capacity is of little value. However, at low overloads the default policy for a larger queue capacity may perform better than the optimal queue fill A/A discarding policy. In some middle region of overload (perhaps quite narrow), the system designer can choose between a larger system with the default discarding policy or a smaller system with an optimal queue fill A/A policy and get approximately the same performance.

3.6 References

[Gerl80] M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey," *IEEE Transactions on Communications*, Vol. COM-28, No. 4, April 1980.

[Kamo81] F. Kamoun, "A Drop and Throttle Flow Control Policy for Computer Networks," *IEEE Transactions on Communications*, Vol. COM-29, No. 4, April 1981.

[Lam79] S. S. Lam and M. Reiser, "Congestion Control of Store-and-Forward Networks by Input Buffer Limits -- An Analysis," *IEEE Transactions on Communications*, Vol. COM-27, No. 1, January 1979.

[Li89] S. Q. Li, "Study of Information Loss in Packet Voice Systems," *IEEE Transactions on Communications*, Vol. COM-37, No. 11, November 1989.

[Muis86] R. W. Muise, T. J. Schoenfeld, G. H. Zimmerman, "Experiments with Wideband Packet Technology," Proceedings of the 1986 International Zurich Seminar on Digital Communications, March 1986.

[Petr89] D. W. Petr, L. A. DaSilva, V. S. Frost, "Priority Discarding of Speech in Integrated Packet Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 5, June 1989.

[Pres86] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1986.

Chapter 4

Optimal FS/Q Packet Discarding

CONTENTS

Chapter 4

LIST OF FIGURES

LIST OF TABLES

Chapter 4

Optimal FS/Q Packet Discarding

We now turn our attention to the analysis of the FS/Q structural class of priority packet discarding algorithms, that is, those for which the discarding interval is the Fixed Service interval, and the discarding set is any Queued packets. We first discuss our motivation for analyzing systems of this class in section 4.1. In section 4.2, we develop an analysis model for these systems which is consistent with the general model of Chapter 2. We then (section 4.3) formulate the problem in dynamic programming terms and generate some results for general FS/Q policies and compare the performance of optimal FS/Q policies to the performance of the optimal queue fill A/A policies of Chapter 3 and the default policy. In section 4.4 we further analyze FS/Q systems constrained to retain at most $B=1$ packet, obtaining more concrete results for this particular case. We close the chapter with the summary of results in section 4.5.

4.1 Motivation for FS/Q Discarding Analysis

We would in general expect the different structural classifications for discarding algorithms of Chapter 2 to differ both in performance and in implementation complexity for the same system dimensions and parameters. For example, we would expect the A/A systems of Chapter 3 to be very easy to implement, but a system designer would like to know if a structurally different discarding algorithm could perform better. If so, a system designer might consider a more complex implementation in order to improve overall system performance. However, the decision will depend on the balance between the extra costs associated with the

more complex implementation and the magnitude of the performance advantage. The analysis in this chapter will give us a means of determining the performance of optimal FS/Q discarding policies and comparing this to the performance of optimal queue fill A/A policies analyzed in Chapter 3. An analysis of implementation costs is left to others with expertise in that area.

Of the two structural classifications that are allowed to discard any queued packets, we have chosen FS/Q systems rather than A/Q systems (discarding interval is Arrival interval) in order to make the analysis tractable. We have seen in Chapter 3 that analyzing arrival intervals involves dividing the service interval into subintervals corresponding to possible arrivals. The analysis is then complicated because the subinterval associated with a service completion behaves differently (in terms of transition probabilities) than all the other subintervals. This requires treating the subintervals of a service interval as "phases" and results in a multi-dimensional analysis model. This was tractable in Chapter 3 because we restricted queue state information to queue fill. This restriction was reasonable for systems allowed to discard only arriving packets, but it is much less appealing when the system is allowed to discard packets already in the queue, since we would like to know the delivery priorities of the queued packets in order to decide *which* packets to discard. Because of this added complexity in the description of the queue state, we have chosen to avoid the added phase dimension of the overall system state by considering systems for which the discarding interval is the fixed service interval, that is, FS/Q systems. We will see that the analysis is far from trivial even with this choice.

We note that the A/A systems of Chapter 3 and the FS/Q systems in this chapter are in diagonally opposite quadrants of the classification matrix of Table 2-1; that is, they differ in both classification dimensions, control interval and discarding set. This means that, although we will be able to compare the performance of the two classes, we will not be able to see the

performance effect of changing a single dimension, such as the discarding set. So we will not be able to verify our intuition that discarding any queued packets is superior in performance to discarding only arriving packets, all else being equal. There is, however, an advantage to leaving no dimension unchanged. By developing models for A/A and FS/Q systems, we will have developed analytical techniques that address each of the two possibilities in each of the two classification dimensions. This experience in each of the two dimensions should prove helpful in future analyses.

4.2 Specific FS/Q Analysis Model

We again build on the foundation of the general analysis model introduced in Chapter 2. We first discuss the effect of the discarding interval on the model parameters, and then discuss the effect of the discarding set on the model state description and possible policies.

4.2.1 Model Parameters

In section 2.3 we defined p_d as the probability that an arriving packet has delivery priority d, so that $\{p_d : 1 \le d \le D\}$ is a probability distribution. We also defined c_d as the cost of discarding a priority d packet, so that $\{c_d : 1 \le d \le D\}$ is the cost function. Recall from section 2.3.1 that we may, without loss of generality, assume that the cost function is strictly increasing. For the current model in which the discarding decision interval is the fixed service interval, we must also specify the distribution of the number of arrivals in a service (discarding) interval. In Chapter 3, the model definition constrained this to be a binomial distribution, but we have no such constraint here. We will thus specify a completely general number-of-arrivals probability distribution $\{q_l : 0 \le l \le L\}$, where q_l is the probability of l arrivals in a service interval. With this general distribution for the number of arrivals per service interval, the expected number of arrivals is $\lambda = \sum_l l \cdot q_l$.

4.2.2 Specifying Queue State

In considering how to specify the queue state, we first note that since the discarding set is *any* queued packets, including the most recent arrivals, these most recent arrivals need not be treated separately from the previously queued packets. Consequently, we may add packets to the queue as they arrive and then define the *state* of the system to be the state of the queue (*including* the most recent arrivals) *at the discarding instant,* that is, just before discarding takes place.

We choose here to represent the queue state as the ordered set of delivery priorities of the queued packets. More specifically, let n be the number of queued packets (including the most recent arrivals and excluding the packet in service) at a discarding instant. Then the state identifier σ may be represented as a vector of appropriate length (discussed below):

$$\sigma = (0 \cdots 0\ d_{n(\sigma)}\ \ d_{n(\sigma)-1} \cdots d_2\ \ d_1) \tag{4.1}$$

where a 0 represents an empty queue slot, $n(\sigma)$ is the queue fill for state σ, $d_{n(\sigma)}$ is the delivery priority of the packet which last joined the queue (furthest from the server), and d_1 is the delivery priority of the packet that has been queued the longest (closest to the server). We will often refer to the packets furthest from the server as the "leftmost" packets and those closest to the server as the "rightmost" packets. We will have no need to consider "null" packets as we did in Chapter 3, so the delivery priorities take values in the range $1 \leq d_j \leq D$. Note that we are re-using the subscripted d notation: in Chapter 3, d_σ represented the control decision (maximum rejected priority value) for state σ, but we are using d_j here as part of the state description to represent the delivery priority of the packet in position j of the queue.

Note also that this is a *complete description* of the queue state for an FS/Q system with expected discarding cost as the performance metric, in the sense that no queue state information which could possibly be relevant has been omitted. Also note that it is a much

more detailed description than the simple queue fill description used in Chapter 3, and so the number of possible queue states is much larger. In fact, for a given queue fill n, the number of possible states for this queue fill is D^n. The total number of possible states is then D^n summed over all possible values of n. The minimum value of n is obviously 0, but we must be more careful in determining the maximum value, as we now show.

4.2.3 Effect of Discarding Instant

We saw in Chapter 2 that if we are to avoid discarding packets without a deliberate decision to do so, we must reserve some of the queue positions for the arrivals between discarding decisions. This means that there is an upper limit B on the number of packets which can be *retained* as a result of any discarding decision, that is, the queue fill immediately after the decision is implemented. For a system that uses the arrival intervals as the control interval, such as the A/A systems in Chapter 3, only one position needs to be reserved, so we effectively had $B = N - 1$. For systems using the fixed service interval as the control interval, such as the FS/Q systems currently being considered, the exact relationship of B to the queue capacity N depends on the precise timing of the discarding decision, as we will presently show. For now we note that, given B, the maximum value of n (the queue fill just *before* a discarding decision instant) is $B + L - 1$ since there will be at most L arrivals and exactly one service completion before the next discarding instant. So the length of the state vector in (4.1) is $B + L - 1$ and the number of possible queue states is:

$$S = \sum_{n=0}^{B+L-1} D^n. \tag{4.2}$$

We now consider two possible instants for the discarding action: just before the service completion, and just after service completion. The timing of events for both cases is illustrated in Figure 4-1, where we have designated the service interval as τ.

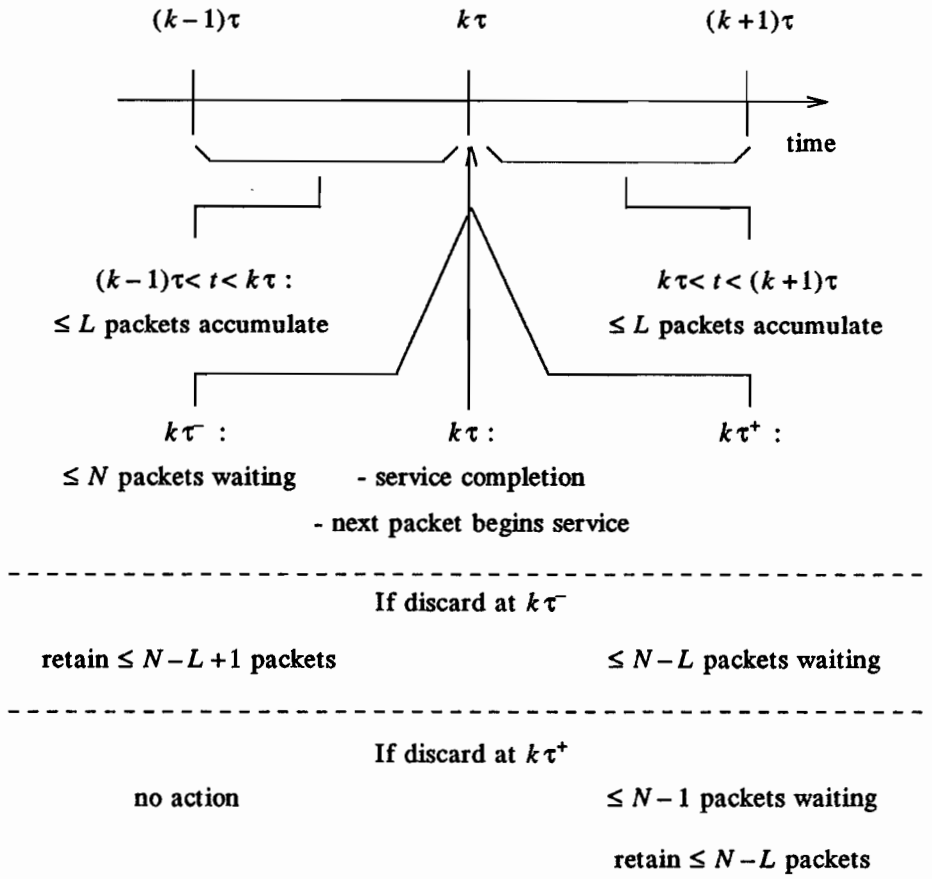


Figure 4-1. Event Timing for FS/Q Model

4.2.3.1 Discarding Before Service Completion

As indicated in Figure 4-1, if we discard just before the service completion (at $k\tau^-$), we may retain up to $B = N-L+1$ packets since this will leave $N-L$ packets in the queue *after* the service completion, *i.e.*, L empty queue slots to accommodate the maximum number of arrivals before the next discarding decision. This will guarantee that the queue does not overflow while we are waiting for the next discarding instant. For this case of discarding just before the service completion, the length of the state vector in (4.1) is $B+L-1 = (N-L+1)+L-1 = N$ and the number of possible states is:

$$S = \sum_{n=0}^{B+L-1} D^n = \sum_{n=0}^{N} D^n. \tag{4.3}$$

We also note that the smallest sensible value of B is one, so for systems that discard before service completion, we require $N \geq L$.

4.2.3.2 Discarding After Service Completion

Figure 4-1 also illustrates the case of discarding just *after* the service completion (at $k\tau^+$). In this case, we may retain up to $B = N-L$ packets, leaving L queue positions for the maximum number of arrivals before the next service completion. The maximum number of waiting packets just before the service completion is N, and the maximum number of waiting packets just before the discarding decision is $N-1$. For this case of discarding just after the service completion, the length of the state vector in (4.1) is thus $N-1$ and the number of possible states is:

$$S = \sum_{n=0}^{B+L-1} D^n = \sum_{n=0}^{N-1} D^n. \tag{4.4}$$

Here, in order to have $B \geq 1$, we require $N \geq L+1$.

4.2.3.3 Relationship of the Two Discarding Instants

We will assume in our analyses that no time is required to make and implement a discarding decision. In this case, discarding before the service completion would have an advantage because it is allowed to retain $B = N-L+1$ packets, instead of the $B = N-L$ packets which can be retained if we discard after the service completion.

However, this assumption of zero implementation time may not be reasonable for a practical implementation, especially if the service interval is extremely short due to high bandwidth transmission links or short packet lengths or both. From this point of view, discarding after the service completion has an advantage. Since the maximum queue fill

immediately following a service completion is $N-1$, it is only necessary that the discarding decision be made and implemented before the *second* packet arrives after the service completion to avoid uncontrolled packet loss. For a system that discards just before the service completion, it is necessary that the *single* packet arrival probability within the decision implementation time be negligibly small. If this is not a reasonable assumption, we would have to reserve another queue position for an arrival in the decision implementation interval, reducing the effective value of B from $N-L+1$ to $N-L$, the same as for the case of discarding after the service completion.

We now show that we can perform a detailed analysis for discarding just after the service completion, and then make only slight modifications to analyze a system that discards before the service completion. Suppose we wish to analyze a system (call it System 1) with dimensions (N_1,L_1,D_1) which discards packets just *before* the service completion. We would then have $B = N_1-L_1+1$ and $S = \sum_{n=0}^{N_1} D_1^n$. We first note that analyzing another system (call it System 2) with dimensions (N_2,L_2,D_2) which discards packets just *after* the service completion yields $B = N_2-L_2$ and $S = \sum_{n=0}^{N_2-1} D_2^n$. So we can get the same values of B and S for the two analyses if we make $(N_2,L_2,D_2) = (N_1+1, L_1,D_1)$.

We next claim that the two systems differ in their behavior only for the control option of discarding all packets (retaining zero packets), and that this option is guaranteed to be suboptimal for System 1 (discarding just before the service completion). That is, retaining zero packets is not a candidate option for any non-empty queue state, though it is the only possible option for the empty queue state.

Consider first the retain zero packet discarding option for System 1. Because the fixed service intervals are kept synchronized by transmitting "null" packets whenever the queue is

empty when the server is ready for another packet, we see that the next state for this option is identical to the next state for *any* option that retains one packet. In both cases, the next state is represented by the packets that arrive in the next service interval. That is, the *transition vectors* are the same. As discussed in Chapter 2, any policy that does not minimize the *present cost* in such a situation is guaranteed to be suboptimal. Since the option of retaining zero packets always has a higher present cost than any option which retains one packet, the option of retaining zero packets is always suboptimal for System 1.

For System 2 (discarding after the service completion), the next state if one packet is retained can again be represented by the arrivals in the following service interval. However, if zero packets are retained, the first arrival in the next service interval will move immediately into the server and not be a part of the next state. That is, the retain zero packet option will have a *different* transition vector. So the next state for the option of retaining zero packets is generally different from the next state for retaining one packet, and there is no reason to believe that the option of retaining zero packets is always suboptimal. We conclude that any general analysis of System 2 must consider the option of retaining zero packets.

Suppose now that System 1 and System 2 are in the same state and both systems take the option of retaining the same single packet. We note from the previous discussion that the behavior of the two systems is identical: if they are given identical inputs, the next state of the two systems will be the same, *i.e.,* the transition vectors for the two systems is the same. In fact, the two systems will behave identically (have the same transition vector) for *any* decision to retain the same set of packets, if at least one packet is retained. The reason is that the difference in the two systems is only a difference in phasing: the events between discarding instants are the same, but occur in a different order. For both systems, if one or more packets are retained, the retained packet closest to the server will move into the server prior to the next discarding instant and the remaining packets will be joined by the arrivals in the service

interval to form the next state.

We see from this discussion that System 1 and System 2 differ only in the precise expression for B (and thus S) and in their behavior (the transition vectors) for the case of retaining zero packets. We conclude that we may perform an exact analysis of System 1 (discarding before service completion) by performing an analysis of System 2 (discarding after service completion) with the following restrictions.

1. We must make $(N_2,L_2,D_2) = (N_1+1, L_1, D_1)$.
2. To obtain results about System 1, we must force the transition vector for the retain zero packet option to be the same as the transition vector for the retain one packet option. The retain zero packet option will then be sub-optimal for all but the empty queue state (for which it is the only option).

Based on this conclusion, we will hereafter consider only systems that discard after the service completion (such as System 2), unless specifically noted. In our analysis of these systems, we will calculate the transition vector for the retain zero packet option as it should be for a system discarding after the service completion, with the understanding that this transition vector could be modified to yield a valid analysis for a system (such as System 1) that discards before the service completion.

4.2.4 Policy Universe

We would like to know how many stationary policies are possible with this model. Recall that the number of policies in the policy universe is:

$$U = \prod_{\sigma} O(\sigma). \tag{4.5}$$

where $O(\sigma)$ is the number of distinguishable discarding options for state σ. The expressions for $O(\sigma)$ were simple for the models in Chapter 3, but for the current model, the dependence

of $O(\sigma)$ on σ is much greater. We can express this dependence as follows. If $n(\sigma)$ is the number of waiting (queued) packets for state σ, $O(\sigma)$ is the number of distinct combinations of $n(\sigma)$ (or B if $B < n(\sigma)$) or fewer packets which can be retained from the $n(\sigma)$ packets represented by σ. For example, the smallest value of $O(\sigma)$ is $n(\sigma)+1$ (or $B+1$ if $B < n(\sigma)$), which is obtained when the packets waiting in the queue (represented by σ) all have the same delivery priority. This is because there is only one distinct combination for each of the $n(\sigma)+1$ possible number of retained packets, including the option of retaining zero packets. We know that for a given queue fill n, there are D^n states having this queue fill. So a *very loose* lower bound on the number of policies in the policy universe is:

$$U \geq \prod_{n=1}^{B-1} (n+1)^{D^n} \cdot \prod_{n=B}^{N-1} (B+1)^{D^n}. \tag{4.6}$$

Even this lower bound is huge compared to the values of U for the models in Chapter 3. For example, for a simple system with $(N,L,D) = (3,2,3)$, this lower bound is 4096 (which is actually achieved in this case), compared with the 256 possible policies for the restricted queue state, 2-dimensional model in Chapter 3. For a system with only one more queue position, that is, $(N,L,D) = (4,2,3)$, this lower bound is larger than 10^{18}! We show in section 4.3.2 that the policy universe can be reduced with little effort by eliminating certain policies which can never be optimal, but it is generally still too large to even consider exhaustive search as a possible method for finding the optimal policy.

4.3 General FS/Q Analysis and Results

Having developed a well-defined model in the previous section, we will now use the model to analyze FS/Q systems with any dimensions (N,L,D) and any model parameters. The next section will prove some more specific results for the case of $B=1$, that is, systems that are allowed to retain only one packet.

We begin this section by setting up the problem in dynamic programming terms. We then show how the policy universe can be reduced by considering only candidate policies. Finally, we derive a computationally efficient method for value determination, which can then be used in the policy iteration algorithm for finding the optimal policy for a given set of model parameters.

4.3.1 Dynamic Programming Problem Formulation

The search for an optimal discarding policy for an FS/Q system is formulated as a dynamic programming problem by describing the present costs, states and transition probabilities, and showing that average cost criterion is proportional to our optimality criterion of expected discarding cost per arrival.

4.3.1.1 Present Discarding Costs

Unlike the A/A model of Chapter 3, we have incorporated the most recently arrived packets into the state description, so the present cost of a discarding decision is deterministic and there is no need to consider an *expected* present cost. The present cost of a discarding decision for this model is clearly the sum of the costs of the discarded packets. We will often find it more convenient to consider *retained* packets, in which case the cost of a discarding decision is the sum of the discarding costs of *all* the packets represented by the state *minus* the sum of the discarding costs of the retained packets. However, except for the special case of retaining a single packet, we will see that it is difficult to write simple mathematical expressions for these present costs for a given *candidate* option.

4.3.1.2 The Average Cost Criterion

Due to the difficulty in expressing the discarding costs mathematically, we will use a statistical argument to show that minimizing the average cost per step J is equivalent to minimizing the expected discarding cost $E[C]$. We note that the average cost per step for this

model is the limit as $K \to \infty$ of the cost per service completion, time-averaged over K service completions. If the policy is time-invariant and the arrival distributions are statistically stationary, then the system is ergodic, so that time averaging and ensemble averaging yield the same values. So the time-averaged cost per service completion is also the statistical expected cost per service completion. Since the expected number of arrivals in a service interval is λ, we then have $J = \lambda \cdot E[C]$, and minimizing J also minimizes $E[C]$.

4.3.1.3 State Transitions

We have already described the states of the model; we now consider transition probabilities. Suppose that a discarding decision for a given state σ is to *retain* b packets, with $0 \le b \le B$. Suppose for now that $b \ge 1$. We may then represent the ordered set of delivery priorities of the retained packets as a vector of length b $(r_b \; r_{b-1} \cdots r_2 \; r_1)$, using our convention that leftmost packets are farthest from the server. Before the next discarding decision instant, there will be a service completion and r_1 will move into the server. Note that the next state will then *always* have $(r_b \cdots r_2)$ as the delivery priorities of the $b-1$ packets closest to the server, regardless of the arrivals in the service interval following the discarding decision. We will refer to this set of possible states as the next state *form* corresponding to this particular discarding decision.

Now represent the l $(0 \le l \le L)$ arrivals (inputs) in the service interval following the discarding decision as the vector of length l of their delivery priorities $(i_l \; i_{l-1} \cdots i_2 \; i_1)$, where i_1 is the delivery priority of the first arrival. The next queue state (call it ω) for this discarding decision and this arrival can be represented as:

$$\omega = (0 \cdots 0 \; i_l \cdots i_1 \; r_b \cdots r_2). \tag{4.7}$$

For systems that discard after the service completion, the total length of this vector will be $N-1$; for systems that discard before the service completion, the length will be N. However,

this is the *only* difference (for $b \geq 1$) between the two types of systems, as noted in the previous section. The probability that this will be the next state is:

$$q_l \cdot (p_{i_l} \cdot p_{i_{l-1}} \cdot \cdots \cdot p_{i_2} \cdot p_{i_1}) = q_l \cdot \prod_{j=1}^{l} p_{i_j} \tag{4.8}$$

where q_l is the probability of l arrivals in a service interval and p_d is the probability that an arriving packet will have discarding priority d. For a given number of arrivals l, difference (for $b \geq 1$) between the two types of systems, there are D^l possible next states, so the total number of possible next states for any particular discarding decision with $b \geq 1$ (that is, the number of states in the next state *form* as defined above) is $\sum_{l=0}^{L} D^l$.

As noted in the previous section, the $b=0$ option (discard all packets) has the same transition vector as the $b=1$ option for systems that discard before the service completion, so we consider $b=0$ only for systems that discard after the service completion. For such systems, the first *arrival* i_1 will enter the server prior to the next discarding decision, so the next state for a given set of l arrivals can be represented as a vector of length $N-1$:

$$\omega = (0 \cdots 0\ i_l \cdots i_2). \tag{4.9}$$

The probability of this particular next state is:

$$q_l \cdot (p_{i_l} \cdot p_{i_{l-1}} \cdot \cdots \cdot p_{i_3} \cdot p_{i_2}) = q_l \cdot \prod_{j=2}^{l} p_{i_j} \tag{4.10}$$

and the number of possible next states for the $b=0$ (retain zero packets) discarding decision (the number of states in the $b=0$ next state form) is $\sum_{l=0}^{L-1} D^l$.

4.3.1.4 Existence of Optimal Stationary Policy

We would like for all stationary policies to be irreducible to ensure the existence of a stationary policy which is optimal. This is clearly not the case, since a possible stationary

policy is to discard all queued packets (retain zero packets) regardless of state. For such a policy, it is clear from the previous discussion of transition probabilities that it is impossible to transition to any state with queue fill n larger than $L-1$. So we will again (as in Chapter 3) restrict our search to policies which are guaranteed to be irreducible. This is illustrated in Figure 4-2, where we now focus on FS/Q policies in the lower right quadrant of Figure 2-5.

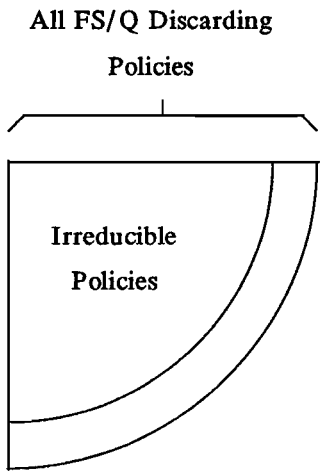


Figure 4-2. Irreducible FS/Q Policies

The following two conditions, taken together, are sufficient to guarantee irreducibility of the transition matrix for an FS/Q system.

1. All arrival probabilities are strictly positive. That is, $q_l > 0$ for all l and $p_d > 0$ for all d.
2. For all b in the range from 1 to B and for all combinations of $b-1$ delivery priorities, there exists in the set of states with queue fill $n \leq L+b-2$ at least one state such that the control decision for that state is to retain b packets with the leftmost $b-1$ retained delivery priorities forming the given combination of $b-1$ delivery priorities.

We show irreducibility under these conditions by showing that we can transition from any arbitrary initial state to any arbitrary target state in a finite number of steps, each with non-

zero transition probability. Condition 1 implies that $q_0 > 0$, which means that we can transition from any initial state to the all zero state (empty queue) with non-zero probability in a finite number of steps regardless of the policy. Condition 1 then implies that we may transition from the empty queue state to any state with queue fill $n \leq L-1$. Then condition 2 with $b=1$ (along with condition 1) implies that we may transition to any state with queue fill $n=L$. Condition 2 with $b=2$ (along with condition 1) then implies that we may transition to any state with queue fill $n=L+1$, and we may continue in this way, finally reaching any state with the maximum queue fill $B+L-1$ * by invoking condition 2 with $b=B$.

4.3.2 Reduction of Policy Universe

We show here how the policy universe can be reduced by considering only candidate policies.

4.3.2.1 Candidate Policies

Consider any state $\sigma = (0 \cdots 0\ d_n \cdots d_1)$ and recall that we are calling the set of all possible next states for a given discarding decision the next state *form*. Now consider any set of discarding options for which the next state form is the same. Equivalently, this is a set with member options that all have the same *state transition vector*. For example, in a system with $(N,L,D) = (4,2,2)$ let $\sigma = (1\ 1\ 2)$. Figure 4-3 illustrates that there are three discarding options that have the next state form characterized by a 1 in the rightmost place.

The first option discards the leftmost priority 1 packet, so that the priority 2 packet moves into the server before the next discarding instant and the middle priority 1 packet is first in line at the next discarding instant. The second option illustrated discards the middle 1 packet,

* For systems that discard after the service completion, we have already noted that $B=N-L$, so the maximum queue fill *at the discarding instant* is $N-1$. For systems that discard before the service completion, the maximum queue fill is N.

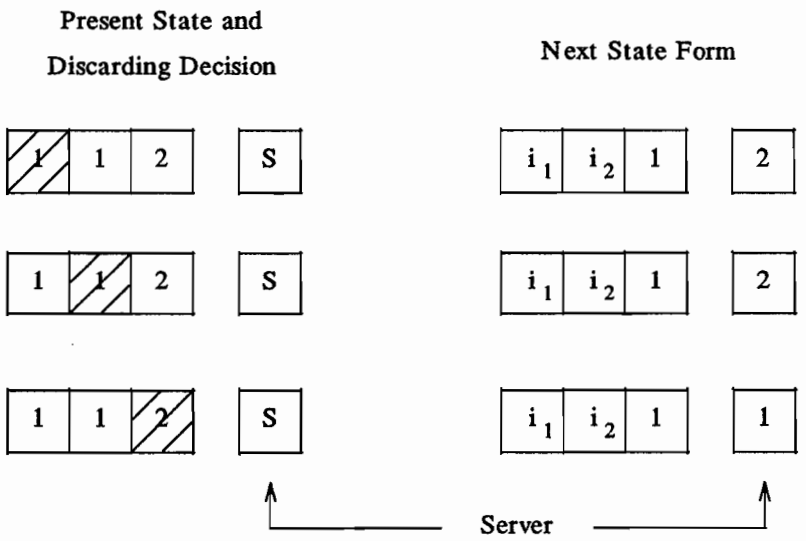


Figure 4-3. Example of Discarding Options with Same Next State Form

allowing the 2 packet to move into the server and leaving the leftmost 1 packet first in line. The third decision discards the 2 packet, allowing the middle 1 packet to move into the server and leaving the leftmost 1 packet first in line.

In such a case where the state transition vectors are identical, we saw in Chapter 2 that at every dynamic programming step, the expected future costs of all the options in the set are identical. So the best decision *in this set* is the one that minimizes the present discarding cost, and all other options are *guaranteed* to be sub-optimal. If two or more options minimize the present discarding cost, they are *indistinguishable* with respect to costs, so they may be treated as a single option. According to our previous definition, options that are not guaranteed to be sub-optimal are *candidate options*. So in our example, the first two options are indistinguishable and constitute a candidate option and the third option is guaranteed to always be sub-optimal. We may draw the following conclusion.

> *The number of distinguishable candidate discarding options for a given state is equal to the number of distinct next state forms for that state, that is, the number of distinct state transition vectors.*

Candidate policies are composed of candidate options, and we represent the candidate policies in Figure 4-4 as a subset of all the irreducible policies in our optimal search region.

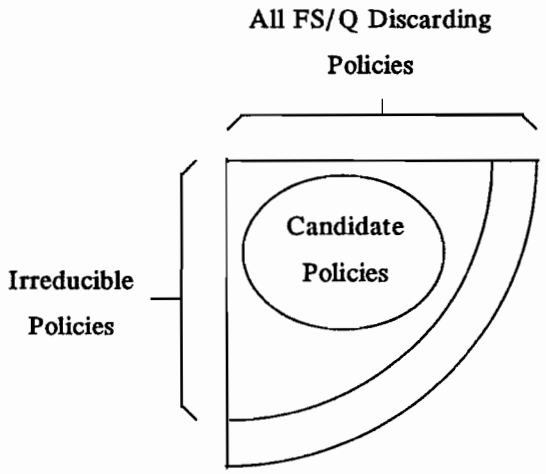


Figure 4-4. Candidate FS/Q Discarding Policies

4.3.2.2 Present Discarding Cost of Candidate Options

We now consider the general form of the present discarding cost for a candidate option. Fix a state σ and a particular next state form, that is, a particular number of retained packets b and a particular pattern of $b-1$ retained packets which will constitute the rightmost entries of the next state. There may in general be several such combinations of $b-1$ packets in a given state. For example, suppose the state is $\sigma=(d_8\ d_7\ d_6\ d_5\ d_4\ d_3\ d_2\ d_1)$ = (2 4 1 3 2 1 2 1) and we are retaining $b=3$ packets such that the $b-1=2$ leftmost retained packets are (2 1). There are two (not three) such combinations in this state, since the two *rightmost* packets in the state cannot be the two *leftmost* of three retained packets.

Once the next state form is fixed, the present discarding cost is determined by the delivery priority of the final (rightmost) retained packet. To minimize the present discarding cost, we must maximize the delivery priority of the this rightmost retained packet. This is done in general by choosing the *leftmost* combination of $b-1$ packets, and then choosing the single

packet to the right of this combination with the largest delivery priority. In our example, we will choose to retain packets represented by $d_8 = 2$, $d_6 = 1$ and $d_5 = 3$.

We conclude that it would be notationally cumbersome (at best) to write a general mathematical description of the present discarding cost for a candidate discarding option. However, for $b=1$, there is only one possible next state form and we know from previous discussions that the candidate option will retain the highest priority packet in the state. So the present discarding cost is just the sum of the discarding costs of the packets in the state minus the discarding cost of the highest priority packet in the state. We will use this result in the section on optimal $B=1$ policies.

4.3.2.3 Number of Candidate Policies

We would also like to know how many such candidate policies there are. We will consider as a group all states with a given queue fill n, for each value of n. The case of $n=0$ is simple, since there is only one such state and it has only the one control "option" of doing nothing! The case of $n=1$ is not much more difficult, since there are D such states and the only possibilities for b, the number of retained packets, are $b=0$ and $b=1$. For $b=1$, there is still only one next state form since the retained packet will always move into the server, implying that the retained packet should be the one in the queue with the highest priority. Compare this with the number of *possible* $b=1$ options, which is just the number of different delivery priorities in the state.

For any $n \geq 2$, there are D^n states with the given queue fill and we may retain up to n packets if $n < B$ or B packets if $n \geq B$. The cases of $b=0$ and $b=1$ have already been covered. For larger b, the rightmost retained packet will move into the server, leaving a combination of $b-1$ packets to define the next state form. So for a given state and number of retained packets b, the number of *candidate* options is the number of distinct, order-preserving combinations of

$b-1$ packets selected from the $n-1$ leftmost packets in the state. Compare this with the total number of *possible* options for a given n and b, which is the number of distinct, order-preserving combinations of b packets selected from the n packets in the state. Of course, both of these quantities depend strongly on the particular set of packets in the state, so a precise general comparison is not feasible. We can say that the difference is not great *for a given n and b,* but the effect on the number of *policies* can be significant. However, the total number of candidate policies is still generally huge.

For example, we have calculated the two quantities for a system that discards after the service completion with dimensions $(N,L,D) = (4,2,2)$. In calculating the number of *possible* distinguishable policies, we found that there are 2 states with 2 distinguishable options, 4 states with 3 options, 2 states with 4 options, 4 states with 5 options, and 2 states with 6 options. So the total number of policies in the policy universe is $2^2 \cdot 3^4 \cdot 4^2 \cdot 5^4 \cdot 6^2$ which is approximately 117 million. For the number of *candidate* policies, there are 2 states with 2 distinguishable candidate options, 8 states with 3 options, and 4 states with 4 options, yielding approximately 6.7 million candidate policies. The policy universe has been reduced by a ratio of 17-to-1, but the number of candidate policies is still unmanageable.

The case of $B=1$ (retain at most one packet) is again a special case in that it has a simple expression for the number of candidate policies. Any system that discards after the service completion and has $B=1$ (equivalently, $N=L+1$) has only two candidate options for each non-empty queue state: retain zero packets or retain the single highest priority packet. In this case, the number of candidate policies is 2^{S-1} where $S = \sum_{d=0}^{N-1} D^n$. For a $B=1$ system that discards before the service completion, the $b=0$ option is always sub-optimal for any non-empty queue state, so there is only one candidate policy, which is then the optimal policy! We will consider these cases in detail in section 4.4.

We have reduced the policy universe by considering only candidate policies, but we are still generally left with an unmanageable number of policies to consider. Our effort has not been in vain, though, since the concept of next state form will prove useful again in the next section.

4.3.3 Computationally Efficient Value Determination

We saw in Chapter 2 that we need to be able to determine the relative values and average cost of a policy, either to check the optimality of a single given policy or as the value determination step in the policy iteration algorithm for finding the optimal policy for a given set of parameters. In general, this involves solving a linear system of S equations in S unknowns. For the current model, we have $S = \sum_{n=0}^{B+L-1} D^n$, which is quite large even for small values of B, L, and D. However, for our FS/Q discarding model, not all of the S equations are independent, and we show in this section how the value determination can be accomplished by solving a set of simultaneous linear equations of size $\sum_{b=0}^{B-1} D^b$, which is generally much smaller than S. We begin by specifying all of the possible distinct transition vectors for any policy. We then use these transition vectors to define a smaller set of auxiliary variables which we will then use to solve for the relative values and average cost.

4.3.3.1 Distinct Transition Vectors

In the last section on candidate policies, we focused on the number of discarding options *for a given state* and found that this number is the same as the number of next state forms or, equivalently, the number of distinct transition vectors possible for that state. Here we wish to consider two related questions. How many distinct transition vectors are possible within any *policy*, and what form does each transition vector take?

We now consider the set consisting of every possible discarding option for every possible state. Let b again denote a specific *number* of packets retained by some discarding decision.

In the set of all possible discarding options, there are only $B+1$ possible values for b: the integers from 0 to B. For $b=0$, we have already seen that there is only one possible next state form. The same is true of $b=1$ since the retained packet will move into the server and be part of the next state. For a given value of $b > 1$, the $b-1$ leftmost retained packets determine the next state form, so there are D^{b-1} possible transition vectors for each value of b. Letting T be the number of possible distinct transition vectors in the set of all possible discarding options, we have:

$$T = 2 + \sum_{b=2}^{B} D^{b-1} = 1 + \sum_{b=0}^{B-1} D^{b}. \tag{4.11}$$

Re-examining the two sufficient conditions for irreducibility given in section 4.3.1.4, we see that the second condition (on the policy) implies that *every one* of these transition vectors is included in any policy satisfying the condition.

We now wish to specify the form of each possible transition vector. We begin with an example, and generalize from there. Consider a system that discards after the service interval with dimensions $(N,L,D) = (4,2,2)$. The maximum number of retained packets is $B = N-L = 2$ and there are $T = 1 + \sum_{b=0}^{1} 2^b = 4$ possible transition vectors. We will denote them as $\mathbf{t}_{01}$, $\mathbf{t}_{11}$, $\mathbf{t}_{21}$ and $\mathbf{t}_{22}$, where the first index indicates the number of retained packets b and the second index (for $b \geq 2$) indicates the particular pattern of $b-1$ leftmost retained packets. The second index has been included for $b=0$ and $b=1$ for consistency of notation only.

Table 4-1 lists the 4 transition vectors in terms of the arrival probabilities, where we have arranged the states in increasing numerical order (treating the vector notation for the state (4.1) as a base $D+1$ number). Although the transition vectors are row vectors, we show them as column vectors in Table 4-1 due to their size.

TABLE 4-1. Distinct Transition Vectors for (4,2,2) System

States	Transition Vectors			
	$\mathbf{t}_{01}$	$\mathbf{t}_{11}$	$\mathbf{t}_{21}$	$\mathbf{t}_{22}$
000	q_0+q_1	q_0	0	0
001	q_2p_1	q_1p_1	q_0	0
002	q_2p_2	q_1p_2	0	q_0
011	0	$q_2p_1p_1$	q_1p_1	0
012	0	$q_2p_1p_2$	0	q_1p_1
021	0	$q_2p_2p_1$	q_1p_2	0
022	0	$q_2p_2p_2$	0	q_1p_2
111	0	0	$q_2p_1p_1$	0
112	0	0	0	$q_2p_1p_1$
121	0	0	$q_2p_1p_2$	0
122	0	0	0	$q_2p_1p_2$
211	0	0	$q_2p_2p_1$	0
212	0	0	0	$q_2p_2p_1$
221	0	0	$q_2p_2p_2$	0
222	0	0	0	$q_2p_2p_2$

Vector $\mathbf{t}_{01}$ has non-zero values only for states representing $L-1$ or fewer queued packets, since the first arrival will move into the server before the next discarding instant. The three non-zero entries are easily derived. If we retain zero packets and one or zero packets arrive, the next state will be the all-zero state. If two packets arrive, the next state will be $00d$, where d is the delivery priority of the second arrival. Vector $\mathbf{t}_{11}$ has non-zero values only for states representing L or fewer queued packets, since the retained packet will move into the server before the next discarding instant and there can be no more than L arrivals. Vector $\mathbf{t}_{21}$ has non-zero values only for states with a 1 in the rightmost position and vector $\mathbf{t}_{22}$ has non-zero values only for states with a 2 in the rightmost position, since the leftmost *retained* packet will be first in line (the *rightmost* packet) at the next discarding instant. The values for these vectors are derived in a manner similar to the derivations for $\mathbf{t}_{01}$.

To generalize from this example, we begin by defining $\mathbf{t}_{bj}$ as the transition vector corresponding to the j^{th} pattern of $b-1$ packet delivery priorities resulting from retaining b packets. For $b \geq 2$, we arrange the patterns in increasing order, as we did with the states, and

assign an integer to each pattern, so that j in the pair bj must be an integer between 1 and D^{b-1}. For $b=0$ and $b=1$, we force $j=1$. These definitions are consistent with our example.

To construct the transition vectors, we begin by introducing some odd-looking recursive vector definitions. First, for $0 \le b-1 \le B-1$, define a vector of length D^{b-1} as:

$$\psi^{0(b-1)} := (1\ 0\ \cdots\ 0). \tag{4.12}$$

This starting vector will "spread out" the arrival probabilities appropriately. In our example, we have $\psi^{00} = (1)$ and $\psi^{01} = (1\ 0)$. Now for each value of $b-1$, construct L more vectors recursively as:

$$\psi^{(l+1)(b-1)} = \left[p_1\psi^{l(b-1)}\ p_2\psi^{l(b-1)}\ \cdots\ p_D\psi^{l(b-1)}\right] \tag{4.13}$$

for $0 \le l \le L-1$, where p_d is the probability that a given arrival has delivery priority d. These are vectors describing particular combinations of arrivals. Note that $\psi^{(l+1)(b-1)}$ has $D^{(l+1)+(b-1)} = D^{l+b}$ elements. For $b=0$ in our example, $\psi^{00} = (1)$, $\psi^{10} = (p_1\ p_2)$ and $\psi^{20} = (p_1\psi^{10}\ p_2\psi^{10}) = (p_1p_1\ p_1p_2\ p_2p_1\ p_2p_2)$. For $b=1$, $\psi^{01} = (1\ 0)$, $\psi^{11} = (p_1\ 0\ p_2\ 0)$ and $\psi^{21} = (p_1\psi^{11}\ p_2\psi^{11}) = (p_1p_1\ 0\ p_1p_2\ 0\ p_2p_1\ 0\ p_2p_2\ 0)$.

As placeholders on the left side of some of the transition vectors, define $\mathbf{0}_b$ as the all-zero vector of length $\sum_{i=0}^{b-2} D^i$ for $b \ge 2$, and as no vector for $b=0$ or $b=1$. As placeholders on the right side of some of the transition vectors, define $\mathbf{0}^b$ as the all-zero vector of length $\sum_{i=L+b}^{B+L-1} D^i$ for $b \ge 2$, and as no vector for $b=B$. In our example, $\mathbf{0}_2 = (0)$, $\mathbf{0}^0$ is the all-zero vector of length $2^2 + 2^3 = 12$ and $\mathbf{0}^1$ is the all-zero vector of length 8.

Finally, we can express the transition vectors as:

$$\mathbf{t}_{01} = \left[(q_0+q_1)\psi^{00}\ q_2\psi^{10}\ q_3\psi^{20}\ \cdots\ q_L\psi^{(L-1)0}\ 0\ \cdots\ 0\right] \tag{4.14}$$

where there are $\sum_{i=L}^{B+L-1} D^i$ zeros at the end. For $1 \le b \le B$:

$$\mathbf{t}_{b1} = \left[\mathbf{0}_b \;\; q_0\boldsymbol{\psi}^{0(b-1)} \;\; q_1\boldsymbol{\psi}^{1(b-1)} \;\; \cdots \;\; q_L\boldsymbol{\psi}^{L(b-1)} \;\; \mathbf{0}^b\right] \tag{4.15}$$

and

$$\mathbf{t}_{b(j+1)} = \mathbf{t}_{bj} \textit{ circular shifted right one place} \tag{4.16}$$

for $1 \le j \le D^{b-1}-1$. The transition vector $\mathbf{t}_{b1}$ is the basic transition vector for retaining b packets, and is applied to the all-one combination of $b-1$ retained vectors. The circular shift just applies the same basic vector to a new combination of $b-1$ retained vectors. As a check, each transition vector should have $S = \sum_{n=0}^{B+L-1} D^n$ elements. From our previous definitions, we see that the number of elements in $\mathbf{t}_{b1}$ is $\sum_{i=0}^{b-2} D^i + \sum_{i=b-1}^{b+L-1} D^i + \sum_{i=L+b}^{B+L-1} D^i = \sum_{i=0}^{B+L-1} D^i = S$, and every vector has the same number of elements.

Using the previous results for the $\boldsymbol{\psi}$ vectors as applied to our example, the reader can easily verify from Table 4-1 that equations (4.14) through (4.16) are a valid description of the transition vectors.

4.3.3.2 Simplification Based on Distinct Transition Vectors

Recall that for a given policy, the relative values and average cost are the solutions to the vector equation:

$$J\cdot\mathbf{e} + \mathbf{v} = \boldsymbol{\chi} + \mathbf{P}\cdot\mathbf{v} \tag{4.17}$$

where one element of the v vector may be arbitrarily set to 0 to get S linear equations in S unknowns. But if the transition matrix $\mathbf{P}$ contains only $T < S$ distinct rows (the transition vectors), we would expect that at most T of these equations are independent. We will show here how to find all S unknowns by solving only $T-1$ (independent) simultaneous linear equations.

We begin by defining the T dot products:

$$R_{bj} = \mathbf{t}_{bj} \cdot \mathbf{v} \tag{4.18}$$

where again bj is the pair representing the discarding action: b is the number of packets retained and j is the particular pattern of leftmost $b-1$ retained packets. From the form of (4.17) and our discussion in Chapter 2 on dynamic programming, these scalar values correspond to expected future costs in the basic dynamic programming problem, one for each possible set of retained packets. So for every state σ, there exists a pair $bj(\sigma)$ such that:

$$J + v_\sigma = \chi_\sigma + \mathbf{t}_{bj(\sigma)} \cdot \mathbf{v} = \chi_\sigma + R_{bj(\sigma)}. \tag{4.19}$$

We now let $\sigma=1$ correspond to the empty queue state, so that $\chi_1=0$ and $bj(1)=01$, and exercise our one degree of freedom by setting $v_1=0$. Note that we then have:

$$J = R_{01}. \tag{4.20}$$

Subtracting the first equation from each of the others, we get:

$$v_\sigma = \chi_\sigma + (R_{bj(\sigma)} - R_{01}). \tag{4.21}$$

We now define the difference variables for all pairs bj:

$$R'_{bj} := R_{bj} - R_{01} = (\mathbf{t}_{bj} - \mathbf{t}_{01}) \cdot \mathbf{v} \tag{4.22}$$

so that for all σ:

$$v_\sigma = \chi_\sigma + R'_{bj(\sigma)}. \tag{4.23}$$

We already know that $v_1=0$ and $R_{01}=0$. Then substituting the $S-1$ remaining equations (4.23) into the $T-1$ remaining equations (4.22) yields a set of $T-1$ simultaneous linear equations in the $T-1$ unknowns R'_{bj}. By solving this set of equations, we can substitute the R'_{bj} in the $S-1$ equations (5.23), yielding the relative values v_σ. Having all of these values, we can find the average cost per step J from any one of the equations (4.19). We have thus

solved for all of the relative values and the average cost (S values in all) by solving only $T-1$ simultaneous linear equations and substituting the results into S simple (non-simultaneous) equations.

Since the computational cost of solving a general set of M simultaneous equations grows approximately as M^3 [Pres86] the computational savings can be substantial if $T-1$ is much less than S. We already know that $S = \sum_{n=0}^{B+L-1} D^n$ and $T-1 = \sum_{b=0}^{B-1} D^b$, so $T-1$ is just S with the *largest* L powers of D omitted from the sum. So $T-1$ is generally *much* less than S, and the computational savings in the value determination process are generally substantial.

We now illustrate the entire solution process for our example (4,2,2) system. For this example, we have $S=15$ but $T-1$ is only 3, allowing us to get a closed form solution for the relative values and average cost for a given policy in terms of a general set of model parameters. We must first specify a discarding policy, which we do in Table 4-2 by showing for each state the packets that are *retained*, indicating a discarded packet by an X. We also show the equation corresponding to (4.23) for each relative value.

TABLE 4-2. Example Policy for (4,2,2) System

State Index σ	State	Packets Retained	Relative Value Equation
1	000	None	$v_1 = 0$
2	001	1	$v_2 = R'_{11}$
3	002	2	$v_3 = R'_{11}$
4	011	11	$v_4 = R'_{21}$
5	012	12	$v_5 = R'_{21}$
6	021	21	$v_6 = R'_{22}$
7	022	22	$v_7 = R'_{22}$
8	111	X11	$v_8 = c_1 + R'_{21}$
9	112	X12	$v_9 = c_1 + R'_{21}$
10	121	12X	$v_{10} = c_1 + R'_{21}$
11	122	X22	$v_{11} = c_1 + R'_{22}$
12	211	2X1	$v_{12} = c_1 + R'_{22}$
13	212	2X2	$v_{13} = c_1 + R'_{22}$
14	221	22X	$v_{14} = c_1 + R'_{22}$
15	222	X22	$v_{15} = c_2 + R'_{22}$

For any indistinguishable options for a given state, we have arbitrarily shown the leftmost (most recently arrived) packet being discarded. For example, in state 211, we could equivalently discard either of the two priority 1 packets.

This policy is an example of what we will call a *maximum retention policy*, which we define as a policy which retains the maximum number of packets for each state *and* which chooses the highest priority packets in the queue to retain. For this particular system, there is only one other maximum retention policy, which differs from the one shown only by discarding the leftmost 1 packet in state 121, rather than the rightmost packet. Note that this alternative decision would have the $\mathbf{t}_{22}$ transition vector (so that R_{22} would appear in the relative value equations) instead of $\mathbf{t}_{21}$ (R_{21} in the equation).

We can now use Tables 4-1 and 4-2 (with some intermediate algebraic steps which have been omitted) to obtain the following $T-1=3$ simultaneous linear equations for the R' variables.

$$R'_{11} = (\mathbf{t}_{11}-\mathbf{t}_{10})\cdot\mathbf{v} \tag{4.24}$$
$$= (q_1-q_2)R'_{11} + q_2p_1R'_{21} + q_2p_2R'_{22}$$

$$R'_{21} = (\mathbf{t}_{21}-\mathbf{t}_{01})\cdot\mathbf{v} \tag{4.25}$$
$$= (q_0-q_2)R'_{11} + (q_1+q_2)p_1R'_{21} + (q_1+q_2)p_2R'_{22} + q_2c_1$$

$$R'_{22} = (\mathbf{t}_{22}-\mathbf{t}_{01})\cdot\mathbf{v} \tag{4.26}$$
$$= (q_0-q_2)R'_{11} + (q_1 + q_2p_1)p_1R'_{21} +$$
$$(q_1p_2 + q_2p_1p_2 + q_2p_2p_1 + q_2p_2p_2)R'_{22} +$$
$$q_2(p_1p_1c_1 + p_1p_2c_1 + p_2p_1c_1 + p_2p_2c_2)$$

We notice that a simpler expression for the solution can be obtained if we define some new variables.

$$R^*_{11} = R_{11} - R_{01} = R'_{11} \tag{4.27}$$

$$R^*_{21} = R_{21} - R_{11} = R'_{21} - R'_{11}$$

$$R^*_{22} = R_{22} - R_{21} = R'_{22} - R'_{21}$$

Note that we can express the R' variables in terms of the R^* variables as follows:

$$R'_{11} = R^*_{11} \tag{4.28}$$

$$R'_{21} = R^*_{21} + R^*_{11}$$

$$R'_{22} = R^*_{22} + R^*_{21} + R^*_{11}.$$

Note that (4.24) is also an expression for R^*_{11}. Also, we can get an expression for R^*_{21} by subtracting (4.24) from (4.25). Similarly, we can get an expression for R^*_{22} by subtracting (4.25) from (4.26). Again omitting the intermediate algebraic steps, we can substitute relations (4.27) into these expressions for the R^* variables to get the following matrix equation:

$$\begin{bmatrix} R^*_{11} \\ R^*_{21} \\ R^*_{22} \end{bmatrix} = \begin{bmatrix} q_1 & q_2 & q_2p_2 \\ q_0 & q_1 & q_1p_2 \\ 0 & 0 & q_2p_1p_2 \end{bmatrix} \cdot \begin{bmatrix} R^*_{11} \\ R^*_{21} \\ R^*_{22} \end{bmatrix} + \begin{bmatrix} 0 \\ q_2c_1 \\ q_2p_2p_2(c_2-c_1) \end{bmatrix}. \tag{4.29}$$

Solving this set of equations, we get:

$$R^*_{10} = \frac{q_2^2p_2^3(c_2-c_1) + q_2^2(1-q_2p_1p_2)c_1}{R} \tag{4.30}$$

$$R^*_{21} = \frac{[(1-q_1)q_1p_2 + q_0q_2p_2]\cdot q_2p_2p_2(c_2-c_1) + q_2c_1(1-q_1)\cdot(1-q_2p_1p_2)}{R} \tag{4.31}$$

and

$$R^*_{22} = \frac{q_2p_2p_2(c_2-c_1)\cdot(q_2^2 + q_0^2 + q_2q_0)}{R} \tag{4.32}$$

where

$$R = (q_2^2 + q_0^2 + q_0q_2)\cdot(1-q_2p_1p_2). \tag{4.33}$$

Having these R^* values, we can substitute them into (4.28) to get the R' values, and then substitute those into the equations in Table 4-2 to get the v_σ values. The only remaining quantity is J, which is found easily from:

$$\begin{aligned} J + v_1 &= \chi_1 + \mathbf{t}_{01}\cdot\mathbf{v} \\ &= (q_0+q_1)\cdot(0) + q_2p_1R'_{11} + q_2p_2R'_{11} \\ &= q_2R'_{11}. \end{aligned} \tag{4.34}$$

Recalling that the value determination step is a major part of the policy improvement algorithm, we conclude that using this computationally efficient method of value determination will speed up the policy improvement algorithm significantly.

We note, however, that the policy improvement algorithm is useful for finding the optimal discarding policy for a *given* set of model parameters, that is, input statistics and discarding costs. Ideally, we would like to be able to identify a *family of optimal policies* which is complete in the sense that, for *any* set of parameters, one of the policies in the family is optimal. A further goal would be to find an expression for each such policy describing conditions (in terms of model parameters) for which it is optimal. We accomplish these goals in the section 4.4 for systems that may retain only $B=1$ packet.

4.3.4 Performance Comparisons for Optimal FS/Q Policies

4.3.4.1 Comparison to Default and Optimal A/A Policies

We are now in a position to compare the performance of optimal FS/Q policies with the other policies we have analyzed, namely the default policy and optimal queue fill A/A policies from Chapter 3. We consider systems with $(N,L,D) = (5,2,2)$ and take an FS/Q system that discards after the service completion. Though we can use any distribution for the number of arrivals in a service interval for the FS/Q analysis, we will choose a binomial distribution since the analysis model of Chapter 3 was restricted to that distribution. We consider a balanced

system with $(p_1, p_2) = (0.5, 0.5)$ and $(c_1, c_2) = (1, 2)$. In Figure 4-5 we plot the normalized expected discarding cost per arrival versus load λ for the default policy, the optimal queue fill A/A policies, and the optimal FS/Q policies, where we have optimized the policies for each value of load.

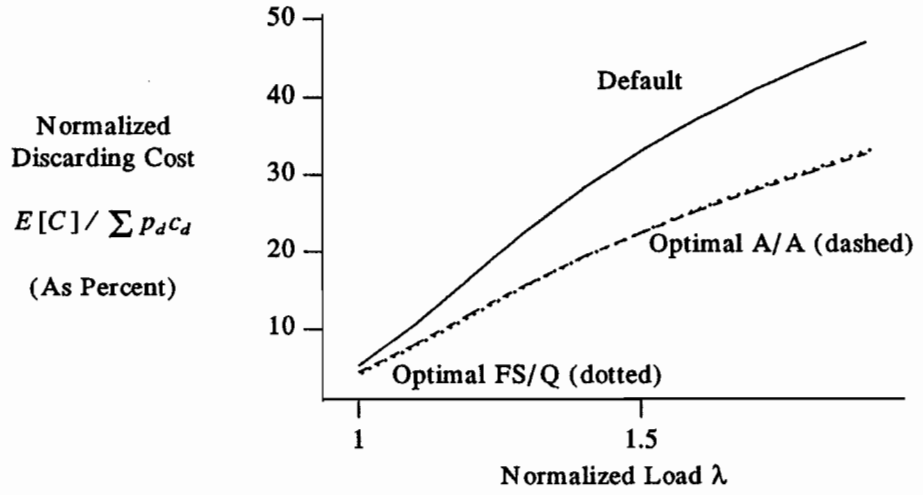


Figure 4-5. Performance Comparison of Discarding Policies for $(N,L,D) = (4,2,2)$ System

The somewhat surprising result is that both sets of optimal discarding policies, queue fill A/A and FS/Q, have almost identical performance. Note also that both sets of optimal policies perform significantly better than the default policy. We might hypothesize that the structure of the discarding policy is not important, as long as the policy is optimized. However, we still strongly believe that systems allowed to discard any queued packets should outperform systems allowed to discard only arriving packets, all else being equal. A more reasonable hypothesis would then be that, for the same size dimensions and parameter sets, A/Q systems should perform better than A/A systems, which perhaps perform about as well as FS/Q systems, which perform better than FS/A systems. Verification of this hypothesis is left for future research.

4.3.4.2 Effect of Queue Capacity N

As we did with the A/A systems of Chapter 3, we also want to discover the effect of queue capacity N on system performance for optimal FS/Q policies. We consider two FS/Q systems that discard after the service completion, each with $L=3$ and $D=2$. The distribution of number of arrivals per service interval is again taken to be binomial, and the delivery priority parameters are again balanced: $(p_1, p_2) = (0.5, 0.5)$ and $(c_1, c_2) = (1, 2)$. We plot in Figure 4-6 the normalized discarding cost versus load for two pointwise optimized systems, one with $N=4$ $(B=1)$ and the other with $N=7$ $(B=4)$.

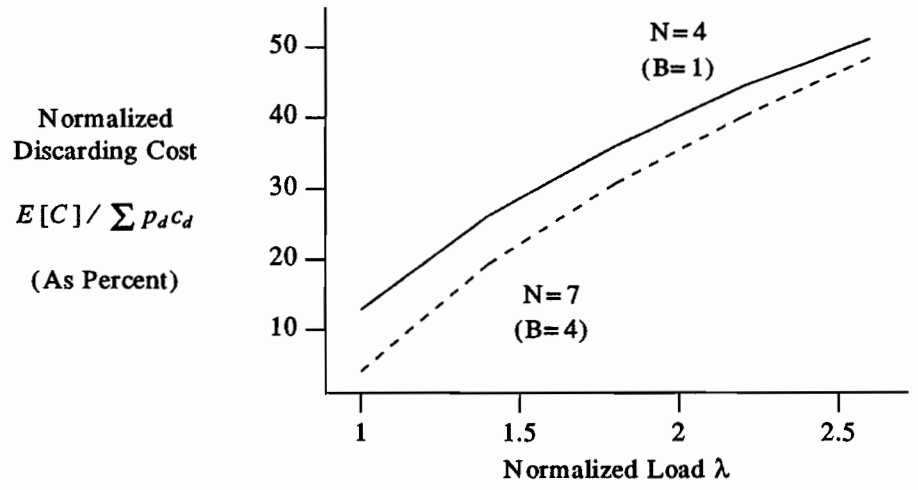


Figure 4-6. Effect of Queue Capacity N For Optimal FS/Q Systems

A larger queue does improve performance, as expected, but the improvement diminishes quickly with increasing load, as it did with the A/A systems. So again, increasing queue capacity does not significantly improve performance for heavily overloaded systems. This result, along with previously noted simplifications for systems allowed to retain only $B=1$ packet, encourages us to investigate these small queue capacity systems in more detail.

4.4 Optimal FS/Q Policies For $B=1$

In this section we will prove some results about the optimal policy for FS/Q systems allowed to retain at most $B=1$ packet, so that queue capacity N is restricted to $N=L$ or $N=L+1$, depending on the discarding instant. Under this restriction, we will see that the policies may also be viewed as a form of FS/A system (discarding only arriving packets), but we will use analysis techniques applicable to any FS/Q system, and thus gain valuable experience and insight concerning these analysis techniques.

We deal first with $B=1$ policies that discard after the service completion. Having analyzed these systems, the analysis for systems that discard before the service completion will involve minor modifications and further simplifications.

The restriction to $B=1$, effectively limiting queue capacity N, may seem quite limiting, but we offer several justifications for considering these systems. First, we have already seen that the advantage of larger queue capacities is marginal for heavily loaded systems, so the extra complexity may not be justified. Secondly, we have also seen that the restriction to $B=1$ allows for several simplifications of the general problem, such as restricting the number of discarding options and allowing for simple expressions for the present discarding cost. Third, analysis of these systems can be a stepping stone of experience and insight for the analysis of larger systems. Fourth, there may be practical applications involving very small systems, such as small internal switching elements interconnected to form a larger switching fabric. Finally, we will see that the analytical solution is elegant and possesses some interesting properties.

4.4.1 Systems Discarding After Service Completion

In keeping with our conclusions about the relationship of the discarding instants (before or after service completion), we will first analyze a system discarding after the service interval, and then modify the analysis slightly to obtain results for a system discarding before the service

interval.

In this section (discard after service completion) we will identify a very small family of optimal policies. For each policy in the family, we provide closed-form sufficient conditions for its optimality in terms of parameter values, as well as a closed-form expression for its average discarding cost. We further prove that this family is complete in the sense that one of the member policies will always be optimal for any set of parameter values.

With the maximum number of retained packets fixed at $B=1$, a system discarding after the service completion has queue capacity $N=L+1$. Also, there are exactly two *candidate* discarding options for each non-empty queue state: retain zero packets (discard all), or retain the single highest priority packet. As such, there are 2^{S-1} candidate policies, where S is the number of states and is given by $S = \sum_{n=0}^{N-1} D^n$. As noted previously, this is still a very large set of policies from which to choose. We will show, however, that for *any* set of model parameters (arrival statistics and discarding costs), one of only D of these policies will always be optimal.

We can retain at most one packet, and if we do, that packet will move into the server before the next discarding instant, so every state is always determined *entirely* by the most recent set of L or fewer arrivals. So for this restriction of $B=1$, an FS/Q system reduces to a particular type of FS/A system (only Arriving packets may be discarded) that is restricted to retain at most one packet ($B=1$) and for which the first arrival may move into the server before the discard decision, and thus not be eligible for discarding. However, we will use analysis techniques that are generally applicable to any FS/Q system.

4.4.1.1 Optimal Policy Family and Sufficient Optimality Conditions

We define an *optimal policy family* as a set of discarding policies that can be optimal under some set of model parameters. In this section, we describe the what we claim is a *complete* optimal policy family for this problem in the sense that one of the member policies is

guaranteed to be optimal for any set of parameter values. We also state sufficient optimality conditions for each member of the family. The proof follows in the next section.

We denote the optimal policy family as the set $\{G_d : 1 \le d \le D\}$, where G_d is defined as follows. We will use our previous representation for each state σ, repeated here for convenience:

$$\sigma = (0 \cdots 0\ d_{n(\sigma)}\ \ d_{n(\sigma)-1} \cdots d_2\ \ d_1) \tag{4.35}$$

The policy G_d is then defined for each state σ as:

If $d_j < d$ for *all* j *such that* $1 \le j \le n(\sigma)$, (4.36)

Then Retain Zero Packets (Discard All).

If there exists a j *such that* $d_j \ge d$,

Then Retain the Highest Priority Packet.

Then policy G_{d^*} is optimal if d^* satisfies:

$$c_{d^*-1} \le \frac{b_{d^*}}{1-a_{d^*}} \le c_{d^*} \tag{4.37}$$

where by definition c_0:=0. The values a_d and b_d are given by:

$$a_d = \left[1 - P^{d-1}\right] \cdot \left[\sum_{l=1}^{L} q_l \cdot \left[P^{d-1}\right]^{l-1}\right] \tag{4.38}$$

$$= \sum_{l=1}^{L} q_l \cdot \left[\left[P^{d-1}\right]^{l-1} - \left[P^{d-1}\right]^{l}\right]$$

where we define 0^0:= 1 so that $a_1 = q_1$, and

$$b_d = \sum_{l=1}^{L} q_l \cdot \left[C^{D} - \sum_{m=d}^{D} c_m \cdot \left\{\left[\left[P^{m}\right]^{l} - \left[P^{m-1}\right]^{l}\right] - \left[\left[P^{m}\right]^{l-1} - \left[P^{m-1}\right]^{l-1}\right]\right\}\right] \tag{4.39}$$

where

$$\overset{x}{P} := \sum_{m=1}^{x} p_m \tag{4.40}$$

and

$$\overset{x}{C} := \sum_{m=1}^{x} p_m \cdot c_m. \tag{4.41}$$

We also claim that:

For any $\{q_l\}$, $\{p_d\}$, and $\{c_d\}$, *there exists a* d^* *such that* G_{d^*} *is optimal.* (4.42)

If these claims are true, then we know that for any set of model parameters, the optimal policy must be one of only D policies, the members of the optimal policy family. This is a small set indeed, and we represent it accordingly in Figure 4-7.

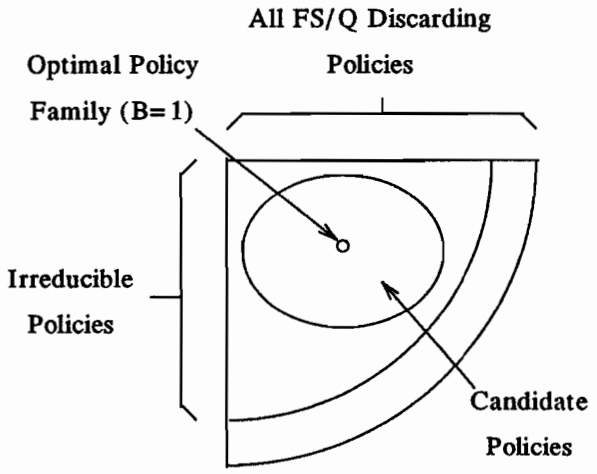


Figure 4-7. The Optimal Policy Family As a Small Subset of Candidate Policies

4.4.1.2 Example and Discussion

Before proceeding to the proof of these assertions, we will provide an example to illustrate the family of policies and some of their properties. We consider a system with $(N,L,D)=(3,2,3)$, so that the number of states is $S = 3^0+3^1+3^2 = 13$. Table 4-3 specifies the

$D=3$ policies in the optimal policy family. Each policy is specified in two ways: first by showing the "state" of the system immediately after discarding (note that this is not the same as the "next state"), and then by showing which packets are discarded.

TABLE 4-3. Optimal Policy Family for $(N,L,D)=(3,2,3)$

Pre-Discard State	Post-Discard State			Discarded Packets		
	G_1	G_2	G_3	G_1	G_2	G_3
0 0	0 0	0 0	0 0	0	0	0
0 1	0 1	0 0	0 0	0	1	1
0 2	0 2	0 2	0 0	0	0	2
0 3	0 3	0 3	0 3	0	0	0
1 1	0 1	0 0	0 0	1	1 1	1 1
1 2	0 2	0 2	0 0	1	1	1 2
1 3	0 3	0 3	0 3	1	1	1
2 1	0 2	0 2	0 0	1	1	2 1
2 2	0 2	0 2	0 0	2	2	2 2
2 3	0 3	0 3	0 3	2	2	2
3 1	0 3	0 3	0 3	1	1	1
3 2	0 3	0 3	0 3	2	2	2
3 3	0 3	0 3	0 3	3	3	3

We will use this table to illustrate several properties of the optimal policy family. As a check, we note first that, as required by $B=1$, there is at most one packet retained by each discarding decision. Note further that the discarding decision is the same for all three policies for any states with a priority D packet. Referring back to the general definition of the family, this property can be generalized as follows.

> *Every optimal policy discards priority D packets only if absolutely required to, that is, if the number of priority D packets queued is larger than* $B=1$.

We believe that this is a general property of any optimal policy for any B, but this remains to be proven. This property is highly intuitive, however, since it seems impossible to ever obtain some future advantage from currently discarding a packet with the largest possible discarding cost.

We now consider the two extreme policies. Policy G_1 is what we have defined as a *maximum retention* policy, in that it always retains the maximum number of packets, and the packets retained have the maximum possible delivery priorities. This is an intuitively appealing policy and we will see that, for $D=2$ and a binomial arrival distribution, it is always either optimal or its performance is very close to the performance of the optimal policy. Note for now that maximum retention policies optimize present discarding costs, but they entirely *ignore* the future costs of the discarding decisions. Seen in this light, we would expect G_1 to be optimal in general when there is a low probability of an arrival in the next interval (low load) or if there is relatively little difference in the discarding costs. We will formalize this observation for the case of $D=2$ shortly.

Policy G_D (G_3 in our example), is at the other extreme. It discards priority D packets only when necessary, but discards *every* queued packet with priority less than D. Does this mean that no packets with priority less than D are ever served? The answer is no, for whenever all packets are discarded, as they frequently are in policy G_D, the *first* arrival in the next service interval moves into the server *before* the next discarding instant and is thus protected from discarding. This policy can be viewed as attempting to minimize the future costs of discarding priority D packets by clearing out as much space for them as possible. Viewed in this way, policy G_D seems appropriate for situations in which the cost of discarding a priority D packet is large compared to the other discarding costs, or the probability of a priority D arrival is high. We formalize this below for the case of two delivery priorities.

4.4.1.3 $D=2$ Case: Two Delivery Priorities

Assume for now that the claims of (4.36) through (4.42) are true. Some of the possible applications for priority packet discarding require only two delivery priorities (for example, the bandwidth management scheme in [Luan88]). Notice that for $D=2$, there are only two policies in the optimal policy family, so either G_1 or G_2 must be optimal. It would be

convenient if we could show that one of these two policies is always optimal or very nearly optimal.

Note that the optimality condition (4.37) for policy G_1 involves the variables a_1 and b_1. In all cases, $a_1 = q_1$. For the case of $D=2$, we recognize that $p_2 = 1-p_1$, so we have:

$$b_1 = q_1 \cdot \left[p_1 c_1 + p_2 c_2 - c_1 \cdot (p_1 - 1 - 0 + 1) - c_2 \cdot (1 - p_1 - 1 + 1) \right] + \sum_{l=2}^{L} q_l \cdot \left[p_1 c_1 + p_2 c_2 - c_1 \cdot (p_1^l - p_1^{l-1}) - c_2 \cdot (1 - p_1^l - 1 + p_1^{l-1}) \right] \tag{4.43}$$

$$= \sum_{l=2}^{L} q_l \cdot \left[(1 - p_{2)c_1} + p_2 c_2 - p_2 p_1^{l-1} (c_2 - c_1) \right]$$

$$= (1 - q_0 - q_1) c_1 + \sum_{l=2}^{L} q_l p_2 (c_2 - c_1) \cdot (1 - p_1^{l-1})$$

So the condition for optimality for policy G_1, $\frac{b_1}{1-a_1} \leq c_1$, reduces to:

$$p_2 \cdot (c_2 - c_1) \cdot \sum_{l=2}^{L} q_l \cdot (1 - p_1^{l-1}) - q_0 \cdot c_1 \leq 0. \tag{4.44}$$

From this expression, we see that factors tending to make G_1 optimal are a large value for q_0 (the probability of no arrivals in a service interval), that is, low load, and small values for $(c_2 - c_1)$ and p_2. By (4.42), if G_1 is not optimal, then G_2 is, so the opposite factors tend to make G_2 optimal. These results are in agreement with our previous general speculations about the optimality of the extreme policies G_1 and G_D.

This result is illustrated in Figures 4-8 through 4-10. These figures plot normalized expected discarding cost versus load for systems with $(N,L,D)=(3,2,2)$ for cost functions of $(c_1,c_2)=(1, 10)$, $(c_1,c_2)=(5, 10)$ and $(c_1,c_2)=(9, 10)$, respectively; that is, for relatively large, medium, and small cost differences. On each plot, we show two priority arrival distributions: $p_2 = 0.9$ (so $p_1 = 0.1$) and $p_2 = 0.1$ (so $p_1 = 0.9$). The distribution of number of arrivals in a service interval was taken to be binomial.

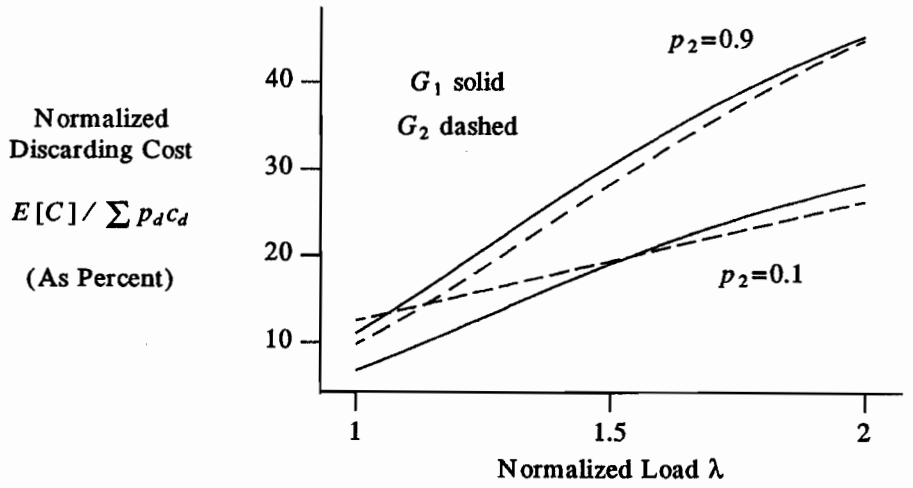


Figure 4-8. Performance Comparison of G_1 and G_2 for $(c_1,c_2) = (1, 10)$

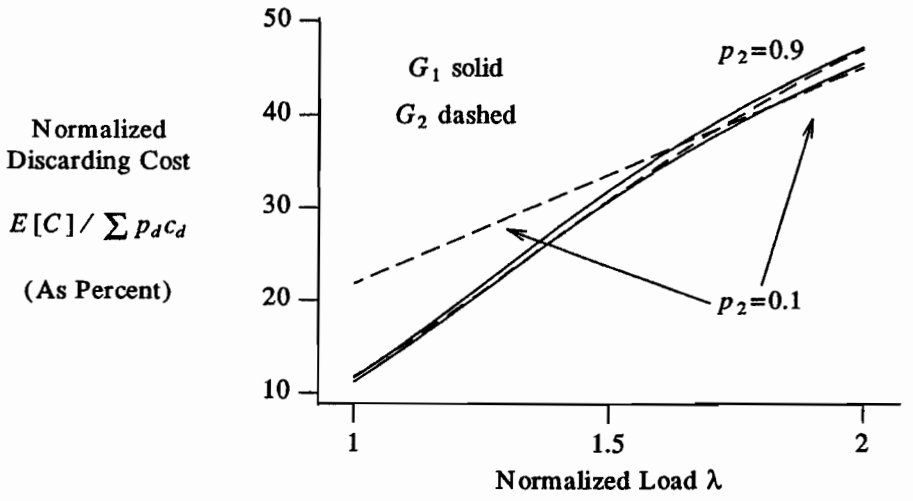


Figure 4-9. Performance Comparison of G_1 and G_2 for $(c_1,c_2) = (5, 10)$

These plots show that, for this binomial arrival distribution, G_1 can be significantly better than G_2, particularly for low overloads (*e.g.*, 27% *vs.* 19% discarding cost in Figure 4-9 for $\lambda = 1.2$ and $p_2 = 0.1$). Furthermore, when G_2 is better, the performance difference is small (*e.g.*, Figure 4-8 for p_2=0.9). So policy G_1 is *always* either optimal or very nearly optimal. This implies that we could achieve a nearly optimal level of performance for *any* parameters by using policy G_1 at all times.

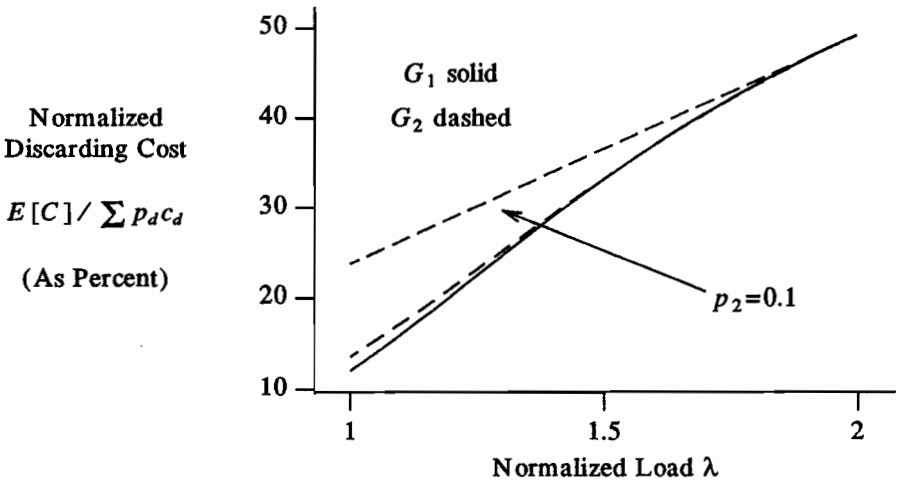


Figure 4-10. Performance Comparison of G_1 and G_2 for $(c_1,c_2) = (5, 10)$

On the other hand, inspection of the inequality (4.44) shows that if q_0 is zero, policy G_1 can never be optimal. So, if for a given system we have some reason to believe that there will always (or nearly always) be an arrival in a service interval (q_0 nearly 0), then G_2 will always (or nearly always) be optimal.

4.4.1.4 Proof of Optimality

To accomplish the proof of claims (4.36) through (4.42), we first prove that the relative values and average cost of each policy G_d in the policy family can be expressed in a particular closed form, then we derive sufficient conditions for the optimality of each, and finally we show that one of these policies must always be optimal.

4.4.1.4.1 Relative Values and Average Cost of Policy G_d

Since we are dealing with $B = 1$, we could theoretically follow the procedure in section 4.3.3 to find the relative values and average cost of any policy. We note that this would require solving only $T - 1 = \sum_{b=0}^{B-1=0} D^b = 1$ equation, but what an equation it would be without specifying numerical values for the parameters! We will choose instead to simply state the form of the

relative values and average cost and then show that this form is valid.

We must begin with some notation. Let G_d represent an arbitrary policy in the policy family defined by (4.36). Then define Σ_0 as the set of all states $\sigma = (0 \cdots 0\, d_{n(\sigma)} \cdots d_1)$ for which $d_j < d$ for all $1 \le j \le n(\sigma)$, that is, the states for which the policy retains 0 packets. Similarly, define Σ_1 as the set of all states σ for which there exists a j such that $d_j \ge d$, that is, the states for which the policy retains 1 packet. Then define:

$$\mathcal{C}_\sigma := \sum_{j=1}^{n(\sigma)} c_{d_j} \tag{4.45}$$

and

$$c_\sigma{}^+ := \max_{1 \le j \le n(\sigma)} c_{d_j} \tag{4.46}$$

so that the present discarding costs are:

$$\chi_\sigma = \begin{cases} \mathcal{C}_\sigma & \text{for } \sigma \in \Sigma_0 \\ \mathcal{C}_\sigma - c_\sigma{}^+ & \text{for } \sigma \in \Sigma_1 \end{cases} \tag{4.47}$$

We now claim that the relative values v_σ are given by:

$$v_\sigma = \begin{cases} \mathcal{C}_\sigma & \text{for } \sigma \in \Sigma_0 \\ \mathcal{C}_\sigma - c_\sigma{}^+ + \dfrac{b_d}{1-a_d} & \text{for } \sigma \in \Sigma_1 \end{cases} \tag{4.48}$$

where a_d and b_d are given in (4.38) and (4.39). Also, the average cost per step is:

$$J^{G_d} = y_d \cdot \left[\frac{b_d}{1-a_d} \right] + z_d \tag{4.49}$$

where y_d and z_d are defined as:

$$y_d := \sum_{l=2}^{L} q_l \cdot \left[1 - \left[P^{d-1} \right]^{l-1} \right] \tag{4.50}$$

$$z_d := \sum_{l=2}^{L} q_l \cdot \left\{ (l-1)\cdot \overset{D}{C} - \sum_{m=d}^{D} c_m \cdot \left[\left[\begin{matrix} m \\ P \end{matrix} \right]^{l-1} - \left[\begin{matrix} m-1 \\ P \end{matrix} \right]^{l-1} \right] \right\}. \tag{4.51}$$

To prove this claim, we need to show that:

$$J^{G_d} + v_\sigma = \chi_\sigma + \mathbf{t}_\sigma \cdot \mathbf{v} \tag{4.52}$$

for all σ, where $\mathbf{t}_\sigma$ is the transition vector for current state σ. We will find it more convenient here to deal with individual transition probabilities, so we represent the next state as ω and let $P_{\sigma,\omega}$ be the transition probability from state σ to state ω. We then must show:

$$J^{G_d} + v_\sigma = \chi_\sigma + \sum_{\omega} P_{\sigma,\omega} \cdot v_\omega \tag{4.53}$$

Suppose first that $\sigma \in \Sigma_0$, so that we retain $b=0$ packets. We begin by recalling the form of the state transition probabilities from section 4.3.1.3, again letting l represent the number of arrivals in a service interval. For any l from 2 to L, the next state will be $\omega = (0 \cdots 0\ i_l\ i_{l-1} \cdots i_2)$ with probability $q_l \cdot (p_{i_l} p_{i_{l-1}} \cdots p_{i_2})$, or in slightly more compact form, $q_l \cdot \prod_{j=2}^{l} p_{i_j}$. For $l \le 1$, we will have $\omega = (0 \cdots 0)$ with probability $q_0 + q_1$, since the first arrival will move into the server before the next discarding instant. To get all of the possible next states, we will need to sum over l, and for each value of l, over all combinations of i_2 through i_l. Since the form of v_ω is different for $\omega \in \Sigma_0$ and $\omega \in \Sigma_1$, we will need to split the summation over all states into these two parts. But by the definition of Σ_0, $\omega \in \Sigma_0$ implies all of the i_2 through i_l are less than d. Similarly, $\omega \in \Sigma_1$ implies that at least one input priority is d or greater. Finally, we will exercise our one degree of freedom in these equations by setting to 0 the relative value for the empty queue state. Substituting all of this into (4.53), we now need to show that:

$$J^{G_d} + \mathcal{C}_\sigma = \mathcal{C}_\sigma + \sum_{l=2}^{L} q_l \cdot \sum_{i_1=1}^{d-1} \cdots \sum_{i_2=1}^{d-1} \prod_{j=2}^{l} p_{i_j} \cdot \mathcal{C}_\omega + \sum_{l=2}^{L} q_l \cdot \sum_{any\ i \geq d} \cdots \sum \prod_{j=2}^{l} p_{i_j} \cdot \left[\mathcal{C}_\omega - c_\omega^+ + \frac{b_d}{1-a_d} \right] \tag{4.54}$$

$$= \mathcal{C}_\sigma + R_{01}$$

where we have used our notation R_{01} from section 4.3.3 for the expected future cost of discarding option $b=0$. Working with R_{01}, we note that the last term of the second partial sum has no dependence on the l or i values, so we work with it first. We find that:

$$\frac{b_d}{1-a_d} \cdot \sum_{l=2}^{L} q_l \cdot \sum_{any\ i \geq d} \cdots \sum \prod_{j=2}^{l} p_{i_j} = \frac{b_d}{1-a_d} \cdot \sum_{l=2}^{L} q_l \cdot \left[1 - \sum_{i_l=1}^{d-1} p_{i_l} \cdots \sum_{i_1=1}^{d-1} p_{i_1} \right] \tag{4.55}$$

$$= \frac{b_d}{1-a_d} \cdot \sum_{l=2}^{L} q_l \cdot \left[1 - \left[P^{d-1} \right]^{l-1} \right]$$

$$= y_d \cdot \frac{b_d}{1-a_d}.$$

We next note that the partial sums of R_{01} have one term in common. Consolidating the two partial sums for this term, we get:

$$\sum_{l=2}^{L} q_l \cdot \sum_{i_1=1}^{D} \cdots \sum_{i_2=1}^{D} \prod_{j=2}^{l} p_{i_j} \cdot \mathcal{C}_\omega = \sum_{l=2}^{L} q_l \cdot (l-1) \cdot \overset{D}{C} \tag{4.56}$$

In this expression, we have taken advantage of the fact that $\mathcal{C}_\omega = \sum_{j=2}^{l} c_{i_j}$, so that for a given j, we may multiply the p_{i_j} with the c_{i_j} and sum over i_j to get $\overset{D}{C}$. The remainder of the p_i probabilities may be individually associated with their respective sums, so that they each sum to one and the product is still $\overset{D}{C}$. We may repeat this for each of the $l-1$ terms in $\mathcal{C}_\omega$ to get the result shown. Note that this is the first term of z_d in (4.51).

We now deal with the remaining term in the second partial sum of R_{01} in (4.54). We first note the following useful relation:

$$\sum_{i_1=1}^{x} \cdots \sum_{i_2=1}^{x} \prod_{j=2}^{l} p_{i_l} \cdot c_\omega^+ = \sum_{m=1}^{x} c_m \cdot \left[\sum_{i_1=1}^{m} \cdots \sum_{i_2=1}^{m} \prod_{j=2}^{l} p_{i_j} - \sum_{i_1=1}^{m-1} \cdots \sum_{i_2=1}^{m-1} \prod_{j=2}^{l} p_{i_j} \right] \tag{4.57}$$

$$= \sum_{m=1}^{x} c_m \cdot \left[\left[P^m \right]^{l-1} - \left[P^{m-1} \right]^{l-1} \right]$$

So we have:

$$-\sum_{l=2}^{L} q_l \cdot \sum_{any\ i \geq d} \cdots \sum \prod_{j=2}^{l} p_{i_j} \cdot c_\omega^+ \tag{4.58}$$

$$= -\sum_{l=2}^{L} q_l \cdot \left\{ \sum_{i_1=1}^{D} \cdots \sum_{i_2=1}^{D} \prod_{j=2}^{l} p_{i_j} \cdot c_\omega^+ - \sum_{i_1=1}^{d-1} \cdots \sum_{i_2=1}^{d-1} \prod_{j=2}^{l} p_{i_j} \cdot c_\omega^+ \right\}$$

$$= -\sum_{l=2}^{L} q_l \cdot \left\{ \sum_{m=1}^{D} c_m \cdot \left[\left[P^m \right]^{l-1} - \left[P^{m-1} \right]^{l-1} \right] - \sum_{m=1}^{d-1} c_m \cdot \left[\left[P^m \right]^{l-1} - \left[P^{m-1} \right]^{l-1} \right] \right\}$$

$$= -\sum_{l=2}^{L} q_l \cdot \sum_{m=d}^{D} c_m \cdot \left[\left[P^m \right]^{l-1} - \left[P^{m-1} \right]^{l-1} \right]$$

We note that this is the second term of z_d in (4.51). So we finally have that R_{01} is:

$$R_{01} = \frac{b_d}{1-a_d} \cdot \sum_{l=2}^{L} q_l \cdot \left[1 - \left[P^{d-1} \right]^{l-1} \right] + \tag{4.59}$$

$$\sum_{l=2}^{L} q_l \cdot \left\{ (l-1) \cdot C - \sum_{m=d}^{D} c_m \cdot \left[\left[P^m \right]^{l-1} - \left[P^{m-1} \right]^{l-1} \right] \right\}$$

$$= y_d \cdot \frac{b_d}{1-a_d} + z_d$$

$$= J^{G_d}$$

so that (4.54) follows immediately. We have finished showing that the relative values given in (4.58) and the average cost given in (4.59) satisfy (4.53) for all states $\sigma \in \Sigma_0$.

Similarly, if $\sigma \in \Sigma_1$, so that $b=1$ packet is retained, we will have

$\omega = (0 \cdots 0\ i_l\ i_{l-1} \cdots i_1)$ with probability $q_l \cdot \prod_{j=d}^{l} p_{d_j}$ for any l from 1 to L. We see immediately that the form of all transition probabilities is the same as for $b=0$, but with l starting at 1 instead of 2. Applying this to the result (4.54) for $b=0$, we see that we need to show that:

$$J^{G_d} + \mathcal{C}_\sigma - c_\sigma{}^+ + \frac{b_d}{1-a_d} = \mathcal{C}_\sigma - c_\sigma{}^+ + \sum_{l=1}^{L} q_l \cdot \sum_{i_l=1}^{d-1} \cdots \sum_{i_1=1}^{d-1} \prod_{j=1}^{l} p_{i_j} \cdot \mathcal{C}_\omega + \sum_{l=1}^{L} q_l \cdot \sum_{any\ i \geq d} \cdots \sum \prod_{j=1}^{l} p_{i_j} \cdot \left[\mathcal{C}_\omega - c_\omega^+ + \frac{b_d}{1-a_d} \right] \tag{4.60}$$

$$= \mathcal{C}_\sigma - c_\sigma{}^+ + R_{11}$$

where we again use our previous notation R_{11} for the expected future cost of the $b=1$ discarding option. We notice that the term $\mathcal{C}_\sigma - c_\sigma{}^+$ is common to both sides of (4.60). We also notice again that the only difference in the summations from the $b=0$ case is that they start at 1 instead of 2, so we can immediately use our previous results for the terms to simplify our requirement to showing that:

$$J^{G_d} + \frac{b_d}{1-a_d} = \sum_{l=1}^{L} q_l \cdot l \cdot \overset{D}{C} + \frac{b_d}{1-a_d} \cdot \sum_{l=1}^{L} q_l \cdot \left[1 - \left[\overset{d-1}{P} \right]^l \right] - \sum_{l=1}^{L} q_l \cdot \sum_{m=d}^{D} c_m \cdot \left[\left[\overset{m}{P} \right]^l - \left[\overset{m-1}{P} \right]^l \right] \tag{4.61}$$

$$= R_{11}.$$

Rather than show this equality directly, we will re-use our efforts in the $b=0$ case as follows. We have:

$$R_{11}-R_{01} = \sum_{l=1}^{L} q_l \cdot \overset{D}{C} + \frac{b_d}{1-a_d} \cdot \sum_{l=1}^{L} q_l \cdot \left[\left[\begin{matrix} d-1 \\ P \end{matrix} \right]^{l-1} - \left[\begin{matrix} d-1 \\ P \end{matrix} \right]^{l} \right] - \tag{4.62}$$

$$\sum_{l=1}^{L} q_l \cdot \sum_{m=d}^{D} c_m \cdot \left\{ \left[\left[\begin{matrix} m \\ P \end{matrix} \right]^{l} - \left[\begin{matrix} m-1 \\ P \end{matrix} \right]^{l} \right] - \left[\left[\begin{matrix} m \\ P \end{matrix} \right]^{l-1} - \left[\begin{matrix} m-1 \\ P \end{matrix} \right]^{l-1} \right] \right\}$$

Now recalling from (4.38) and (4.39) the definitions of a_d and b_d, we have:

$$R_{11}-R_{01} = b_d + \frac{b_d \cdot a_d}{1-a_d} = \frac{b_d}{1-a_d} \tag{4.63}$$

Now utilizing the result from (4.59), we have that:

$$R_{11} = R_{01}+(R_{11}-R_{01}) = J^{G_\sigma} + \frac{b_d}{1-a_d} \tag{4.64}$$

which is equation (4.61), the desired result. We have thus completed the proof of the claims in (4.48) and (4.49) for the relative values and average cost per step for policy G_d. Since G_d was an arbitrary policy in the optimal family, the proof holds for all G_d in the optimal family.

4.4.1.4.2 Sufficient Conditions For Optimality

Recall from the dynamic programming optimality equation that policy G_d is optimal if and only if:

$$J^{G_d} + v_\sigma{}^{G_d} = \min_{g_\sigma} \left[\chi_\sigma{}^{g} + \mathbf{t}_\sigma{}^{g} \cdot \mathbf{v}^{G_d} \right]. \tag{4.65}$$

But in the present problem, there are only two possibilities for the bracketed quantity in (4.65), $\overset{D}{C} + R_{01}$ from (4.54) or $\overset{D}{C} - c_\sigma{}^{+} + R_{11}$ from (4.60), and we have already obtained expressions for these.

We first consider states $\sigma \in \Sigma_0$, so that the *lefthand* side of (4.65) is equal to $\overset{D}{C} + R_{01}$. We then can say that G_d is optimal for these states if and only if $\overset{D}{C} + R_{01} \le \overset{D}{C} + c_\sigma{}^{+} + R_{11}$, that is, $R_{11}-R_{01} = \dfrac{b_d}{1-a_d} \ge c_\sigma{}^{+}$. In words, this condition says, "If the penalty in expected future cost of retaining one packet exceeds the present cost advantage of retaining the highest priority

packet, then it is better to discard all packets." But since $\sigma \in \Sigma_0$, we know that $c_\sigma{}^+ \le c_{d-1}$, so G_d is optimal for $\sigma \in \Sigma_0$ if $\frac{b_d}{1-a_d} \ge c_{d-1}$. This is a sufficient, but not necessary, condition.

Similarly, G_d is optimal for $\sigma \in \Sigma_1$ if and only if $R_{11} - R_{01} = \frac{b_d}{1-a_d} \le c_\sigma{}^+$. In words, "If the penalty in expected future cost of retaining one packet is less than the present cost advantage of retaining the highest priority packet, then retain that packet." But for $\sigma \in \Sigma_1$, $c_\sigma{}^+ \ge c_d$, so a sufficient condition for G_d to be optimal is $\frac{b_d}{1-a_d} \le c_d$.

Combining these two results, we see that G_d is an optimal policy (for all states) if:

$$c_{d-1} \le \frac{b_d}{1-a_d} \le c_d. \tag{4.66}$$

This is the desired result.

4.4.1.4.3 Completeness of the Optimal Policy Family

We now show that one of the policies in the family $\{G_d : 1 \le d \le D\}$ is *always* optimal for any set of model parameters by showing that (4.37) *must* be satisfied for some d in the range 1 to D. That is, the optimal policy family is *complete*. We will first prove three separate results, and then use all three to prove that the optimal policy family is complete as described.

We first show that we *always* have:

$$0 = c_0 < \frac{b_1}{1-a_1} \tag{4.67}$$

by showing that both numerator and denominator are strictly positive. This is easy for the denominator since $a_1 = q_1 < 1$. Referring to the definition of b_1 in (4.39), we can show that $b_1 > 0$ by showing that the multiplier on c_m is less than p_m:

$$\left[\binom{m}{P}^{l} - \binom{m-1}{P}^{l} \right] - \left[\binom{m}{P}^{l-1} - \binom{m-1}{P}^{l-1} \right] \tag{4.68}$$

$$= \binom{m}{P-1} \cdot \binom{m}{P}^{l-1} - \binom{m-1}{P-1} \cdot \binom{m-1}{P}^{l-1}$$

$$= p_m \cdot \binom{m}{P}^{l-1} + \binom{m-1}{P-1} \cdot \left[\binom{m}{P}^{l-1} - \binom{m-1}{P}^{l-1} \right]$$

$$< p_m \cdot \binom{m}{P}^{l-1}$$

$$\leq p_m .$$

So (4.67) has been shown.

The second result is that the order relationship between $\frac{b_d}{1-a_d}$ and c_d must be the same as the order relationship between $\frac{b_{d+1}}{1-a_{d+1}}$ and c_d. Specifically, note from the definition of b_d in (4.39) that:

$$b_{d+1} - b_d = c_d \cdot \sum_{l=1}^{L} q_l \cdot \left[\binom{d}{P}^{l} - \binom{d-1}{P}^{l} - \binom{d}{P}^{l-1} + \binom{d-1}{P}^{l-1} \right]. \tag{4.69}$$

Also, note from the second form of a_d in (4.38) that:

$$a_{d+1} - a_d = \sum_{l=1}^{L} q_l \cdot \left[\binom{d}{P}^{l-1} - \binom{d}{P}^{l} - \binom{d-1}{P}^{l-1} + \binom{d-1}{P}^{l} \right] \tag{4.70}$$

so that:

$$b_{d+1} - b_d = -c_d \cdot (a_{d+1} - a_d) \tag{4.71}$$

or:

$$b_{d+1} + c_d \cdot a_{d+1} = b_d + c_d \cdot a_d \tag{4.72}$$

From this result we have that:

$$\frac{b_d}{1-a_d} = c_d \ <=> \tag{4.73}$$

$$b_d + c_d \cdot a_d = c_d \ <=>$$

$$b_{d+1} + c_d \cdot a_{d+1} = c_d \ <=>$$

$$\frac{b_{d+1}}{1-a_{d+1}} = c_d .$$

where $<=>$ means "if and only if." So we have:

$$\frac{b_d}{1-a_d} = c_d \ <=> \frac{b_{d+1}}{1-a_{d+1}} = c_d . \tag{4.74}$$

It is clear that the result holds if the = sign is replaced by $>$ or $<$.

The third result is that we always have:

$$\frac{b_D}{1-a_D} \leq c_D . \tag{4.75}$$

We first note that:

$$b_D = \sum_{l=1}^{L} q_l \cdot \left\{ \overset{D}{C} - c_D \cdot \left[1 - \left[\overset{D-1}{P} \right]^l - 1 + \left[\overset{D-1}{P} \right]^{l-1} \right] \right\} \tag{4.76}$$

$$= (1 - q_0) \cdot \overset{D}{C} - c_D \cdot \sum_{l=1}^{L} p_D \cdot (1-p_D)^{l-1}$$

and

$$a_D \cdot c_D = p_D \cdot c_D \cdot \sum_{l=1}^{L} q_l \cdot (1-p_D)^{l-1} . \tag{4.77}$$

It is then clear that:

$$b_D + a_D \cdot c_D = (1 - q_0) \cdot \overset{D}{C} < \overset{D}{C} < c_D \tag{4.78}$$

and (4.75) follows immediately.

We now put the three results together. From the first result (4.67), we know that $0 = c_0 < \frac{b_1}{1-a_1}$. If in addition, $\frac{b_1}{1-a_1} < c_1$, then G_1 is optimal. If instead, $\frac{b_1}{1-a_1} = c_1$, then the second result (4.74) implies that $\frac{b_2}{1-a_2} = c_1$, and both G_1 and G_2 are equally optimal. The final possibility is that $\frac{b_1}{1-a_1} > c_1$, in which case the second result (4.74) implies that $c_1 < \frac{b_2}{1-a_2}$. Again, if $\frac{b_2}{1-a_2} < c_2$, then G_2 is optimal. If instead, $\frac{b_2}{1-a_2} = c_2$, then both G_2 and G_3 will be optimal. And if $\frac{b_2}{1-a_2} > c_2$, then $c_2 < \frac{b_3}{1-a_3}$, and we begin the process again. Now suppose we have continued in this way and been unable to show that any of G_1 through G_{D-1} is optimal, so that we have $\frac{b_{D-1}}{1-a_{D-1}} > c_{D-1}$. Then the second result (4.74) implies that $c_{D-1} < \frac{b_D}{1-a_D}$, and the third result (4.75) guarantees that $\frac{b_D}{1-a_D} < c_D$, so that G_D is optimal.

Thus, for *any* set of model parameters, we are guaranteed that one of the policies in the family $\{G_d : 1 \le d \le D\}$ will be optimal, that is, the family is a complete family of optimal policies.

4.4.2 Systems Discarding Before Service Completion

Having accomplished the analysis for systems discarding after the service completion, the analysis for systems discarding before the service completion will be relatively straightforward. We first note that for these systems with $B = 1$, the queue capacity is $N = L$, since any retained packet will move into the server immediately after the discarding decision.

Next, our discussion in section 4.2.3 showed that the option of retaining $b = 0$ packets is guaranteed to be suboptimal, so the optimal policy is *immediately* evident: we always retain the single highest priority packet. This is policy G_1 from the previous section.

However, the relative values and expected cost must be calculated slightly differently than in the previous case. The reason is that the transition vector for the empty queue state (which has no choice but to retain zero packets) must be changed from its previous form to the *same* form as all the other transition vectors. That is, there is only one transition vector $\mathbf{t}_{11}$, hence only one expected future cost R_{11} for the *entire* system. This makes finding the relative values trivial, since we have:

$$J^{G_1} + v_\sigma = \xi_\sigma + R_{11} \tag{4.79}$$

for all σ. Setting $v_\sigma = 0$, we immediately get $J^{G_1} = R_{11}$ and hence $v_\sigma = \xi_\sigma = \mathcal{C} + c_\sigma{}^+$. It only remains to find the average cost J^{G_1}. The form for R_{11} will be the same as in (4.60), but all of the present costs will be of the form $\mathcal{C}_\omega - c_\omega^+$:

$$J^{G_1} = \sum_{l=1}^{L} q_l \cdot \sum_{i_l=1}^{D} \cdots \sum_{i_1=1}^{D} \prod_{j=1}^{l} p_{i_j} \cdot (\mathcal{C}_\omega - c_\omega^+). \tag{4.80}$$

Using (4.56) and (4.57) from our previous analysis, we immediately get:

$$J^{G_1} = \sum_{l=1}^{L} q_l \cdot l \cdot C^D - \sum_{l=1}^{l} q_l \cdot \sum_{m=1}^{D} c_m \cdot \left[\left[\frac{m}{P} \right]^l - \left[\frac{m-1}{P} \right]^l \right]. \tag{4.81}$$

This completes our analysis of $B = 1$ systems that discard before the service completion.

4.4.3 Performance Comparison

We now compare the performance of these $B = 1$ systems to their queue fill A/A counterparts. We first compare an optimal FS/Q system discarding after the service completion having dimensions $(N,L,D) = (3,2,3)$ to comparable queue fill A/A systems. Recall, however, that for this case of $B = 1$, the FS/Q system reduces to an FS/A system. To make a direct comparison, we again force the distribution of the number of arrivals in a service interval to be binomial. We take balanced costs $(c_1,c_2,c_3) = (1,2,3)$ and two skewed priority arrival probability distributions $(p_1,p_2,p_3) = (0.1, 0.3, 0.6)$ and $(p_1,p_2,p_3) = (0.6, 0.3, 0.1)$, so

that we will be adding the FS/Q system to Figure 3-8. The results are shown in Figure 4-11, where we see that the optimal queue fill A/A system performs slightly better than the optimal FS/Q system for all loads.

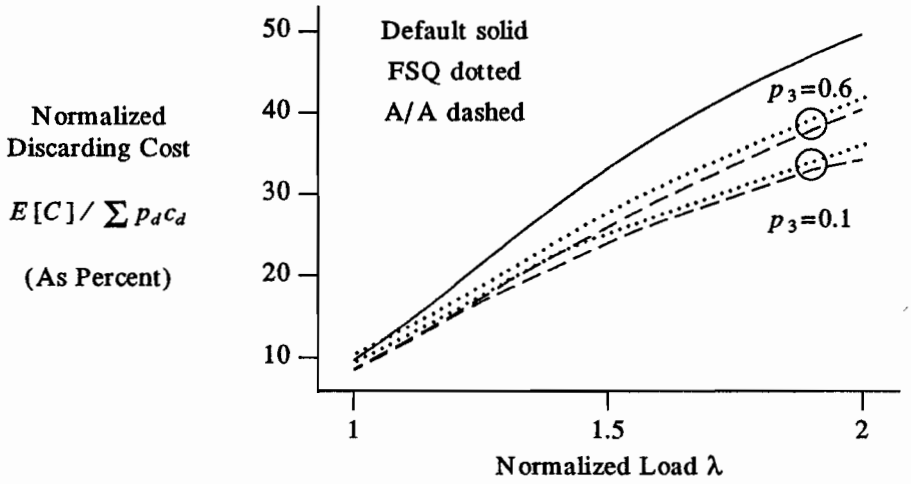


Figure 4-11. Performance Comparison of FS/Q and A/A Systems for $(N,L,D) = (3,2,3)$

This result indicates that the performance penalties which the FS/Q system sustains by 1) allowing some packets to enter the server with no opportunity to be discarded, 2) being able to retain only one packet and 3) being reduced to an FS/A system for $B=1$ outweigh the advantages of being able to defer the discarding decision, thereby gaining information about subsequent arrivals.

We cannot fairly compare the $B=1$ FS/Q system that discards before the service completion in Figure 4-11 since, for the same value of $L=2$, it requires only $N=L=2$ queue positions rather than $N=L+1=3$ positions. So in Figure 4-12, we compare an FS/Q system with $(N,L,D) = (2,2,3)$ that discards before the service interval with queue fill A/A systems of the same dimension. All other parameters are unchanged from Figure 4-11.

This time, even the default policy sometimes outperforms the optimal FS/Q policies! This is

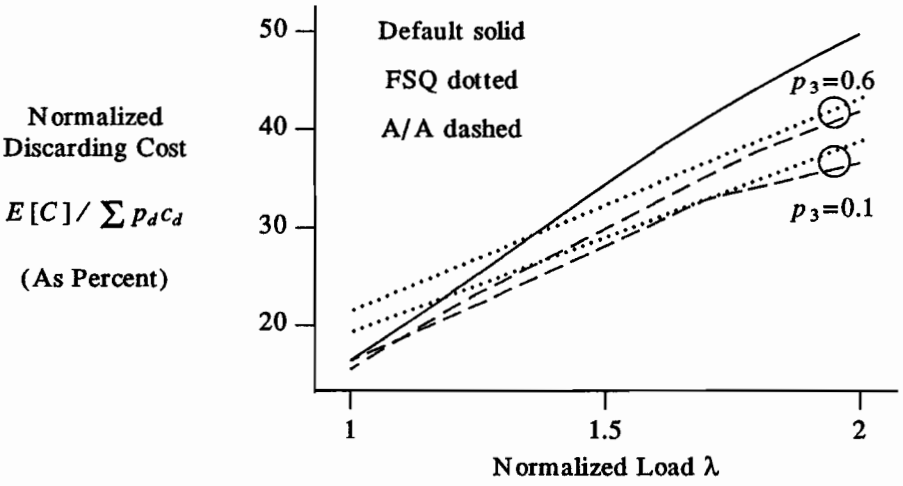


Figure 4-12. Performance Comparison of FS/Q and A/A Systems for $(N,L,D) = (2,2,3)$

not as surprising as it first seems since the latter is optimized only over FS/Q policies, but the default policy is an A/A policy. Again, the explanation lies in the disadvantage of the FS/Q (or FS/A) systems of having to reserve more queue positions for arrivals. For example, suppose that both of these systems begin a service interval with empty queues. If two priority D packets arrive in the service interval, the FS/Q (actually FS/A) system must discard one of them, but the default algorithm can keep them both. Of course, if there is an arrival in the first half of the next service interval, the default system will have to discard it, but under some combination of parameters, we see that it is indeed feasible for the default policy to have an advantage.

4.5 Summary of Results

We summarize the new contributions presented in this chapter, along with remaining open issues, in the following list.

1. We developed a specific analysis model for FS/Q discarding systems. For the two possible discarding instants for this model, we showed how to modify the analysis for systems discarding after the service completion to obtain an analysis for systems

discarding before the service completion.

2. We derived sufficient conditions for the irreducibility of any FS/Q policy. Restricting our search to such policies guarantees the existence of an optimal stationary policy.

3. We showed that the policy universe may be reduced by considering only candidate policies, but that even this reduced universe is generally quite large for this model.

4. We identified the form of every distinct transition vector in the transition matrix of any FS/Q policy.

5. Using these distinct transition vectors, we derived a computationally efficient method for value determination for FS/Q systems. The computational advantage of this method over the general method is substantial. This method can be used to speed up the policy improvement algorithm for finding an optimal policy for a given set of parameters.

6. We demonstrated that the performance of optimal FS/Q discarding policies is similar to the performance of the optimal queue fill A/A policies of Chapter 3, both of which significantly out-perform the default policy. We hypothesized that A/Q policy structures should perform better than these and that FS/A policies should perform worse, but left verification as an open issue.

7. We demonstrated that the performance improvement of increasing queue capacity diminishes significantly with increasing load, just as for A/A systems.

8. We performed a general analysis of FS/Q systems restricted to retain at most one packet. We identified a very small family of optimal policies, derived closed-form sufficient conditions for the optimality of each policy and closed-form expressions for the average discarding cost, and proved that the family is complete, that is, for any set of arrival statistics and discarding costs, one of the family members must be optimal.

9. We demonstrated that for systems with just two delivery priorities, it may be that a single specific policy is either optimal or very nearly optimal for every set of arrival statistics. The validity of this phenomenon for systems with more delivery priorities is left as an open issue.

4.6 References

[Luan88] D. T. Luan and D. M. Lucantoni, "Throughput Analysis of a Window-Based Flow Control Subject to Bandwidth Management," Proceedings of the IEEE Infocom Conference, New Orleans, March 1988.

[Pres86] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1986.

Chapter 5

Conclusions and Future Directions

CONTENTS

Chapter 5

LIST OF FIGURES

Chapter 5

Conclusions and Future Directions

We use this final chapter to draw some conclusions about the implications of this research for the system design of Integrated Packet Networks (IPNs) and to indicate open issues and possible future directions for continuing research.

5.1 Conclusions: Implications for System Design

We will explore in this section ways in which the results of the previous chapters could be applied to the system-level design of packet networks in general and Integrated Packet Networks (IPNs) in particular. We first examine broad packet switching issues to which our results could be applied and then focus on a few other promising applications.

5.1.1 The Overload Control Problem

In Chapter 1, we established that future IPNs, in particular Broadband Integrated Services Digital Networks (B-ISDNs), are likely to exhibit extremely bursty traffic characteristics which will pose significant design challenges. In particular, the problem of queue overload is intensified as the burstiness of the traffic increases for a fixed average load [Mura89].

One possible approach to controlling this overload problem is to operate a B-ISDN at a low average (or "engineered") traffic load. This is not a particularly attractive option from a revenue-generation point of view. This dissertation has shown that there is a viable alternative: priority packet discarding. We have shown that if the traffic can be separated into different delivery priorities and a cost associated with discarding packets of each priority, optimal

discarding policies can be found which significantly decrease the expected discarding costs for overloaded queues. Putting the same result in a different perspective, *for the same level of "discarding effect" or performance degradation, an optimal priority packet discarding system can support a significantly higher average load.* This effect is shown in Figure 5-1 in which the performance criterion is fixed (at $E[C]=0.35$), and we plot the maximum load that meets that criterion for an optimized discarding policy and the default policy for a wide range of priority arrival probabilities. We see that for the same level of performance, the optimized system can handle significantly more traffic.

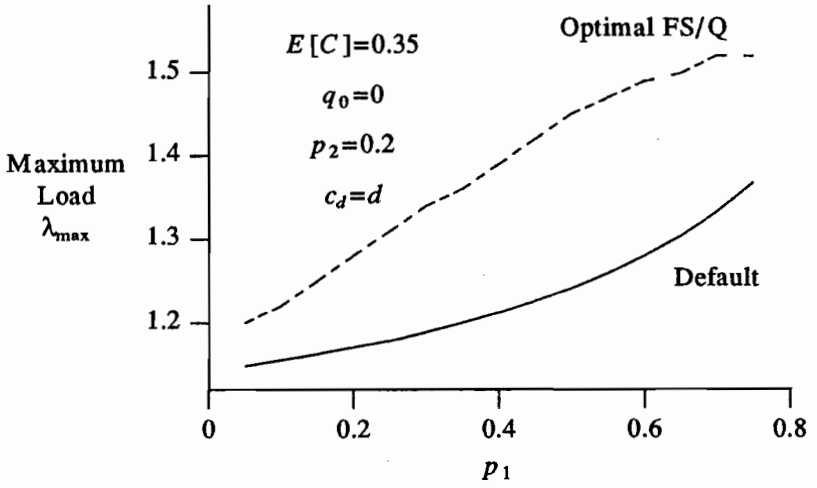


Figure 5-1. Load Comparison for $(N,L,D)=(3,2,3)$

In Chapters 1 and 2, we drew on previous research to show that this separation into delivery priorities and assignment of discarding costs is certainly feasible for speech and image/video traffic, which are envisioned to be the primary traffic types in B-ISDNs. Furthermore, the notion that some information may be lost is more palatable for these perceptually received signals.

5.1.2 Priority Discarding Options and Models

Furthermore, through our classification of priority packet discarding systems in Chapter 2, we showed that there are a number of options available to the system designer in terms of the structure of a priority discarding algorithm. Faced with this choice, we naturally wish to know how the different structures compare in terms of performance so that appropriate cost/performance tradeoffs can be made, if necessary.

To this end, we developed our general analysis model in Chapter 2 specifically with B-ISDN applications in mind. Other models might yield simpler analyses or more elegant solutions, but we believe that our model contains essential elements for B-ISDN system analysis. In particular, the model is based on fixed packet lengths and hence fixed (and synchronized) service intervals, an element that is essential in light of strong support for fixed packet lengths in the B-ISDN technical community. Our model also provides a useful means of characterizing the size or "dimensions" of a particular system, so that we have a convenient means of evaluating the effect of changing, for example, queue capacity for a given switch or, on a more global network planning scale, number of priorities.

5.1.3 The Superiority of Priority Discarding

Armed with these fundamentals, we have shown how the established technique of stochastic dynamic programming can be applied to the analysis of priority packet discarding in general, and to two specific structural classifications in particular. We began in Chapter 3 with simple systems that make a binary discarding decision on each packet as it arrives. We expect that these systems would be relatively simple to implement.

In spite of this simplicity we showed that, if properly optimized, these arrival-based systems can also significantly outperform a default system that discards packets if and only if the queue is full at the packet arrival instant. We further showed that the optimization can be

accomplished for these systems by making use of a computationally efficient recursive algorithm rather than the more general method called for in dynamic programming theory. We thus know that there exists a relatively simple discarding structure with a relatively simple means of optimization that can yield significant performance gains relative to non-prioritized discarding.

A system designer may want to know more than this. Could the performance be further improved by using a different discarding structure? We endeavored to answer this question in Chapter 4 by considering systems that defer the discarding decision until the time of service completion and are allowed to discard any packets waiting at that time. Since this type of discarding system has more information available at each decision point and more discarding flexibility, we might expect it to outperform the simpler arrival-based system. Undertaking the analysis of these latter systems, we again showed that their structure admits a computationally efficient method (though different from the method for the arrival-based systems) which can be used in the determination of optimal policies. Results of performance comparisons indicate that, in spite of our expectations, the two systems, each properly optimized, have essentially the same performance, at least for some sets of system parameters. The reason appears to be that this latter system must reserve more queue positions to accommodate arrivals while it is waiting to make its decision. We hypothesized that a structure that makes a discarding decision at every arrival instant, but which is allowed to discard any queued packets should perform the best; verification is left for future work.

5.1.4 System Size Considerations

A system designer may pose another question: how effective is additional queue capacity for improving performance? Using our previously developed analysis tools, we were able to reach two conclusions. First, it is possible for light overloads to get better performance with a large queue operating with the default (non-prioritized) discarding policy than with a small

queue operating with an optimized priority discarding policy. However, the advantage of additional queue capacity diminishes significantly with increasing load, so that for a heavily overloaded system, a small, optimized priority discarding system will outperform a default discarding queue of significantly larger size. The system designer's choice between priority discarding and large queues is thus largely influenced by the expected level of overload.

Since there are conditions under which additional queue capacity provides only marginal performance improvement, we undertook a more complete analysis of the deferred discarding system for the smallest possible system size. We were able to show that, if the discarding is done just after the service completion, one of a very small family of discarding policies must always be optimal for any parameters. If the discarding is done just before the service completion, there is always only a single optimal policy. We also found closed-form expressions for the discarding cost of each policy. These results combine to indicate that we could efficiently search through this family, if necessary, to find the optimal policy for a given set of parameters. The analysis provides useful experience with this class of systems, and the nice form of the result encourages further research in this area. We will also see that the results can be directly applied to an internal switch application.

The limited number of optimal discarding policies is also of particular interest for applications in which there are only two delivery priorities. Such a two-priority system has been proposed for bandwidth management in B-ISDN [Luan88] to control user access while allowing excess bandwidth to be used during uncongested periods. Our result in this case indicates that one of only two discarding policies must be optimal. We investigated this further and found that one of these two is either optimal or nearly optimal for a wide range of parameters. This is significant in light of an issue that we have not yet mentioned: policy adaptation. In general the optimal policy depends on parameters beyond the system designer's control. For optimal performance, then, we would generally have to be able to measure the

parameters and change the policy as the parameters change. But for these small systems with only two priorities, all of this would be unnecessary.

5.1.5 Other Applications

There is a potential application for which our small system assumption is not limiting. Many high-speed packet switch designs are based on an interconnected network of 2 by 2 switch elements, each with a buffer [Muis86, Jian89]. If our system has only two delivery priorities, or if it is possible to discard just before the service completion, it might be feasible to incorporate the single optimal discarding policy into the hardware design of each switch element. It would then be interesting to compare the expected discarding cost for the entire switch fabric with and without this optimal priority discarding.

We also indicated in Chapter 2 that this research has potential applications to traditional data networks, where one goal is the optimization of overall network throughput. In this case, the discarding cost of a packet could be proportional to the number of hops it has traversed in its path through the network. Our priority discarding policies would then be interpreted as preventing the loss of packets in which the network has invested significant resources. Whether or not using optimal discarding policies at each switch would optimize overall network throughput is another interesting research question.

Of course, application of this work is not limited to packet communication networks. We can consider the question of optimal discarding in any situation in which there is a finite, overloaded queue and arrivals that can be divided into different delivery priorities with different discarding costs. If a particular situation in another area of application should fit one of the models described here, our results could be directly applied.

5.2 Open Problems and Future Directions

We divide this final section into a subsection on interesting questions which have arisen in the course of the research and another on extensions to the analysis techniques, though the distinction is not always clear for a given item.

5.2.1 Questions To Be Answered

We noted in Chapter 3 that it is highly intuitive that a nested threshold form is required for a queue fill A/A policy to be optimal. We have been unsuccessful at proving this to date, but remain confident that it will be possible with further effort. We also note that previous implementations of nested threshold discarding have not distinguished the service interval "phases" as we have and have thus not allowed different threshold settings for the different phases. It would be interesting to compare the performance of these two approaches.

We have focused in this dissertation on expected discarding cost as the performance metric, but some applications may also require knowledge of the probability of discarding for each packet delivery priority. We effectively obtained this for the queue fill A/A systems by deriving a method for finding the state distribution, but made no attempt to find the state distribution or probability of discarding for the FS/Q analysis. We believe these derivations would be feasible within our model.

For FS/Q systems, we mentioned that maximal retention policies minimize current discarding costs and showed that, for systems retaining at most $B=1$ packet, the maximal retention policy appears to be always either optimal or very nearly optimal. We would like to formalize this result analytically and discover how it carries over into larger systems.

5.2.2 Extensions to Analysis Techniques

We noted in Chapter 3 that other forms for the queue state information are possible for A/A systems. Of particular interest would be systems that reserve a certain number of queue

slots for each priority (sharing with minimum allocation) or place a limit on the number of positions occupied by each priority (sharing with maximum queues), with default discarding applied to all other (shared) positions. The optimization in this case would be on the number of queue positions allocated to each priority. We believe it may be possible to better control the packet discarding probabilities for each priority with such systems. This may be important for applications in which the performance metric is more naturally expressed in terms of loss probabilities for each class of packet. For example, in the speech system considered in [Petr89], our results were stated in terms of maximum loss probability for each priority for indistinguishable speech quality.

Another way to control the discarding probabilities is with an extension to our current analyses to allow for time-varying discarding costs. For example, we can hypothesize a system that keeps its discarding history and increases the discarding cost for a given priority when some limit on discarding probability is approached.

We have noted that we believe the best discarding structure to be the one that makes a discarding decision for every arrival, but is allowed to discard any queued packets (A/Q). We have noted the potential difficulties with applying the present methods to this case, so an analysis of this type of system may well require a different sort of model than the ones we have considered in this dissertation. We do not believe that an analysis of systems discarding only arriving packets once per service interval (FS/A) would be worthwhile, since we would expect the performance of such systems to be worse than the FS/Q systems that we have already analyzed.

We have made the simplifying assumption in our general model that packet arrivals are uncorrelated. The dynamic programming method seems to allow for a relaxation of this restriction if we keep information on the most recent arrivals as part of the system state. This would be an important extension, since the packet discarding process itself is likely to produce

correlations in the traffic for downstream nodes.

We have noted earlier in this chapter that the optimal discarding policy generally depends on system parameters. If these parameters are subject to change, as they would be in any practical system, we would need to adapt our control policies to the changing parameters. Note that our previous idea of time-varying costs would require such a system. We would thus need to apply the theory of optimal stochastic adaptive control to our problem, so that we could compare the performance of adaptive policies to that of fixed policies. In the process, we would want to identify fixed policies that provide good, though perhaps not optimal, performance over a wide range of model parameters.

5.3 References

[Jian89] X. Jiang and J. S. Meditch "Integrated Services Fast Packet Switching," Proceedings of the IEEE Globecom Conference, Dallas, November 1989.

[Luan88] D. T. Luan and D. M. Lucantoni, "Throughput Analysis of a Window-Based Flow Control Subject to Bandwidth Management," Proceedings of the IEEE Infocom Conference, New Orleans, March 1988.

[Muis86] R. W. Muise, T. J. Schoenfeld, G. H. Zimmerman, "Experiments with Wideband Packet Technology," Proceedings of the 1986 International Zurich Seminar on Digital Communications, March 1986.

[Mura89] M. Murata, Y. Oie, T. Suda, and H. Miyahara, "Analysis of a Discrete-Time Single-Server Queue with Bursty Inputs for Traffic Control in ATM Networks," Proceedings of the IEEE Globecom Conference, Dallas, November 1989.

[Petr89] D. W. Petr, L. A. DaSilva, V. S. Frost, "Priority Discarding of Speech in Integrated Packet Networks," *IEEE Journal on Selected Areas in Communications,* Vol. 7, No. 5, June 1989.